Ловягин Никита Юрьевич

Операционные системы и оболочки

Презентации к курсу лекций
(учебное пособие)

Санкт-Петербург, 2019

Курс «Операционные системы и оболочки»

- Тематика курса.
  - Операционные системы (и оболочки): решаемые **задачи**, используемые **алгоритмы**, **принципы** разработки, архитектура.
  - На стыке:
    - **компьютеры** (вычислительные машины и системы) и сети, их архитектура и аппаратное обеспечение;
    - взаимодействие с **пользователем**;
    - разные смежные вопросы.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры, решения и проблемы.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры, решения и проблемы.

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

Компьютер

- Компьютер (computer, "вычислитель"), "вычислительная система". Одно из возможных определений компьютера (наиболее близкое к программированию): "*устройство для выполнения операций над данными в соответствии с инструкцией".*
- Компьютер:
  - **твердый продукт** (hardware, "железо") — аппаратное обеспечение;
  - **мягкий продукт** (software) — математическое (программное) обеспечение.
- Многообразие вычислительных машин и систем:
  - эволюция — от "больших машин" до микрокомпьютеров, изменение внешнего вида, типа устройств и др.
  - дифференциация — устройство различного назначения: бытовые компьютеры (настольные компьютеры, портативные компьютеры, смартфоны), встроенные компьютеры (телевизоры, плееры, холодильники, автомобили, самолеты), высокопроизводительные вычислительные системы (суперкомпьютеры, кластеры) и др.

Устройство компьютера: прикладной подход (1)

подход от решаемых задач,
взгляд "пользователя", "оператора"

- Устройства взаимодействия компьютера с человеком (управления компьютером, ввода и вывода информации от человека к компьютеру и обратно).
  - Устройства вывода:
    - передающие информацию в процессе работы: дисплеи, мониторы, индикаторы, динамики и др.;
    - вывод человекочитаемой информации на долгосрочный, независимый от компьютера, носитель: принтеры, плоттеры и др.
  - Устройства ввода:
    - прямой ввод информации человеком: кнопки, клавиатуры, манипуляторы, сенсоры, пульты, терминалы и др.;
    - получение информации из внешней среды: сканеры, камеры, микрофоны, датчики и др.
  - Деление на устройства ввода и вывода условно (индикаторы на клавиатуре, сенсорный дисплей и др).

Устройство компьютера: прикладной подход (2)

- Устройства хранения информации (в компьютерочитаемом виде):
  - внутренние (временный — оперативная память, перезаписываемый — различные носители, постоянный — встроенные данные);
  - внешние (перфокарты, магнитные ленты, дискеты, лазерные диски, флеш-накопители, карты памяти) — как правило перезаписываемые или постоянные.
- Устройства получения данных о внешней среде и самодиагностики:
  - датчики, камеры, микрофоны, сканеры.
- Устройства, управляемые компьютером
  - телевизоры, микроволновые печи, самолеты
  - могут сами снабжаться датчиками.
- Средства подключения внешних устройств:
  - приводы (дисководы, оптические);
  - разъемы, кабели, переходники.
- Средства взаимодействия с другими компьютерами:
  - проводные (разъемы, кабели, сети, сетевое оборудование),
  - беспроводные (радио, ИК).
- Другие внешние устройства.

Устройство компьютера: архитектурный подход (1)

подход от математической организации,
взгляд "программиста", традиционная "схема ЭВМ"

- Центральное процессорное управление:
  - устройство управления (декодирование и выполнение команд);
  - арифметико-логическое устройство (выполнение вычислений).
- Постоянно-запоминающее устройство:
  - встроенное ПО, "прошивка", программа, выполняющаяся при включении компьютера;
  - на самом деле на многих современных компьютерах может быть перезаписываемым;
  - примеры — для "обычных" компьютеров: BIOS, UEFI, многообразное собственное ПО для различных устройств, единственное ПО некоторых устройств (калькуляторы, встроенные компьютеры, телефоны, устройства ввода-вывода).

Устройство компьютера: архитектурный подход (2)

- Оперативно-запоминающее устройство
  - условно: хранит код исполняемых программ и их рабочие данные.
- Устройства ввода-вывода:
  - клавиатуры, мониторы, принтеры, жесткие и SSD диски, оптические приводы, флеш-накопители, средства чтения смарт-карт, датчики, камеры и т. д.;
  - архитектурно являются отдельными компьютерами со своим набором компонент данной классификации, взаимодействие с которыми осуществляется путем обмена данными.
- Средства коммуникации компонентов
  - **шина** (система передачи данных внутри компьютера)
  - разъемы, кабели, переходники, сети и т.д.

Устройство компьютера: аппаратный подход (1)

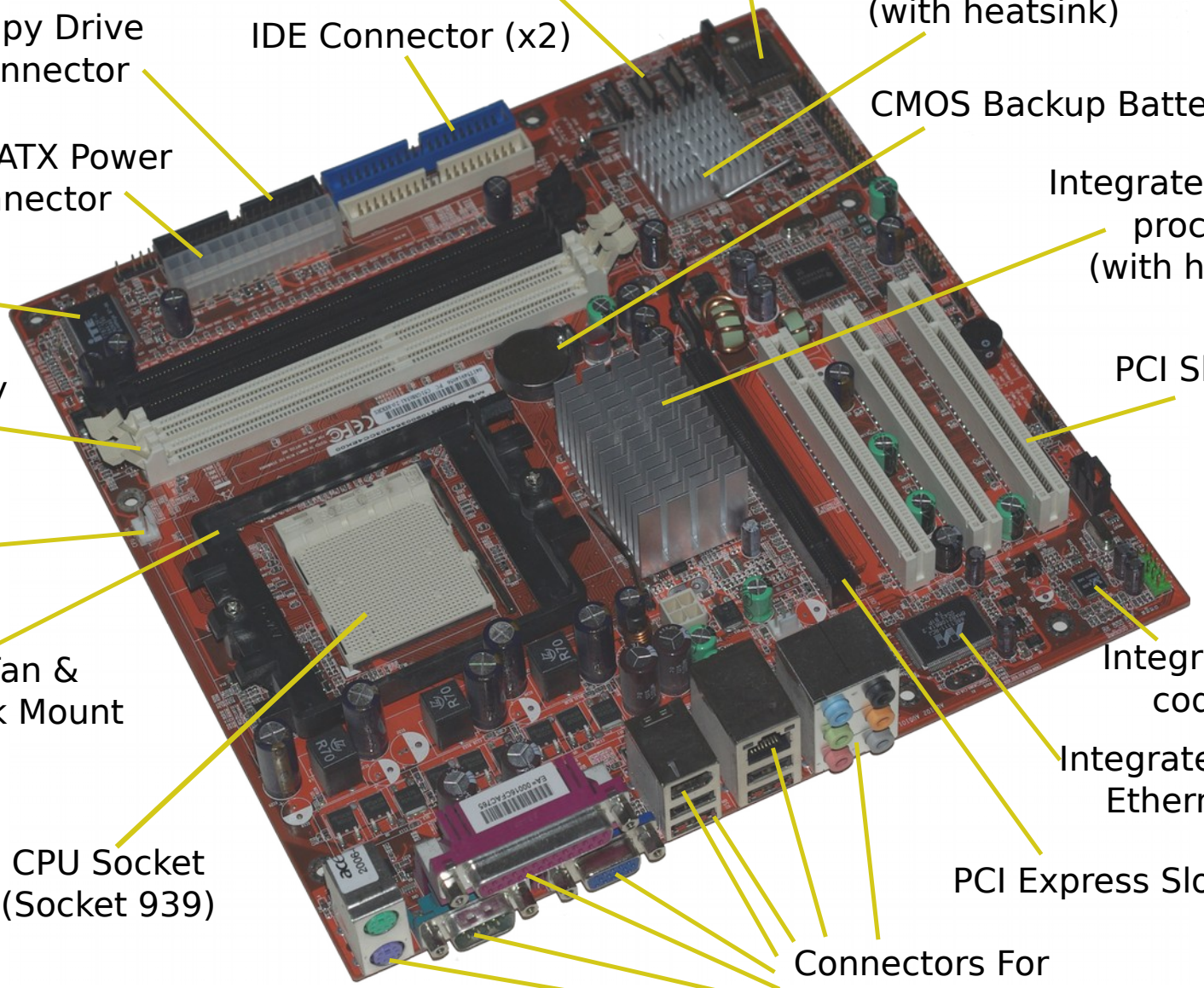
подход от аппаратных частей компьютера
взгляд "инженера", "сборщика"
то, что из чего "на самом деле" состоит компьютер
то, что можно потрогать

- Собственно устройства:
  - корпус, блок питания, системы охлаждения и др.;
  - **главная ("материнская", "родительская", "системная") плата**
    - содержит устройства системы и средства расширения системы (подключения других устройств) — разъемы;
  - центральный процессор (CPU, Central Processor Unit)
    - может быть одиночный (одно или многоядерный), может быть несколько процессоров;
  - оперативная память;
  - средства подключения внутренних и внешних устройств:
    - различные разъемы, карты, кабели, переходники: видеокарта (имеет свой процессор — **видеопроцессор**, GPU, Graphical Processor Unit, — и разъем подключения монитора), звуковая карта и т. п.,
    - могут подключаться или быть встроенными (в материнскую плату, в процессор).

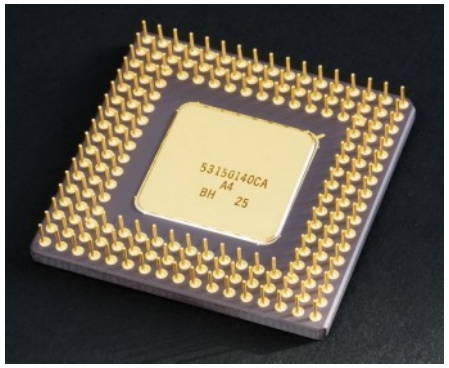
Устройство компьютера: аппаратный подход (2)

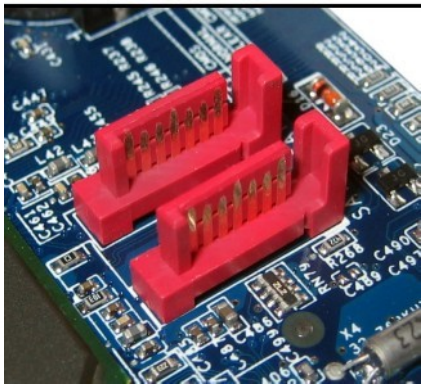
- Собственно устройства (продолжение):
  - внешние (**периферийные**) устройства
    - мониторы, принтеры, клавиатуры, мыши, внешние накопители и т. д.,
    - сюда же: сетевое оборудование (маршрутизаторы, свитчи и т. д.),
    - подключаются к разъемам корпуса напрямую или посредством кабелей;
  - внутренние устройства
    - внутренние накопители, приводы, мониторы, кнопки и т. д.,
    - подключаются к разъемам внутри корпуса напрямую или посредством кабелей.
- Материальные компоненты и физические среды
  - основа: материал корпуса, платы, держатели (в разъемах) и т.д.;
  - электрика: проводники (на платах, провода в кабелях), трансформаторы, моторы (для дисков, принтеров), изоляция и др.
  - физические носители информации (магнитные и лазерные диски, твердотельные накопители) и среды передачи информации (электрический ток в кабелях, электромагнитные волны в оптоволоконных кабелях и внешней среде, звуковые волны и т. д..

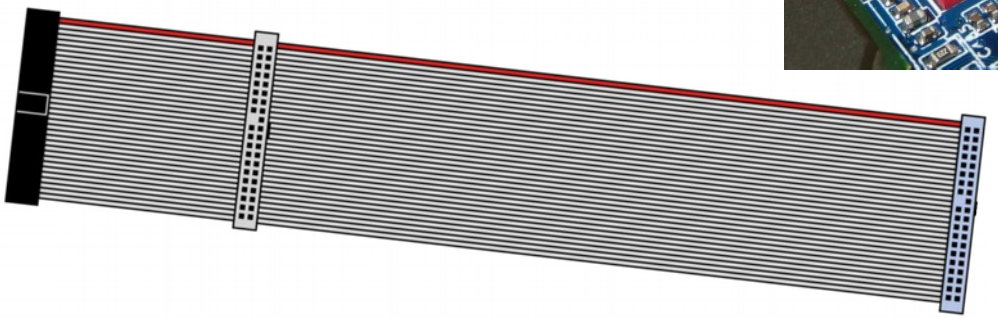
Материнская плата



Аппаратные компоненты







Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.

➔ Некоторые базовые определения и понятия.

- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

Базовые определения (1)

- **Операционная система (Operation System, ОС)**
  - программное обеспечение, предоставляющее базовый интерфейс между приложениями и аппаратурой компьютера.
- **Оболочка (shell)**
  - пользовательский интерфейс для доступа к функциям операционной системы (часть ОС или отдельный продукт?).
- **Интерфейс**
  - совокупность средств, с помощью которых компоненты компьютерной системы могут взаимодействовать друг с другом:
    - **аппаратный** — для взаимодействия оборудования;
    - **внутренний** — программный (Application Programming Interface, API), бинарный (Application Binary Interface, ABI) и т.д.,;
    - **пользовательский** — командный, текстовый, графический и т. д.;
    - др.
- **Машинные команды**
  - команды, интерпретируемые непосредственно устройством управления;
  - различные вычислительные системы имеют различный набор машинных команд.

Базовые определения (2)

- **Архитектура вычислительной системы**
  - организация конкретной вычислительной системы или ее компонента (i686, x64, ARMv6 для процессоров, определяет различный набор машинных команд и не только);
  - математическая концепция, лежащая в основе такой организации (**архитектура фон Неймана**, гарвардская архитектура).
- **Процесс**
  - по существу: программа во время ее выполнения.
- **Ресурс**
  - объект вычислительной системы, который может быть использован процессом: **аппаратные ресурсы** (процессоры, коммуникационное оборудование, память, внешние устройства и др.), **программные ресурсы** (процессы, средства коммуникации, виртуальная память, файловая система и др.).
- **Уровень абстракции**
  - набор возможностей, реализованных аппаратно или посредством других возможностей, принадлежащих абстракциям более низкого уровня.

Основные функции ОС

- Предоставление программного интерфейса для доступа к аппаратному обеспечению компьютера (абстракция и виртуализация)
  - "удобного" ("красивого", более высокоуровневого, более "простого" в использовании, чем аппаратный),
  - "унифицированного" (один программный интерфейс вместо различных аппаратного — например, общий файловый интерфейс вместо различных интерфейсов для разных типов носителей информации и файловых систем),
  - "безопасного" (разграничение доступа к аппаратуре для пользователей и программ).
- Управление **ресурсами** компьютера
  - "распределение" ресурсов между процессами и пользователями.
- Организация безопасности работы пользователей и программ.

Компоненты ОС

- **Ядро операционной системы**
  - модули, предоставляющие основные функции операционной системы (управление аппаратными и программными ресурсами).
- **Системные библиотеки**
  - предоставляют базовый программный интерфейс ОС.
- **Вспомогательные элементы**
  - стандартные системные приложения,
  - приложения по настройке и управлению ОС,
  - др.
- **Оболочки.**

С какой стороны смотреть на ОС?

- Пользователь
  - пользовательский интерфейс: Command Line Interface (CLI), Text User Interface (TUI), Graphical User Interface (GUI).
- Разработчик прикладного ПО
  - программный интерфейс (API, ABI).
- Разработчик оборудования
  - ядро, драйвера (HAL).
- Разработчик ОС.

Сложность определения ОС

- Где заканчивается ОС и начинаются прикладные программы?
  - стандартные приложения ОС?
  - оболочки?
  - системное ПО и ПО, замещающее стандартное, других производителей?
  - драйвера устройств?
- Решение принимает производитель (случаются конфликты: ОС linux или linux — ядро, ОС — GNU/linux?).

Базовые определения (3)

- **Функция (программирование)**
  - поименованный набор программных инструкций, выделенный в самостоятельную единицу; может быть частью программы, библиотеки и др.
- **Системный вызов**
  - вызов функции ядра операционной системы из прикладной программы (по сути аналогичен вызову библиотечных функций, но обычно исполняется другими машинными командами);
- **Режимы процессора: привилегированный режим (режим ядра, режим супервизора)** и **режим пользователя**
  - в режиме пользователя доступно лишь безопасное подмножество машинных команд;
  - прикладные программы работают в режиме пользователя, доступ к оборудованию осуществляется с помощью системных вызовов, проверяющих уровень доступа, безопасность и т.п., и переключающихся в режим ядра при необходимости;
  - поддерживается "большинством" современных процессоров.

Виртуализация

- **Абстрагирование от аппаратной реализации.**
  - позволяет работать с ресурсами в удобном виде, а не в таком, какие они на самом деле (виртуализация реальных ресурсов), повышает уровень абстракции.
- **Создание виртуальных ресурсов**. Примеры:
  - виртуальный оптический привод для подключения файл-образа;
  - **виртуальное адресное пространство процесса** вместо реальной оперативной памяти (каждый процесс работает в своем виртуальном адресном пространстве), часть данных может сохраняться на диске, а не в оперативной памяти — **виртуальная память**;
  - **многозадачность** (кажущееся одновременное выполнение нескольких процессов, превышающее реальное число процессоров).
- **Имитация виртуальных аппаратных ресурсов одной вычислительной системы посредством реальных аппаратных ресурсов другой вычислительной системы:**
  - **эмулятор игровой приставки;**
  - **виртуальная машина** (запуск **гостевых ОС** на виртуальном оборудовании) и др..

Место ОС

- Программное обеспечение.
  - Пользовательский режим:
    - прикладные программы,
    - оболочка ОС,
  - Режим ядра:
    - операционная система
- Аппаратное обеспечение.

Абстрагирование от аппаратного интерфейса (пример)

- Абстракция ОС: файлы.
- Драйвер устройства: чтение и запись блоков данных.
- Интерфейс SATA: 450+ страниц документации.

Управление ресурсами

- Процессы конкурируют за ресурсы, ОС распределяет ресурсы между процессами.
- Ресурсы могут быть
  - монопольно используемые (принтер, сканер, CD-disk при записи, аудиосистема и др.);
  - разделяемые (спулер принтера, аудиомикшер, файл при чтении и др);
  - повторно выделяемые (процессорное время, оперативная память и др).

Безопасность

- Защита данных и устройств от повреждения и несанкционированного доступа: ограничение прав пользователей.
- В режиме пользователя прямой доступ к аппаратуре закрыт, есть доступ к системный вызовам.
- Системный вызов проверяет наличие прав на запрашиваемое действие.

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

История ОС

- **Первое поколение (1945–1955): электронные лампы**
  - "прямое" программирование контактных схем, машинные коды.
- **Второе поколение (1955–1965): транзисторы**
  - системы пакетной обработки: более дешевый вспомогательный компьютер прочитывал перфокарты и записывал машинный код на магнитную ленту, более дорогой исполнял их.
- **Третье поколение (1965–1980): интегральные схемы**
  - IBM (OS/360: многозадачность, CTSS: у каждого пользователя свой диалоговый терминал, **MULTICS**: "одна машина на весь город").
  - DEC (Компьютеры серии PDP, ОС UNIX, стандарт POSIX).
- **Четвертое поколение (с 1980 года): персональные компьютеры**
  - Intel (процессор), дисковая ОС **CP/M**, IBM PC, Microsoft DOS;
  - GUI (Xerox PARC, Apple);
  - Microsoft: Windows (оболочка DOS), Windows NT (отдельная ОС);
  - FreeBSD, GNU/Linux, X11.
- **Пятое поколение (с 1990 года): мобильные компьютеры**
  - Встроенные ОС;
  - КПК: Windows CE, Palm OS;
  - Смартфоны: Symbian OS, Google Android, Apple iOS, Windows Phone.

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

Классификация ОС:
по особенностям аппаратных платформ

- ОС мейнфреймов (больших универсальных высокопроизводительных машин): OS/390, *nix;
- Серверные ОС: Solaris, FreeBSD, Windows Server, Debian, RHEL.
- ОС персональных компьютеров: GNU/Linux, Windows 7-8-10..., FreeBSD, OS X.
- ОС мобильных устройств: Android, Symbian, iOS, Windows Mobile.
- Встроенные ОС (бытовые приборы, цифровые приборы) : Embedded Linux, QNX и VxWorks.
- ОС сенсорных узлов: TinyOS.
- ОС систем реального времени: FreeRTOS, Vxworks, QNX, eCos (плюс отчасти системы смартфонов).
- ОС отказоустойчивых систем: OpenVMS.
- ОС смарт-карт (JVM).

Классификация ОС:
по особенностям аппаратных платформ

- Различие в задачах
  - мощные вычисления vs мобильное устройство
- Различие в интерфейсных устройствах
  - большой экран vs маленький экран vs простейший дисплей
  - клавиатура vs сенсорный экран
- Различие в питании
  - сеть vs аккумулятор vs индукция
- Требование реального времени
  - отсутствует vs слабое vs жесткое
- Отказоустойчивоть vs продвинутые возможности

Классификация ОС: по особенностям алгоритмов управления ресурсами (наличие или отсутствие поддержки)

- Многозадачность
- Многопоточность
- Многопроцессорность
- Многопользовательский режим

Классификация ОС: по задачам целевого оборудования

- Пользовательские (Windows, linux, Symbian, Tizen)
- Серверные (linux, Windows Server)
- Пакетные (IBM 360, linux)
- Управления оборудованием (встроенные и др.) — ?

Классификация ОС: по типу (режиму) среды исполнения

- **Интерактивные системы**: выдача ответов на внешние запросы (пользователей) — персональные компьютеры и др..
  - Время обработки запросов должно быть приемлемым.
- **Пакетные системы**: осуществление большого объема вычислений (научные расчеты, обработка данных и др.) — суперкомпьютеры, кластеры и т.д..
  - Достижение наибольшей производительности оборудования (объема решаемых задач в единицу времени).
  - Минимизация простоев оборудования.
- **Системы реального времени**: выполнение задач (периодических, апериодических, запросов) до достижения крайних сроков — системы управления оборудованием и т.п..
  - Предсказуемость времени решения задачи (обработки запроса).
  - Достижение требуемого уровня качества обработки в отведенный промежуток времени (время важнее точности).

Системы реального времени

- **Жесткое реальное время**: 100% запросов должны быть обработаны до достижения крайнего срока.
  - Возможны "отрицательные" результаты обработки: ухудшение качества (например, изображения при трансляции), отказ в выполнении запроса (сигнал "занято" на телефонной линии при де-факто свободном абоненте и т.п.)
- **Мягкое реальное время**: большинство запросов должны быть обработаны до достижения крайнего срока.
  - Возможны срывы сроков (задержка изображения, поздняя реакция на запрос) или "отрицательные" результаты обработки.
- *Интерактивное реальное время* (строго говоря не является СРВ)
  - Интерактивная система, в которой время обработки запроса не ощутимо пользователем.
- Последствия отрицательных результатов и срывов сроков:
  - катастрофические (взрыв реактора, падение самолета);
  - падение стоимости ответа (задержка получения данных);
  - обнуление стоимости ответа (сигнал "занято").

Классификация ОС: по типу связности

- **Централизованные** (локальные): управляют ресурсами одного компьютера
  - однопроцессорные,
  - многопроцессорные.
- **Сетевые**: виртуальная вычислительная система, предоставляющая доступ к сетевым ресурсам, но для пользователя локальные и сетевые ресурсы разделены.
- **Распределенные**: виртуальная вычислительная система, прозрачно объединяющая сетевые ресурсы в единую систему.

Классификация ОС: по типу архитектуры ядра системы

- Монолитное ядро
  - одна программа, работающая в режиме ядра, использует быстрые переходы между функциями.
- Модульное ядро
  - ядро состоит из компонентов, которые можно подключать к нему.
- Слоистое ядро
  - компоненты образуют уровни, выполняющие отдельные задачи различного уровня абстракции: железо, планирование процессов, управление памятью, управление вводом-выводом, интерфейс пользователя.
- Микроядро
  - выполняет минимум функций по управлению аппаратурой, функции более высокого уровня выполняются другими компонентами.
- Наноядро
  - обрабатывает только аппаратные прерывания
- Экзоядро
  - все отдается пользовательским программам

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- → Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - → сложности разработки ОС.
- Особенности курса "Операционные системы и оболочки".

Сложности разработки ОС

- **Трудоемкий процесс**
  - миллионы строк кода, тысячи человеко-лет разработки (для универсальных ОС).
- **Сложный процесс**
  - требует знания об аппаратуре, безопасности, современных представлениях об интерфейсе, алгоритмах, представлении данных и т.д.
  - требует учета различных факторов (производительность, безопасность, ориентированность на пользователя, многопользовательский доступ, многозадачность, параллелизм, совместимость, многоплатформенность), информации о целевом устройстве (процессор, память, контроллеры, тип устройства — ПК, телефон, телевизор, АЭС, самолет, космическая станция).
- **Операционных систем *"мало".***

Компьютеры и операционные системы

- Представление об устройстве компьютера с разных точек зрения.
- Некоторые базовые определения и понятия.
- Операционные системы:
  - обзор истории ОС;
  - классификация ОС;
  - сложности разработки ОС.

➔ Особенности курса "Операционные системы и оболочки".

Основное содержание курса

Задачи				
Вычисле-ния	Передача данных	Хранение данных	Обработка данных	Ввод и вывод
Функции ОС и оболочек				
Процессы	Комму-никации	Файлы	Виртуальн. адресное простран-ство	Оболочки
Аппаратная платформа				
Процес-соры	Коммуни-кационное обору-дование	Накопи-тели инфор-мации	Память	Интер-фейсное оборудо-вание

Сложность курса ОС

- **Большинство знаний быстро устаревает**
  - конкретные программные решения (2-3 года),
  - архитектура и операционная платформа (5-7 лет),
  - основные идеи организации ОС (7-10-15 и более лет, циклически),
  - фундаментальные знания, математические основы (десятки лет).
- В курсе сделан упор на основные идеи и фундаментальные знания с отсылкой на конкретные аппаратные и программные решения.
  - Общие знания — базис для самообразования (в настоящее время и в будущем), фундамент для создания новых конкретных решений.
- **Литература**

  Э. Таненбаум, Х. Бос

  "Современные операционные системы"

  3-е издание, 2010 г.

  4-е издание, 2015 г.

  …
- **Много терминов и устоявшихся аббревиатур**, в т. ч. исключительно англоязычных.
  - Знание, важное для ориентирования в мире ОС и компьютерной аппаратуры (профессиональный язык).
  - В курсе дается лишь малая часть — чаще всего это конкретные решения.

Общий план курса

- Введение: компьютеры и операционные системы.
- ➔ Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Процессоры (1)

- Основа ВМ (чаще всего) — архитектура Фон Неймана:
  - исполнение команд: последовательно друг за другом (процессор);
  - память: однородная, адресуемая (оперативная память);
  - устройство ввода вывода (отдельный компьютер);
  - шина.
- Процессор: "аппаратная поддержка процессов".
- Компоненты процессоров (как устройств):
  - устройство управления (декодирование и исполнение команд),
  - арифметико-логическое устройство (выполнение вычислений),
  - регистры (арифметические, управления, состояния),
  - кэш (одноуровневый, многоуровневый),
  - другие компоненты (математический сопроцессор, видеопроцессор, и т.д.)

Процессоры (2)

- **Регистры** — сверхбыстрая ячейка памяти процессора, адресуется битовым кодом в структуре или аргументе машинной команды или по имени в мнемокоде (ассемблерном коде) машинной команды.
- Процессы различной архитектуры имеют различное количество и разрядность регистров.
  - Арифметические регистры: производят вычисления (AX, BX,…, EAX, RAX, ...).
  - Регистры управления: указатель кода, указатель стека и др (IP, SP, CS, DS).
  - Регистры состояния (флаги): режим процессора, наличие ошибки, переполнения и др.
  - Другие регистры.
- **Кеш** (общее понятие) — программный или аппаратный буфер, хранящий информацию для более быстрого доступа, чем при использовании обычного способа хранения (более быстрая память) или получения (сохранение удаленных или вычисленных данных).
- **Кеш процессора** — аппаратный кеш оперативной памяти.
  - Сохраняет данные из ОЗУ для более быстрого доступа.
  - Как правило сохранение производится процессором автоматически и не контролируется программно.
  - Модели CPU различаются размером и количеством уровней кеша.

Процессоры (3)

- **Архитектура**: индивидуальный набор машинных команд, регистров и др. возможностей процессора (способа доступа к памяти, виртуализации и т.п.). Различают
  - принципиально разные архитектуры (несовместимы между собой);
  - модификации одной архитектуры (могут добавляться расширенные команды, дополнительные регистры и т.п.).
- Примеры архитектур: 8086, x86; ARM; SPARC (Sun); Apollo Guidance Computer; Intel 4004.
- Быстродействие:
  - тактовая частота (гц) — количество сигналов синхронизации вычислительной системы в секунду (одна операция выполняется за несколько тактов);
  - другие свойства ЦПУ — размер кеша, сопроцессоры, внутренний параллелизм (количество и сложность операций, выполняемых за группу тактов) и т. п.;
  - скорость вычислений (FLOPS) — количество операций с плавающей точкой в секунду;
  - быстродействие вычислительной системы в целом — количество процессоров (ядер), размер ОЗУ, частота шины, размер и быстродействие накопителей, свойства других компонентов, полнота использования (алгоритмы).

Расширения архитектуры фон Неймана

- **Аппаратные прерывания**
  - Вызов обработчика по сигналу устройств ввода-вывода.
- **Обеспечение многопользовательской, многозадачной среды**:
  - защищенный режим (с 80268);
  - страницы памяти (с 80386).
- **Внутренний параллелизм**:
  - конвейерные вычисления (с 8086) — выполнение разных шагов обработки инструкции на разных узлах;
  - суперскалярность (с Intel Pentium) — исполнение несколько инструкций одновременно (параллелизм на уровне инструкций);
  - векторизация (с Intel Pentium MMX) — обработка массивов данных за одну инструкцию.
- **Параллелизм**:
  - многоядерность (с Intel Core),
- **Операции с плавающей точкой**:
  - особые регистры и инструкции (сопроцессор или встроенные).

Параллельные вычислительные системы

- **Таксономия Флинна**

поток данных (D) поток инструкций (I)	Single (один)	Multiple (множественный)
Single (один)	SISD (нет параллелизма)	SIMD (**вектор**)
Multiple (множественный)	MISD (**конвеер**)	MIMD (универсальный)

- **Память**
  - **Общая** (Shared, разделяемая); **распределенная** (Distributed, раздельная): передача по шине (быстрая), передача по сети (медленная).
- Современные параллельные компьютеры:
  - многопроцессорные компьютеры (SM-MIMD);
  - многоядерные (на одном кристалле) процессоры (SM-MIMD);
  - векторные процессоры (GPU — SM-SIMD или DM-SIMD);
  - компьютерные кластеры (DM-MIMD);
  - суперкомпьютеры (SM-SIMD, SM-MIMD);
  - распределенные вычисления (облака — DM-MIMD).

Конвейер

```
процессоры: 1, 2, 3, ...
данные:     x1, x2, x3, ...
команды:    f1, f2, f3, ...
```

	проц. 1	проц. 2	проц. 3	проц. 4	
шаг 1	f1(x1)	простой	простой	простой	...
шаг 2	f1(x2)	f2(f1(x1))	простой	простой	...
шаг 3	f1(x3)	f2(f1(x2))	f3(f2(f1(x1)))	простой	...
шаг 4	f1(x4)	f2(f1(x3))	f3(f2(f1(x2)))	f4(f3(f2(f1(x1))))	...
шаг 5	f1(x5)	f2(f1(x4))	f3(f2(f1(x3)))	f4(f3(f2(f1(x2))))	...
...					

(команды последовательной обработки разных данных могут отличаться)

Вектор

```
процессоры: 1, 2, 3, ..., N
данные:     x1, x2, x3, ..., xM (M<=N)
команды:    f1, f2, f3, ...
```

	проц. 1	проц. 2	проц. 3	проц. 4	...	проц. M	проц. M+1	...	проц N
шаг 1	f1(x1)	f1(x2)	f1(x3)	f1(x4)	...	f1(xM)	простой	...	простой
шаг 2	f2(x1)	f2(x2)	f2(x3)	f2(x4)	...	f2(xM)	простой	...	простой
шаг 3	f3(x1)	f3(x2)	f3(x3)	f3(x4)	...	f3(xM)	простой	...	простой
...									

(команды на разных процессорах не могут отличаться)

Конвейерный режим

- Инструкция выполняется (как минимум) в 5 тактов: выборка (IF), декодирование (ID), обращение к памяти (MEM), исполнение (EX), запись результата (WB).
- Конвейер:

	т 1	т 2	т 3	т 4	т 5	т 6	т 7	т 8	т 9
и 1	IF	ID	MEX	EX	WB				
и 2		IF	ID	MEM	EX	WB			
и 3			IF	ID	MEM	EX	WB		
и 4				IF	ID	MEM	EX	WB	
и 5					IF	ID	MEM	EX	WB

- Характеризуются размером конвейера (5-30+).
- Проблемы:
  - конфликты данных (зависимость инструкции от результата предыдущей инструкции),
  - конфликты управления (при условных переходах).

Разрешение конфликтов

- **Конфликты данных**: перепланирование (в компиляторе):

  Код (высокоуровневый):

  a = b + c;
  d = e - f;

  Прямое ассемблирование:

  - `MOV RC, [b]`
    `MOV RC, [c]` *(простой)*
    `ADD [a], RB, RC`

    `MOV RE, [e]`
    `MOV RF, [f]` *(простой)*
    `SUB [d], RE, RF`

  2 простоя

  Перепланирование:

  `MOV RB, [b]`
  `MOV RC, [c]`
  `MOV RE, [e]`

  `ADD [a], RB, RC`
  `MOV RF, [f]`
  `SUB [d], RE, RF`

  без простоев

- **Конфликты управления**: предсказания переходов
  - в компиляторе,
  - в процессоре.

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- ➔ Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Процессы и потоки

- Процесс (или задача): экземпляр выполняемой программы — исполнение последовательности команд в операционной среде, включающей собственно выполняющуюся программу, а также связанные с ней данные и состояния (открытые файлы, текущий каталог и т. п.).
  - С точки зрения ОС: единица работы, заявка на потребление системных ресурсов.
  - С точки зрения аппаратной платформы: объект, которому выделяется процессорное время.
- Обычно в многозадачной среде исполняется в отдельном **виртуальном адресном пространстве** (не видят данные друг друга).
- Поток (или нить): части одно процесса, набор машинных команд, исполняемых последовательно, потоки одного процесса исполняются в одном виртуальном адресном пространстве.

Процесс

- В памяти (виртуальное адресное пространство):
  - команды и данные (глобальные),
  - стек: содержит экземпляр локальных данных каждой вызванной функции,
  - выделенные блоки памяти.
- В ОС:
  - информация о процессе (идентификатор, владелец, открытые файлы, текущий каталог, переменные окружения, родительский процесс — "дерево" процессов — и др).
- В процессоре:
  - регистры (арифметические, состояния, счетчик команд, указатель стека и др).
- Один процессор может исполнять много процессов "одновременно" (псевдопараллелизм): необходимо переключение между задачами через некоторый интервал времени
  - необходимо сохранить и восстановить значения регистров.

Создание процесса

- Инициализация системы.
- Выполнение процессом системного вызова:
  - "по собственной инициативе",
  - по запросу пользователя.

Завершение процесса

- Добровольное:
  - обычный выход;
  - из-за нефатальной ошибки.
- Принудительное;
  - из-за фатальной ошибки (инициатива ОС);
  - уничтожение другим процессом или ОС (оболочкой)
    - "по собственной инициативе",
    - по запросу пользователя.

Создание процессов: ОС *nix (POSIX)

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>

int  main() {
    pid_t pid;
    char *const pars[] = {"/usr/bin/ls", "-l", "/path/to/dir", NULL};
    pid = fork();
    if (pid == -1) { /* Дочерний процесс не создан */
        perror("fork error");
        exit(2);
    }
    else if (pid == 0) {   /* Дочерний процесс */
        execv("/bin/ls", parmList);
        perror("exec error");
        exit(1);
    } else {   /* Родительский процесс */
        int wstatus;
        pid_t p = wait(&wstatus);
        printf("process: %d status: %d\n", p, wstatus);
        exit(0);
    }
}
```

Создание процессов: ОС Windows

```
#include <windows.h>
#include <stdio.h>

int main () {
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );
    if (!CreateProcess(TEXT("C:\\Windows\\notepad.exe"),
                NULL,NULL,NULL,FALSE,
                CREATE_NEW_CONSOLE,
                NULL,NULL,
                &si,
                &pi
       )) { /* Процесс не создан */
        fprintf(stderr, "exec error");
        exit(1);
    } else { /* Ожидаем процесс */
        WaitForSingleObject(pi.hProcess, INFINITE);
        CloseHandle(pi.hProcess);
        CloseHandle(pi.hThread);
        exit(0);
    }
}
```

Переменные среды

- Представляют собой пары строк (имя, значение). Передаются дочернему процессу как копия всех пар родительского процесса (изменение в дочернем процессе не влияет на родительский).
- Могут задаваться, изменяться и читаться системными вызовами, поддерживаются в командной строке.
- *nix (POSIX)
  - `var=value` — присвоение значения
    `echo $var` — прочтение значения
    `export var` — передача дочернему процессу
    `a_cmd_to_run` — запуск (дочернего) процесса
    `printenv` — вывод значений всех переменных
    `var=value a_cmd` — запуск процесса с заданной переменной
  - некоторые системные переменные:
    - PATH, LANG, HOSTNAME, HOME, http_proxy,...
- DOS/Windows

  `SET var=value` — присвоение значения
  `echo %var%` — прочтение значения
  `a_cmd_to_run` — запуск процесса
  `set` — вывод значений всех переменных
  - некоторые системные переменные:
    - PATH, COMSPEC, TEMP, HOMEPATH, APPDATA,...

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- ➔ Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Планировщик процессов

- Состояния процессов:
  - **выполняемый** (используется ЦПУ),
  - **готовый** (работающий, но ожидающий ЦПУ),
  - **заблокированный** (ожидающий внешнего события),
  - остановлен, создание, завершение, зомби.
- Планировщик процессов (компонент ОС):
  - **таблица процессов**
    - управление процессом: идентификатор, параметры планирования, приоритет, родительский процесс, состояние, регистры, сигналы, использованное процессорное время, время запуска;
    - управление памятью: указатели на сегменты кода, данных, стека;
    - управление файлами: корневой каталог, рабочий каталог, дескриптор файлов и стандартных потоков (ввода, вывода, ошибки), идентификатор пользователя, идентификатор группы;
  - состояние процессов и приоритет.

Алгоритмы планирования

- По наличию прерываний процессов:
  - **Непрерывающий** (пакет выполняется пока не будет заблокирован или завершен);
  - **Прерывающий** (происходят периодические переключения между процессами по таймеру).
- По типу среды выполнения:
  - пакетная (компьютеры для больших вычислений);
  - интерактивная (пользовательские компьютеры);
  - реального времени.
- Общее для всех сред задачи алгоритма планирования:
  - равнодоступность (предоставление каждому процессу справедливой доли времени центрального процессора);
  - принуждение к определенной политике;
  - баланс (поддержка загруженности всех составных частей системы).

Задачи алгоритма планирования

- Пакетные системы:
  - производительность (выполнение максимального количества заданий в час),
  - оборотное время (минимизация времени между представлением задачи и ее завершением),
  - поддержка постоянной загруженности процессора.
- Интерактивные системы:
  - время отклика (быстрый ответ на запросы),
  - пропорциональность (оправдание пользовательских надежд).
- Системы реального времени:
  - соблюдение предельных сроков,
  - предсказуемость.

Планирование: пакетные системы

- **Неприоритетная очередь** (первый пришел — первым обслужен): процесс выполняется до блокировки или завершения.
  - Преимущества: простота реализации, минимальные затраты ресурсов на планирование;
  - Недостатки: неоптимальность. Пример:
    - 1 процесс 1000 сек. вычислений + 1 сек. ввод-вывод
    - 2-101 процессы 1 сек. вычислений + 10 сек. ввод-вывод

    если начать с 1-го: 2001 сек.

    ЦПУ: 1000 сек. (проц. 1) -> 1*100 сек. (проц. 2-101) -> 901 сек (простой)
    УВВ: 1000 сек. (простой) -> 1 (проц. 1) -> 10*100 (проц. 2-101)

    если начать со 2-го процесса: 1101 сек. (45% экономии!)

    ЦПУ: 1*100 сек. (проц. 2-101) -> 1000 сек. (проц. 1) -> 1 сек. (простой)
    УВВ: 1 сек. (простой) -> 10*100 (проц. 2-101) -> 99 сек. (простой) -> 1 сек. (проц. 1)
- **Сначала самое короткое задание**.
  - Преимущества: оптимальность (можно доказать).
  - Недостатки: необходимо предварительное знание времени выполнения, оптимален только при одновременном запуске заданий.
- **Приоритет заданию с наименьшим оставшимся временем выполнения**
  - Преимущества: оптимальность.
  - Недостатки: необходимо предварительное знание времени выполнения.

Планирование: интерактивные системы

- **Циклическое планирование**: каждому процессу назначается квант времени.
  - Переключение процессов не должно происходить ни часто, ни редко (малый квант – большие затраты на переключения, большой квант – большое время отклика, оптимально – 20-50 мс).
- **Приоритетное планирование**: запускается тот, у кого выше приоритет, после исчерпания кванта времени приоритет понижается.
- **Гарантированное планирование**: отведение каждому процессу "положенной" доли процессорного времени.
  - Если процесс превысит соотношение доли своего конкурента, происходит переключение.
- **Лотерейное планирование**: выделение процессам нужного количества "лотерейных билетов", пропорционально приоритету.
  - Каждый квант времени происходит розыгрыш следующего кванта с равной вероятностью выигрыша для каждого билета.
- **Справедливое планирование**: учет количества процессов, запущенных одним пользователем.
- **Выбор следующим самого короткого процесса.**
- *Использование нескольких очередей.*

Планирование: системы реального времени

- Если происходит m периодических событий, событие i возникает с периодом, τ_i и для обработки каждого события требуется t_i секунд процессорного времени, то поступающая информация может быть обработана только в том случае, если

$$S = \sum_{i=1}^{m} \frac{t_i}{\tau_i} \leqslant 1$$

- Система реального времени, отвечающая этому критерию, называется планируемой.

Алгоритм RMS

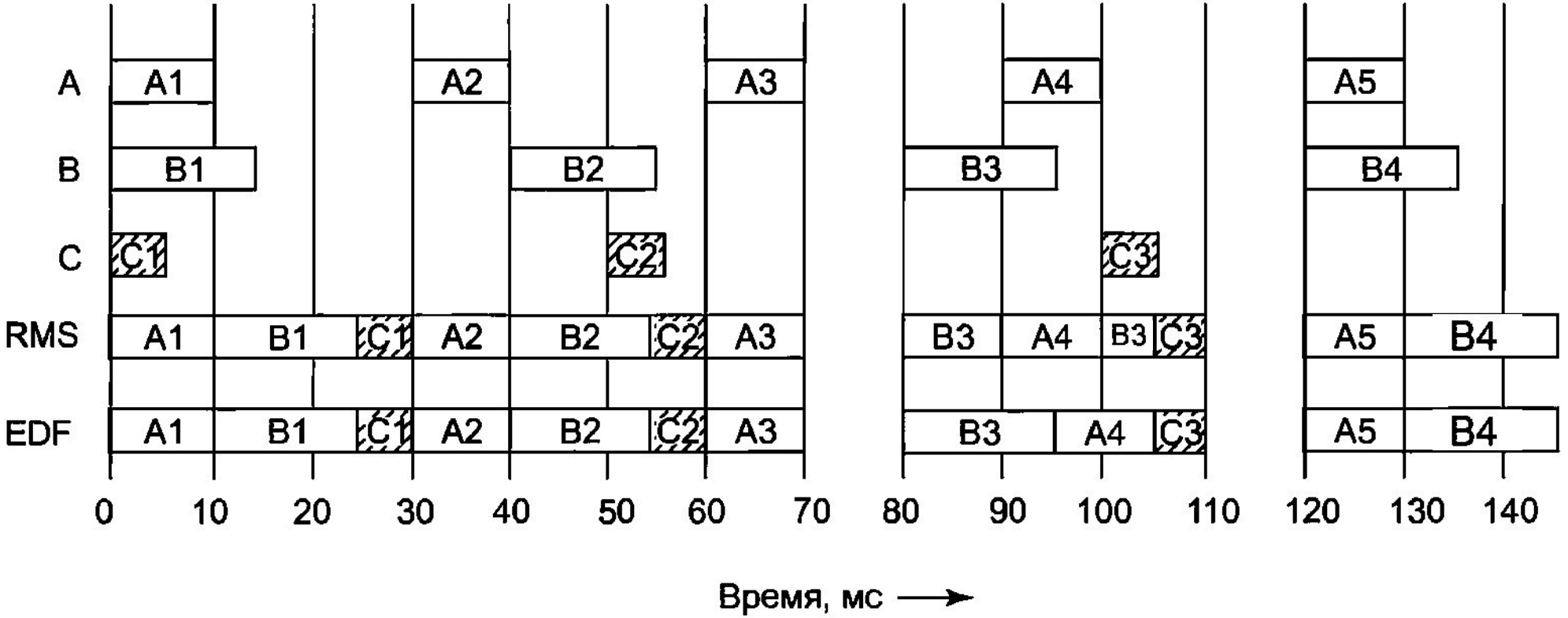
- Rate Monotonic Scheduling — планирование с приоритетом, пропорциональным частоте запуска процесса: статический алгоритм.
  - Каждый периодический процесс должен быть завершен в течение своего периода.
  - Ни один из процессов не зависит от других процессов.
  - Каждому процессу для каждого интервала его работы требуется одинаковое количество процессорного времени.
  - У непериодических процессов нет крайних сроков.
  - Вытеснение процесса происходит мгновенно без каких-либо издержек.
- Запускает готовый к работе процесс с наивысшим приоритетом, вытесняя, при необходимости, выполняемый процесс.

Алгоритм EDF

- Earliest Deadline First — алгоритм с приоритетом для процесса с ближайшим истекающим крайним сроком: динамический алгоритм.
  - Не требует, чтобы процесс был периодическим.
  - Не требует постоянства интервалов времени использования центрального процессора.
- Планировщик ведет список готовых к работе процессов, отсортированных по их крайним срокам, запускается процесс, который идет первым по списку.
- Если появился новый готовый процесс, у которого крайний срок раньше, текущий процесс вытесняется новым процессом.

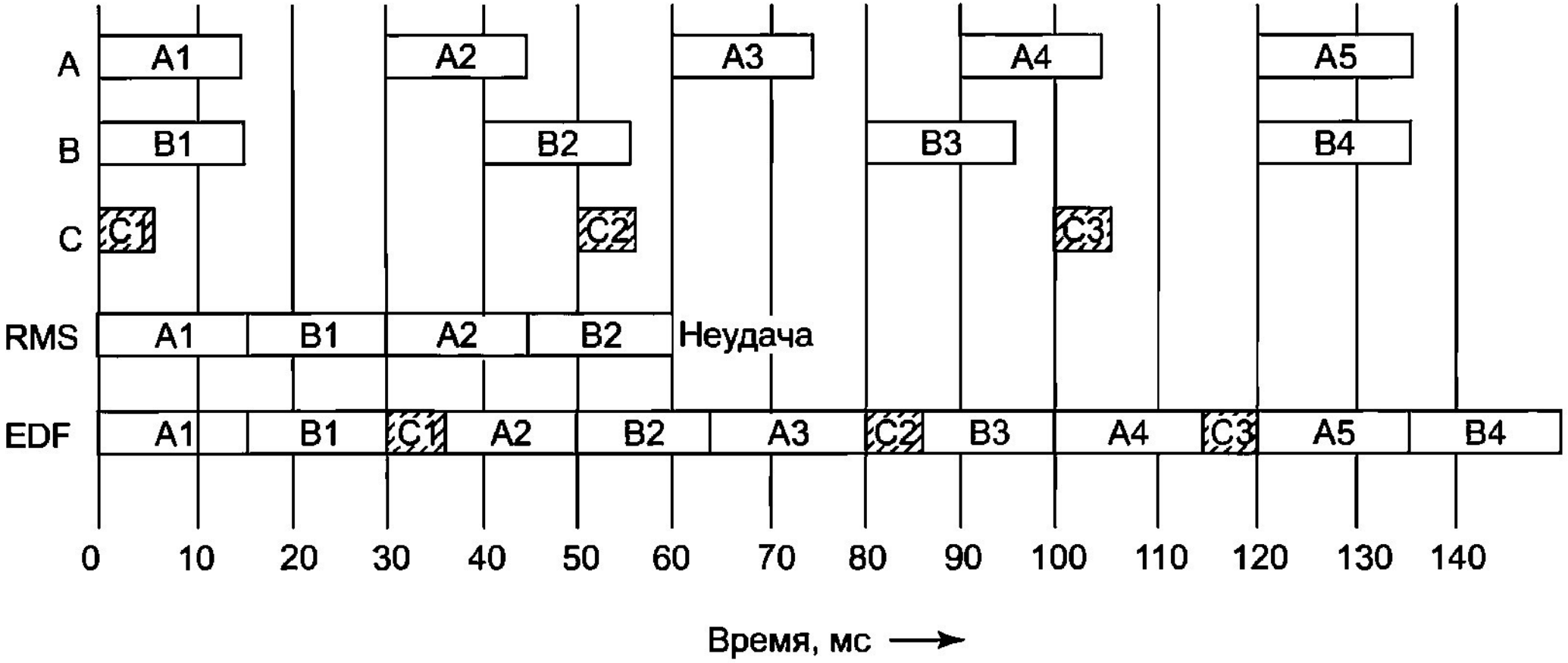
Пример работы алгоритмов

- Процесс A: 33 Гц (τ=30 мс, t=10 мс)
  Процесс B: 25 Гц (τ=40 мс, t=15 мс)
  Процесс C: 20 Гц (τ=50 мс, t=5 мс)
  S=0.80833<1



Пример работы алгоритмов

- Процесс A: 33 Гц (τ=30 мс, t=15 мс)
  Процесс B: 25 Гц (τ=40 мс, t=15 мс)
  Процесс C: 20 Гц (τ=50 мс, t=5 мс)
  S=0.975<1



Планирование: многопроцессорные и распределенные системы

- Многопроцессорные централизованные системы: назначение процессов процессорам:
  - общая очередь (или несколько);
  - разделение нагрузки.
- Распределенные системы:
  - **модель рабочих станций** (станции разных категорий, соединенные локальной сетью);
  - **модель процессорного пула** (массив процессоров и терминалов).

Рабочие станции (1)

- **Координатор**:
  - простаивающей рабочая станция должна сообщить координатору о простое;
  - процесс, желающий воспользоваться станцией, сообщает об этом координатору;
  - координатор сообщает очередному процессу об освободившейся рабочей станции.
- **Диспетчер рабочей станции**:
  - процесс обращается к диспетчеру рабочей станции с просьбой о выделении ресурсов;
  - диспетчер сообщает координатору о занятости и освобождении станции.
- **Процесс**:
  - запускает, выполняет и завершает работу на удаленной рабочей станции.

Рабочие станции (2)

- Распределение процесса по процессорам (станциям): минимизация времени простоя процессоров, минимизация времени работы процессов:
  - немигрирующая стратегия,
  - мигрирующая стратегия.
- Алгоритмы:
  - детерминированные или **эвристические**;
  - централизованные или **распределенные**;
  - оптимальные или **приемлемые**;
  - инициированные простаивающей или перегруженной станцией.

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- ➔ Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Межпроцессное взаимодействие (InterProcess Communication, коммуникация процессов).

- Цель 1 — передача информации от одного процесса другому:
  - обмен информацией, необходимой для решения поставленной процессам совместной задачи;
  - обмен информацией, необходимой для достижения двух других целей.

  Проблема: раздельное адресное пространство. Решения — разделяемая (общая) память, передача сообщений, передача потоков данных.
- Цель 2 — обеспечение совместной работы без взаимных помех и блокировок:
  - например, попытка захватить последнее место, попытка одновременного доступа к общим ресурсам и т.п.
- Цель 3 — выработка последовательности взаимозависимых действий:
  - например, начало работы процесса B над данными только после того, как процесс A выработает эти данные.

Механизмы межпроцессного взаимодействия

Способ	Описание	Поддержка
Файл	Использование обычного файла на носителе информации.	Большинство ОС
Сокет	Данные передаваемые от одного процесса другому через сетевой интерфейс (клиент-сервер).	Большинство ОС
Канал	Файл особого рода, позволяющий проводить запись с одного конца (один процесс) и чтение с другого (другой процесс).	POSIX, Windows
Общая память	Получение несколькими процессами доступа к одному блоку памяти.	POSIX, Windows
Сигнал	Системное сообщение, передаваемое от одного процесса другому для выполнения команды (а не передачи данных).	Большинство ОС
Очередь сообщений	Реализуется в ОС, позволяет процессам записывать и прочитывать сообщения без прямого взаимодействия.	Большинство ОС
Передача сообщений	Использование очереди сообщений или не-ОС средств коммуникации.	RPC, на уровне библиотек (MPI, Java)
Отображение файла в память	Прямое изменение по адресу в памяти.	POSIX, Windows

Сигналы

- Программное средство, с помощью которого может быть прервано функционирование процесса в ОС *nix. Позволяет процессам реагировать на события:
  - произошедшие внутри себя;
  - произошедшие в ОС или аппаратной среде;
  - произошедшие в другом процессе.
- Каждый сигнал характеризуется уникальным номером, определяющим тип сигнала (в Си сопоставляется символическая константа):
  - внутри процесса регистрируются функции-обработчики сигналов по номеру.
- Возможные реакция процесса на сигнал:
  - вызов обработчика сигнала;
  - завершение процесса;
  - игнорирование сигнала;
  - приостановка исполнения.

Основные виды сигналов

Сигналы разделяют по 5 типам. Примеры.

- Управление: SIGKILL (9, неперехватываемый), SIGSTOP (23, неперехватываемый), SIGCONT (25), SIGTERM (15), др.
- Уведомление: SIGALRM (28), SIGCLHD (18).
- Исключение: SIGILL (4), SIGSYS (12), SIGFPE (8).
- Пользовательский: SIGUSR1 (16), SIGUSR2 (17).
- Отладка: SIGPROF (29).

Посылка сигналов из оболочки

kill [-signal] pid

```
$ kill 1234          $ kill -STOP 1234
$ kill -9 1234       $ kill -CONT 1234
```

Обработка сигналов (1)

```
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>

void sig_handler(int i)
{
    switch (i)
    {
        case SIGINT:
            printf("Ctrl-C pressed, ignoring\n");
            break;
        case SIGQUIT:
            printf("Quit signal recived, exitting\n");
            exit(0);
    }
}
```

Обработка сигналов (2)

```
void sig_kill_handler(int i)
{
    printf("Kill signal recived, exitting\n");
    exit(1);
}

int main()
{
    if (signal(SIGINT, sig_handler) == SIG_ERR)
        fprintf(stderr, "Can't catch SIGINT\n");
    if (signal(SIGQUIT, sig_handler) == SIG_ERR)
        fprintf(stderr, "Can't catch SIGQUIT\n");
    if (signal(SIGKILL, sig_kill_handler) == SIG_ERR)
        fprintf(stderr, "Can't catch SIGKILL\n");

    printf("Ctrl-c disabled, use ctrl-\\ to quit\n");
    while (1) sleep(1);
}
```

Каналы (pipe)

- Анонимные (стандартный ввод, стандартный вывод, стандартное сообщение об ошибках):
  - возможность перенаправлений (в файлы)

```
$ sort <unsorted.txt >sorted.txt 2>errs.txt
$ ./too_talkative_prg >/dev/null
```

  - организации конвейеров:

```
$ cat file_in_cp1251.txt | iconv -f cp1251 |
  sort -u > out_file_sorted_in_utf8.txt
```

  - обмен информации с дочерними процессами.
- Именованные: особые файлы файловой системы, создаются процессами (и открываются) самими процессами.
  - обмен информацией между независимо запущенными процессами.

Использование каналов (1)

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
int main(int argc, char *argv[])
{
    int pipefd[2];
    pid_t cpid;
    char buf;

    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    if (pipe(pipefd) == -1) / * Создание канала */
    {
        perror("pipe");
        exit(EXIT_FAILURE);
    }
    cpid = fork();
```

Использование каналов (2)

```
    if (cpid == -1)
    {
        perror("fork");
        exit(EXIT_FAILURE);
    }
    if (cpid == 0) /* Дочерний процесс считывает */

    {
        close(pipefd[1]);  /* неиспользуемый конец */
        while (read(pipefd[0], &buf, 1) > 0)
            write(STDOUT_FILENO, &buf, 1);
        write(STDOUT_FILENO, "\n", 1);
        close(pipefd[0]);
        _exit(EXIT_SUCCESS);
    } else {  /* Родительский процесс записывает */
        close(pipefd[0]);  /* неиспользуемый конец */
        write(pipefd[1], argv[1], strlen(argv[1]));
        close(pipefd[1]);  /* послать EOF */
        wait(NULL);
        exit(EXIT_SUCCESS);
    }
} /* (C) Linux man pages */
```

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- ➔ Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Синхронизация процессов

- Параллельные (одновременно существующие процессы) могут быть
  - независимыми,
  - взаимодействующими (нуждающиеся в синхронизации).
- Синхронизация процессов требуется при
  - работе с общими ресурсами (**состязательная ситуация**),
  - использовании одним процессом, результатов работы другого процесса.
- Синхронизация — часть межпроцессного взаимодействия (требуется передача информации).

Состязательная ситуация: спулер принтера

- Процесс 0 — диспетчер печати, имеет каталог (очередь) печати, а также счетчики out (номер документа к печати) и in (номер свободного места).
- Процессы 1 и 2 — пытаются распечатать файл.
  - Процесс 1 считывает значение in (in=15, out=4).
  - Процесс 2 считывает значение in (in=15, out=4).
  - Процесс 1 записывает файл к печати на место 15, увеличивает его на 1 (in=16, out=10).
  - Процесс 2 записывает файл к печати на место 15, увеличивает его на 1 (in=16, out=10).
  - Диспетчер печати "потеряет" файл процесса 1.
- Произойдет или нет "нехорошая" ситуация зависит от многих факторов, разные запуски могут давать разный результат (часто удачный, но, чем больше параллелизма в целом, тем больше шансов, что процесс 1 никогда не сможет ничего напечатать).

Взаимное исключение

- Концептуальное решение: **критическая секция** — выделенный участок кода процессов, два процесса не могут находиться в одной критической секции одновременно. Как реализовать алгоритмически?
- Простейшее решение: запрет прерываний (переключений процессов). Проблемы:
  - в многопроцессорной системе прерывания запрещаются только на одном процессоре;
  - разрешать запрещение в пользовательским процессом чревато (длительные задержки, зависания системы).
- Нужно другое решение, удовлетворяющее ряду условий:
  - два процесса не могут одновременно находиться в своих критических областях;
  - реализация не может опираться на предположения о скорости и количестве процессоров (в т.ч. на время работы процессов);
  - процессы, находящиеся вне критической области, не должны блокироваться;
  - процессы не должны находиться в вечном ожидании входа в свои критические области.

Идея: переменная-замок

```
/* shared */ int lock = 0;

while (WORK)
{
    while(lock);
    lock = 1;
      enter_critical();
    lock = 0;
      non_critical();
}
```

- Не решает проблемы:

```
    while(lock); lock = 1;
```

не является атомарной процессорной операцией (возможно переключение процессов и одновременный вход двух процессов в критическую секцию).

Идея: строгое чередование

```
/* shared */ int turn = 0;

while (WORK)
{
    while (turn != i);
      enter_critical();
    turn = 1 - i;
      non_critical();
}
```

(для двух процессов, i=0,1).

- Не решает проблемы: когда один процесс работает существенно медленнее другого, поочередная организация вхождения в критическую область не подойдет.
  - Процесс 0 выходит из критической области, устанавливает значение переменной turn в 1.
  - Процесс 1 (разрешен вход в критическую область): быстро выходит из своей критической области. Оба процесса находятся вне своих критических областей, а переменная turn установлена в 0.
  - Процесс 0 быстро завершает свой полный цикл, выходит из критической области и устанавливает значение turn в 1. В этот момент значение turn равно 1 и оба процесса выполняются вне своих критических областей.
  - Процесс 0 завершает работу вне своей критической области и возвращается к началу цикла: ему не разрешено войти в его критическую область, поскольку переменная turn имеет значение 1 и процесс 1 занят работой вне своей критической области.

Решение: алгоритм Петерсона

```
/* shared */ int turn;              /* чья очередь? */
/* shared */ int interested[2]; /* исходные значения 0 */
void enter_critical(int process);
{
    int other = 1 - process;     /* номер другого
                                    процесса */
    interested[process] = 1;     /* демонстрация
                                    заинтересованности */
    turn = process;
    while (turn == process &&
           interested[other]);
}

void leave_critical(int process)
{
   interested[process] = 0;
}
```

- Недостаток: **активное ожидание** (трата процессорного времени, риск, что более приоритетный процесс вытеснит ожиданием менее приоритетный, находящийся внутри критической обрасти).

Аппаратное решение: команда TSL

```
enter_critical:
    TSL R, [LOCK]
    CMP R, 0
    JNE enter_critical
    RET
leave_critical:
    MOVE [LOCK], 0
    RET
```

- `TSL`: атомарная, аппаратная команда, блокирует шину от доступа других процессоров. Копирует переменную в регистр и сразу же устанавливает в 1.
- Недостаток: *активное ожидание*.

Аппаратное решение: команда XCHG

```
enter_critical:
    MOVE R, 1
    XCHG R, [LOCK]
    CMP  R, 0
    JNE enter_critical
    RET
```

- `leave_critical:`

```
    MOVE [LOCK], 0
    RET
```

- `XCHG`: атомарная аппаратная команда, меняет значение переменной и регистра.
- Недостаток: не подходит для многопроцессорных (многоядерных) систем (нет блокировки шины от команд других процессоров), *активное ожидание*.

Обход активного ожидания: приостановка и активация

- Блокирование работы, пока не будет разрешено войти в критическую область (вариант решения: добавить в систему вызовы "sleep" и "wakeup"?).
- Пример: задача ограниченного буфера.
  - Процесс-производитель помещает информацию в буфер, процесс-потребитель прочитывает. Что делать производителю, когда буфер полон? Блокироваться?
  - Пусть в буфере N мест, счетчик count показывает количество занятых мест, производитель блокируется, если буфер полон, потребитель разблокирует производителей, если в буфере появилось место.
  - Состязательная ситуация: доступ к count (не решает проблемы).

Производитель и потребитель

```
/* shared */ int count = 0;

void producer()
{
    int item;

    while (TRUE)
    {
        item = produce_item ();
        if (count == N) sleep();
        insert_item(item);
        ++count;
        if (count == 1) wakeup(consumer);   /* этот wakeup может "пропасть" */
    }
}
void consumer()
{
    int item;
    while (TRUE)
    {
        if (count == 0) sleep (); /* при count == 0 может произойти перекл. */
        item = remove_item ();    /* на потребителя, там count станет 1     */
        --count;                  /* но потребитель "заснет"                */
        if (count == N - 1) wakeup(producer);
        consume_item(item);
    }
}
```

Решение: семафоры (Дейкстра)

- **Семафор** — целочисленная переменная для подсчета количества активаций, отложенных на будущее.
- Две операции:
  - **down** уменьшает значение семафора и продолжает работу процесса, если оно не равно нулю, при нулевом значении процесс приостанавливается;
  - **up** увеличивает значение семафора на 1 и проверят, нет ли процессов, приостановленных в операции down, если есть — позволяет завершить их, в т.ч. уменьшить значение семафора на 1.
- Операции должны неделимые (без возможности прерывания процессов), доступ к семафорам осуществляется с помощью переменной-замка (команды TSL/XCGH) — *ожидание активное, но очень короткое*, в отличие от времени блокировки процесса в down.

Задача ограниченного буфера: семафор

- Решение задачи производителя-потребителя с помощью семафоров:
  - full подсчитывает количество заполненных мест в буфере,
  - empy — количество пустых мест
  - mutex предоставляет одновременный доступ (собственно организация критической секции).

```
/* shared */ semaphore mutex = 1,
                       empty = N,
                       full = 0;
```

```
void producer ()
{
    int item;
    while (TRUE)
    {
        item = produce_item ();
        down(&empty);
        down(&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}
```

```
void intconsumer ()
{
    int item;
    while (TRUE)
    {
        down(&full);
        down(&mutex);
        item = remove_item ();
        up(&mutex);
        up(&empty);
        consume_item(item);
    }
}
```

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- ➔ Взаимоблокировка: тупики и способы борьбы с ними.
- Потоки: создание, выполнение, планирование, взаимодействие.

Взаимоблокировка (1)

- Ресурсы
  - выгружаемые (могут быть "безболезненно" отобраны у процесса);
  - невыгружаемые (не могут быть отобраны у текущего владельца — иначе потеря данных, материальные убытки и др. проблемы).
- Использование ресурса:

(0)

```
semaphore resource_1;


void process_A(void)
{
    down(&resource_1);
    use_resource_1 ();
    up(&resource_1);
}
```

(1)

```
semaphore resource_1;
semaphore resource_2;

void process_A(void)
{
    down(&resource_1);
    down(&resource_2);
    use_both_resources ();
    up(&resource_2);
    up(&resource_1);
}
```

(2)

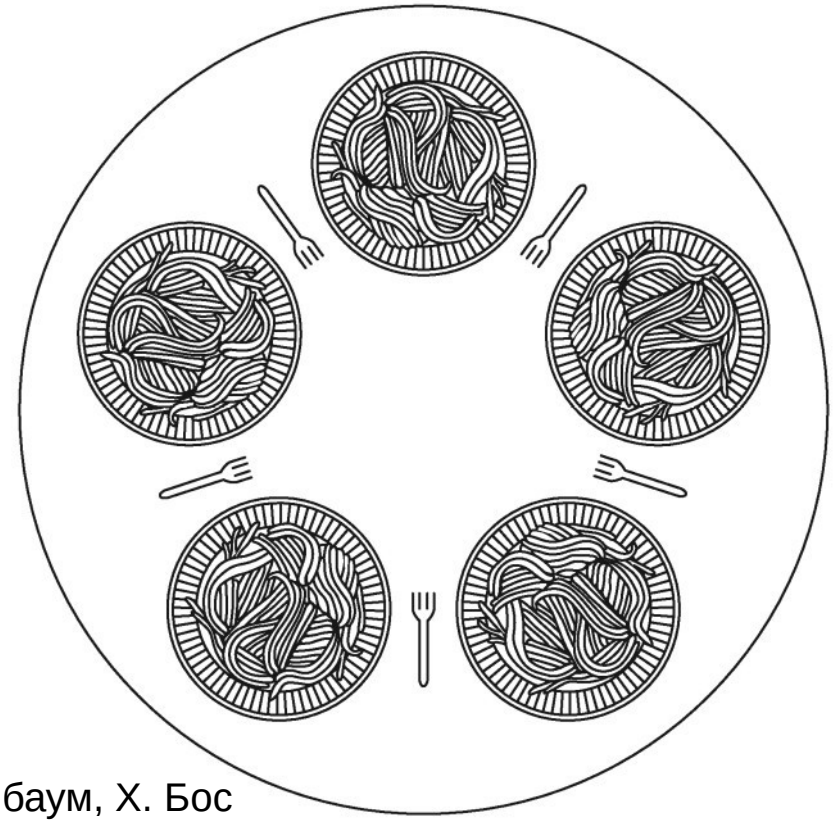
```
semaphore resource_1;
semaphore resource_2;

void process_A(void)
{
    down(&resource_2);
    down(&resource_1);
    use_both_resources ();
    up(&resource_1);
    up(&resource_2);
}
```

- Когда один ресурс (процессы типа 0) — проблемы нет. Процессы 1 и 2 могут попасть во взаимоблокировку (тупик): ресурс 1 занят процессом 1, ресурс 2 — процессом 2, они ждут друг друга.

Задача обедающих философов

- Формулировка.
  - Пять философов сидят за круглым столом, у каждого из них есть тарелка спагетти. Эти спагетти настолько скользкие, что есть их можно только двумя вилками. Между каждыми двумя тарелками лежит одна вилка. Жизнь философа состоит из чередующихся периодов приема пищи и размышлений.
  - Когда философ становится голоден, он старается поочередно в любом порядке завладеть правой и левой вилками. Если ему удастся взять две вилки, он на некоторое время приступает к еде, затем кладет обе вилки на стол и продолжает размышления. Задача написать программу для каждого философа, который действует предполагаемым образом и никогда при этом не попадает в состояние зависания?



(C) Э. Таненбаум, Х. Бос

- Наивная реализация:

```
void philosopher(int i)
{
    while (TRUE)
    {
        think ();
        take_fork(i);
        take_fork((i+1) % N);
        eat();
        put_fork(i);
        put_fork((i+1) % N);
    }
}
```

- Все философы могут одновременно взять левые вилки и никогда не получат правые: голоданием, или зависанием процесса ("взаимоблокировка").

ЗОФ: классическое решение

- ```
  semaphore mutex = 1;
  semaphore s[N];
  enum {THUNKING, HUNGRY, EATING} state[N];

  void philosopher(int i)
  {
      while (TRUE)
      {
          think();
          take_forks(i);
          eat();
          put_forks(i);
      }
  }

  void take_forks(int i)
  {
      down(&mutex);
      state[i] = HUNGRY;
      test(i);
      up(&mutex);
      down(&s[i]);
  }

  void test(i)
  {
      if (state[i] == HUNGRY &&
          state[LEFT] != EATING &&
          state[RIGHT] != EATING
         )
      {
          state[i] = EATING;
          up(&s[i]);
      }
  }

  void put_forks(i)
  {
      down(&mutex);
      state[i] = THINKING;
      test(LEFT);
      test(RIGHT);
      up(&mutex);
  }
  ```
  ```

Взаимоблокировка (2)

- Состояние системы, при которой образовалась группа процессов, каждый процесс которой не может продолжить работу из-за ожидания по крайней мере одного ресурса, занятого другими процессами той же группы.
- Необходимые условия возникновения взаимоблокировки
  - **Условие взаимного исключения**: процессы требуют монопольного владения ресурсами.
  - **Условие ожидания**: процессы удерживают уже выделенные им ресурсы, ожидая выделения дополнительных.
  - **Условие нераспределяемости**: ресурсы нельзя отобрать у удерживающих их процессов, пока они не будут использованы.
  - **Условие циклического ожидания**: существует кольцевая цепь процессов, в которой каждый процесс удерживает за собой один или более ресурсов, требующихся следующему процессу
- Борьба со взаимоблокировками:
  - **игнорирование** взаимоблокировок;
  - **выход** из взаимоблокировок;
  - **уклонение** от взаимоблокирововок;
  - **предотвращение (конструктивное недопущение)** взаимоблокировок.

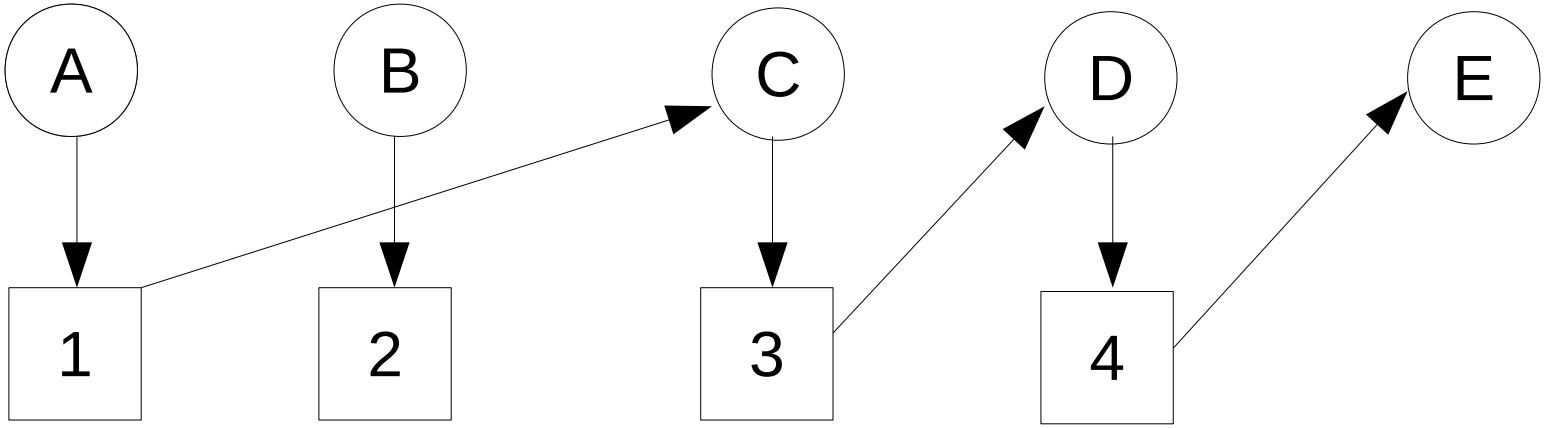
Игнорирование взаимоблокировок

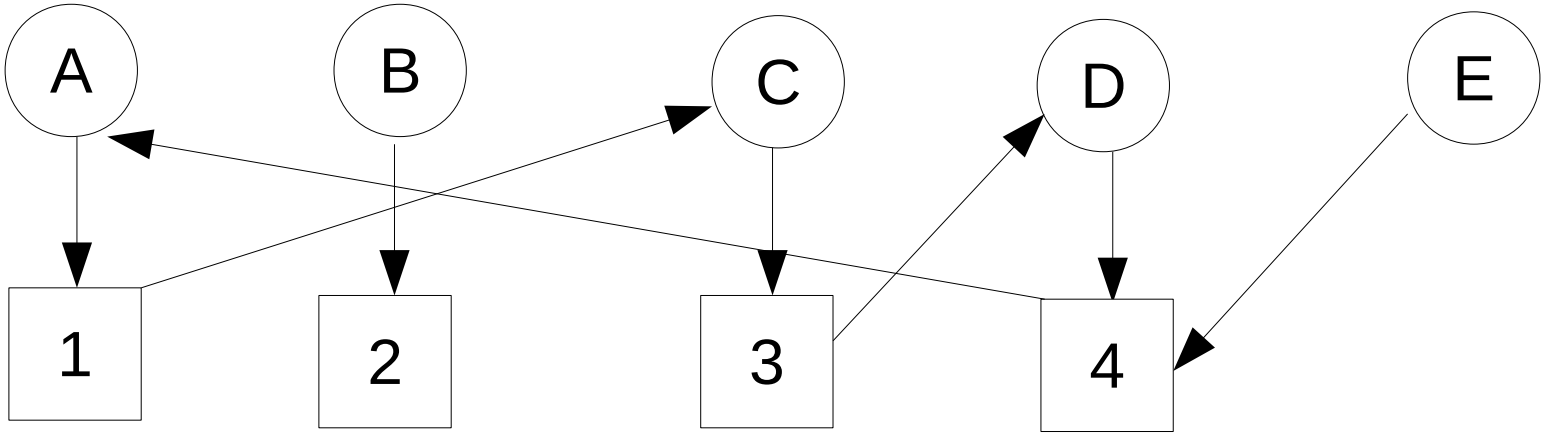
- "Алгоритм страуса": что дороже, ущерб от взаимоблокировок или борьба с ними?
  - взаимоблокировки очень редки;
  - большинство реальных систем обычно действует именно так;
  - Предотвращение зависания без выявления взаимоблокировки: таймаут ожидания ресурса и освобождение занятого ресурса.

Выход из взаимоблокировок

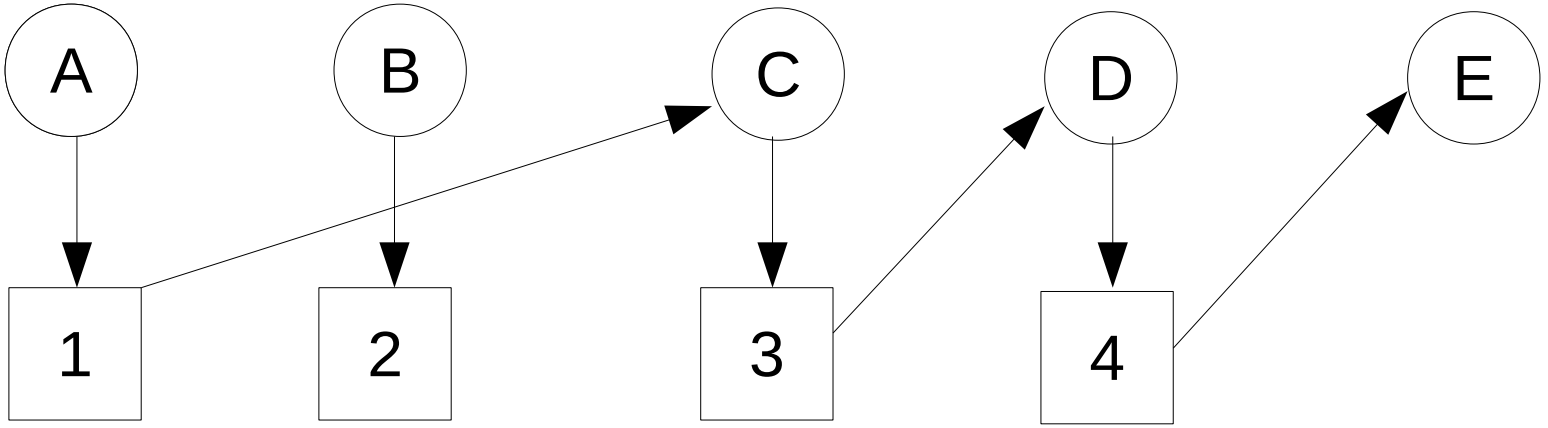
- Обнаружение взаимоблокировок: **граф процессов и ресурсов**
  - вершины двух типов (процесс и ресурс);
  - упорядоченные ребра
    - процесс → ресурс — запрос,
    - ресурс → процесс — удержание.
  - наличие цикла означает взаимоблокировку.
- Выход из взаимоблокировки:
  - временное изъятие ресурса;
  - откат процесса (запись состояния на диск и восстановление);
  - уничтожение процесса;
  - запрос оператора.

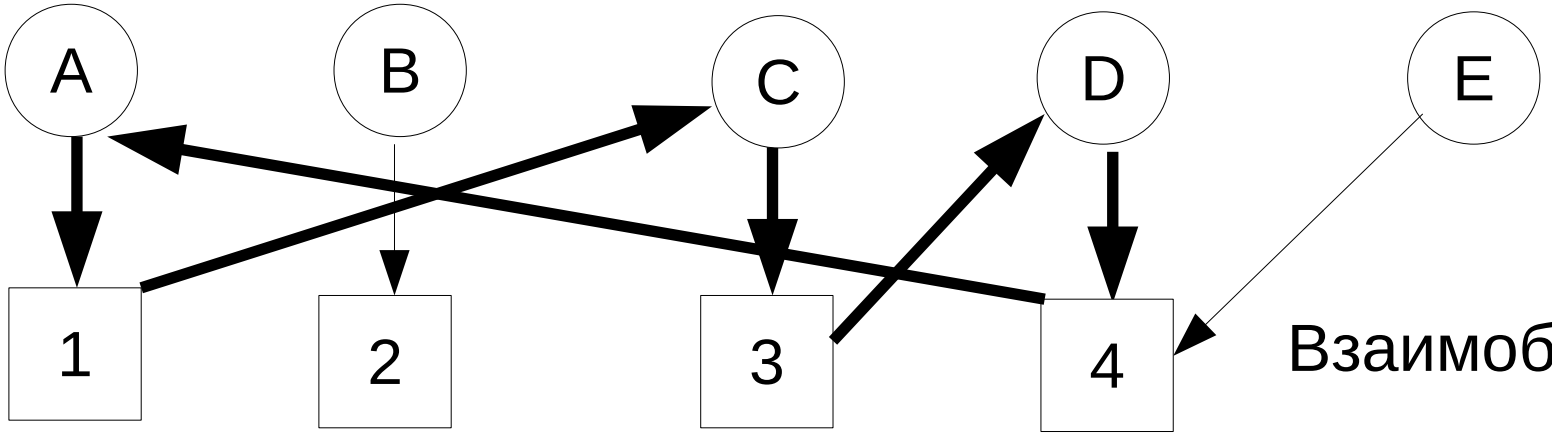
Граф процессов и ресурсов





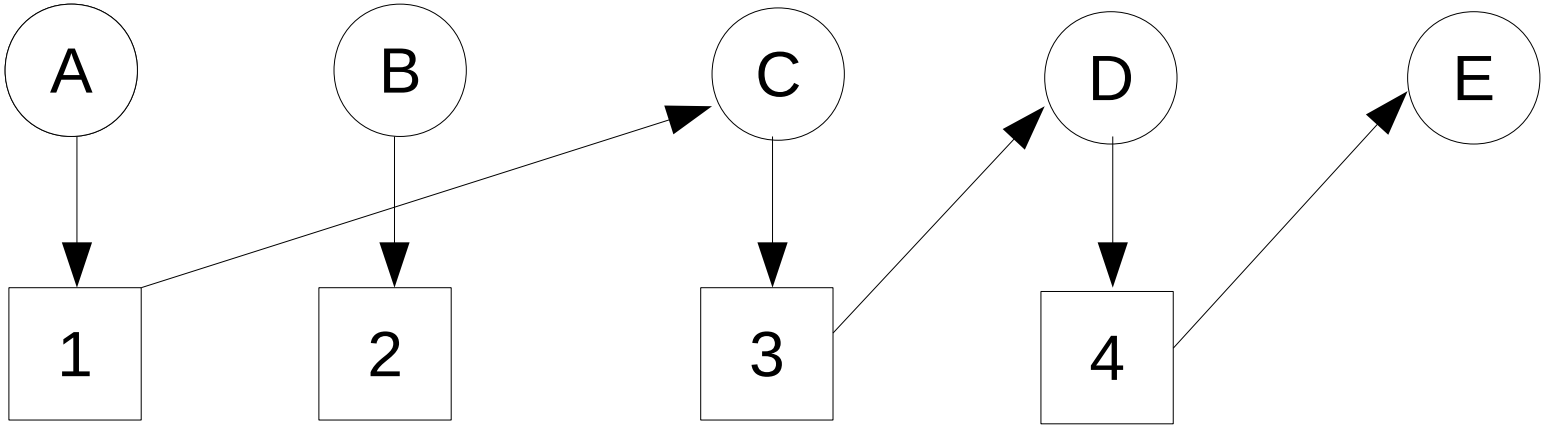
Граф процессов и ресурсов

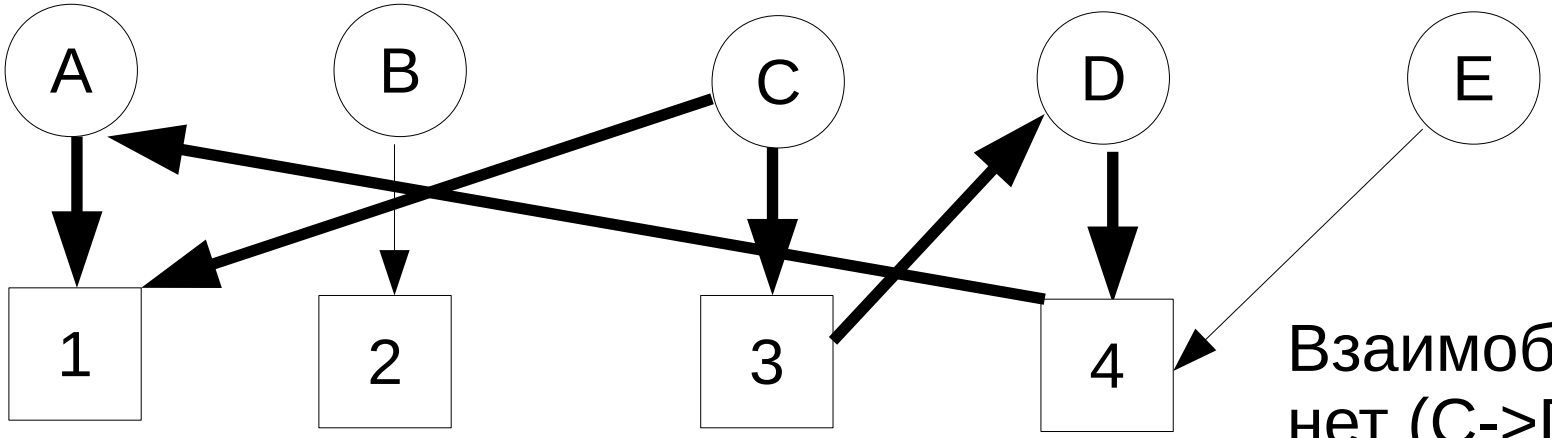




Взаимоблокировка!

Граф процессов и ресурсов





Взаимоблокировки нет (C->D->A)

Уклонение от взаимоблокировок

- Основано на понятии "безопасное" и "небезопасное" состояние системы. Задача недопустить перехода системы в небезопасное состояние.
- Алгоритм банкира (Дейстра, 1965)
  - Моделирует поведение банкира, выдающего ссуды: выдать ссуду можно только в том случае, если это безопасно.
  - Если запрос на ресурсы приведет к небезопасному состоянию, запрос отклоняется.
- Алгоритм не реализуем на практике:
  - требуется предзнание о будущих процессах;
  - требуется предзнание о будущих запросов ресурсов процессом.

Предотвращение взаимоблокировок

- Конструктивное недопущение возникновения одного из необходимых условий возникновения взаимоблокировки (атака на условие).

Атака на условия возникновения тупиков

- Условие взаимного исключения.
  - Невозможно для всех типов ресурсов.
  - Нужно минимизировать ресурсы, отдаваемые в монопольное использование (спулинг принтера вместо прямой печати на принтер).
- Условие взаимного ожидания.
  - Процессы должны сначала сообщать обо всех требуемых ресурсах, только потом система предоставит их.
  - Чаще всего заранее комплект ресурсов не известен.
  - Может быть реализовано на пакетных системах.
- Условие невыгружаемости.
  - Принудительный отбор ресурсов (иногда невозможен)
- Условие циклического ожидания.
  - Один процесс один ресурс: недостаточно.
  - **Иерархия ресурсов:** запрашивать ресурсы только по порядку (общая нумерация всех ресурсов) и запрет на получение ресурсов с меньшим номером, чем уже выделенные.

Процессы

- Процессоры: компоненты, архитектура, расширения архитектуры фон Неймана.
- Процессы: создание и выполнение.
- Планировщик процессов: алгоритмы планирования в различных системах.
- Межпроцессное взаимодействие: задачи и механизмы.
- Синхронизация процессов: состязательная ситуация и способы решения.
- Взаимоблокировка: тупики и способы борьбы с ними.
- ➔ Потоки: создание, выполнение, планирование, взаимодействие.

Потоки

- Потоки — части одного процесса, выполняются в одном адресном пространстве (каждый процесс может иметь один или более потоков).
  - Поток — последовательности команд.
  - Нужны не только для параллельных вычислений (например, поток управления графическим интерфейсом и рабочий поток).
- Элементы присущие процессу:
  - адресное пространство, данные (глобальные),
  - открытые файлы, дочерние процессы, сигналы и обработчики сигналов, учетная информация (в ОС).
- Элементы, присущие потоку:
  - счетчик команд, регистры, состояние (своя последовательность команд),
  - стек (свои локальные данные).

Реализация потоков

- В пользовательском пространстве (у каждого процесса своя таблица потоков и свой планировщик потоков) — реализовано с помощью библиотек.
  - Преимущества: быстрое переключение между потоками.
  - Недостатки: раздельное планирование потоков разных процессов, проблема реализации блокирующих системных вызовов (остановит все потоки? решение — использование библиотечных "заглушек" для системных вызовов, проверяющих, приведет ли данный вызов к блокировке и переключающий поток при необходимости).
- В ядре ОС (единая таблица потоков).
  - Решает проблему блокирующих вызовов, не решается проблема разветвления, обработки сигналов и др.
- Гибридная реализация (программист решает, какие потоки будут видны ядру, а какие окажутся в пользовательском пространстве).

Планирование и взаимодействие потоков

- Планирование потоков: аналогичные планированию процессов алгоритмы, но считается, что
  - процессы конкурируют (состязаются) между собой за ресурсы,
  - потоки одного процесса не конкурируют ("дружеское взаимодействие", кооперация).
- Взаимодействие потоков: не стоит проблема передачи информации (общее адресное пространство — взаимодействие потоков разных процессов попадает под взаимодействием процессов), остальные задачи решаются аналогично.

Потоки в pthreads

- Создание потока: pthread_create, pthread_exit.
  - `pthread_t` — дескрипор потока.
- Мьютексы — "упрощенные" семафоры, переменная, которая может быть или в заблокированном или разблокированном состоянии:
  - `pthread_mutex_init` (инициализация), `pthread_mutex_destroy` (уничтожение),
  - `pthread_mutex_lock` (овладение блокировкой или блокирование потока), `pthread_mutex_unlock` (разблокирование).
- Условные переменные — блокировка потока до выполнения определенного условия:
  - `pthread_cond_init`, `pthread_cond_destroy`,
  - `pthread_cond_wait` (блокировка в ожидании сигнала), `pthread_cond_signal` (сигнализация потоку об активации), `pthread_cond_broadcast` (сигнализация нескольким потокам об активации).

Производитель и потребитель

```
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutex;     pthread_cond_t condc, condp;      int buffer = 0;

void *producer (void *ptr)
{
    int i = 0;
    while (is_producing())
    {
        pthread_mutex_lock(&mutex);       /* исключительный доступ к буферу */
        while (buffer != 0) pthread_cond_wait(&condp, &mutex);
        buffer = ++i;
        pthread_cond_signal (&condc);     /* активизация потребителя        */
        pthread_mutex_unlock(&mutex);     /* освобождение доступа к буферу  */
    }
    pthread exit(0);
}

void *consumer (void *ptr)
{
    int i;
    for (i = 1; i <= max_producing(); i++)
    {
        pthread_mutex_lock(&mutex);
        while (buffer == 0) pthread_cond_wait(&condc, &mutex);
        buffer = 0;
        pthread_cond_signal (&condp);  /* активизация производителя */
        pthread_mutex_unlock(&mutex);
    }
    pthread exit(0);
}
```

Производитель и потребитель

```
int main(int argc, char **argv)
{
    pthread_t pro, con;
    pthread_mutex_init(&mutex, 0);
    pthread_cond_init(&condc, 0);
    pthread_cond_init(&condp, 0);
    pthread_create(&con, 0, consumer, 0);
    pthread_create(&pro, 0, producer, 0);
    pthread_join(pro, 0);
    pthread_join(con, 0);
    pthread_cond_destroy(&condc);
    pthread_cond_destroy(&condp);
    pthread_mutex_destroy(&mutex);
}
```

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- ➔ Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Компьютерные сети

- Сокеты.
- Сетевое оборудование.
- Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Компьютерные сети

- Сокеты.
- Сетевое оборудование.
- Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Сокеты

- **Сокет** — механизм межпроцессного взаимодействия, программный интерфейс обмена данными между процессами, исполняемыми как на одной, так и на разных машинах, соединенных посредством *сетевого оборудования*.
  - В настоящее время поддерживаются практически всеми операционными системами.
- Сокет *(другое значение)* — аппаратный разъем (для центрального процессора).
- Модель "клиент-сервер":
  - **сервер** — процесс, выполняющий действия по запросу клиента (клиентов),
  - **клиент** — процесс, посылающий запросы серверу.
- Слово "сервер" также используется для обозначения
  - любого компьютера, на котором запущен сервер;
  - специализированного компьютера, предназначенного для запуска серверов — **серверного компьютера**.

Клиент и сервер

- Различают клиентский и серверный сокеты.
  - Сервер: открытие сокета, ожидание запросов.
  - Клиент: открытие сокета, отправка запроса.
  - Сервер: подготовка и отправка ответа на запрос.
  - Клиент: получение и обработка ответа на запрос.
- Сокет — только конечная точка сетевого API. Задачи сетевого ПО и оборудования:
  - Как передать информацию на расстояние?
  - Как найти нужный компьютер в (большой?) сети?
  - Как закодировать (представить) информацию?
  - Как обеспечить безопасность передачи информации?
  - ...

Клиент и сервер: примеры

Задача	Сервера	Клиенты
Веб-браузинг	HTTP-серверы (apache, nginx)	веб-браузеры
Передача файлов	FTP-серверы (vsftpd)	ftp, FileZilla, Midnight Commander, Far Manager, большинство веб-браузеров
	Samba-сервер, встроеный в Windows	Проводник Windows, smb-клиенты (Midnight Commander, gvfs-smb)
Удаленный доступ	SSH-сервер	ssh, putty
	rdp-сервер, встроенный в Windows	rdp-клиент, встроенный в Windows
Сетевая база данных	mariaDB	mariaDB, веб-сервера (плагины)
Электронная почта	Сервер почты	Почтовый клиент (Thunderbird, Claws Mail), веб-сервера (плагины)

Компьютерные сети

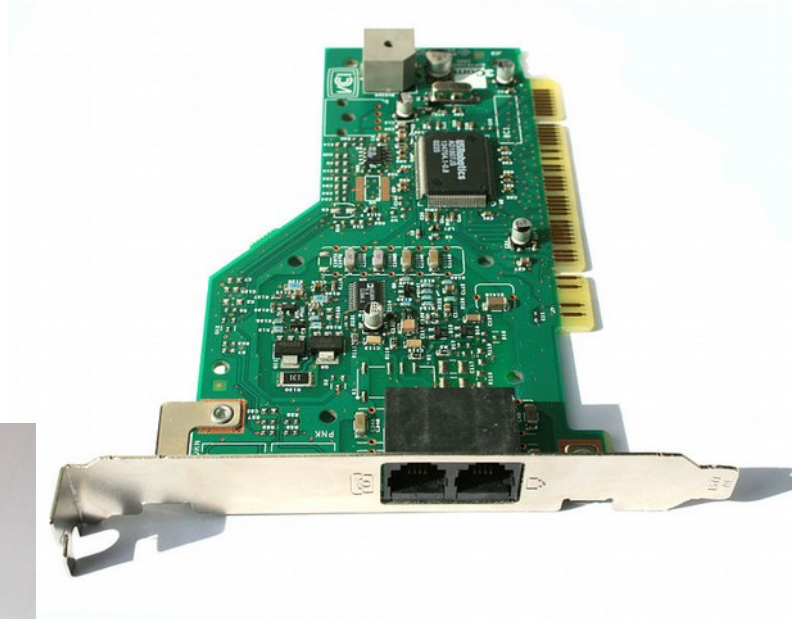
- Сокеты.
- ➔ Сетевое оборудование.
- Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

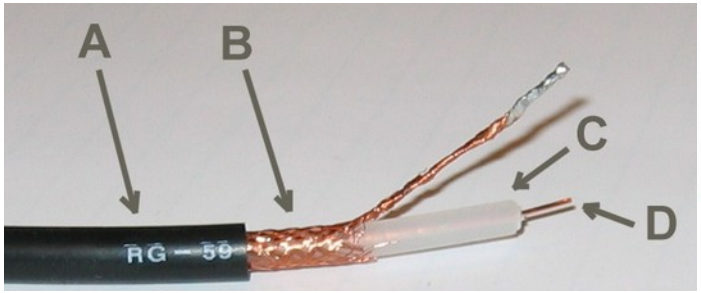
Сетевое оборудование

- "Частный случай" телекоммуникационного оборудования.
- Оборудование для подключения компьютера к сети:
  - сетевые карты — внешние или встроенные в системную плату;
  - модемы — телефонный, ADSL, сотовый, кабельный и др;
  - эмуляция сети с помощью иного оборудования (USB-кабель и др).
- Средства передачи информации на расстояние
  - проводные ("витая пара", оптико-волоконный кабель, коаксиальный кабель, телефонный кабель и др),
  - беспроводные (электро-магнитные волны и др.)
- Оборудование для соединения компьютеров в сеть и маршрутизации в сетях:
  - концентраторы, коммуникаторы,
  - маршрутизаторы, точки доступа,
  - усилители,
  - ...


MAC: 78DD08E88727
T77H126.00 LF AR5895
5200069811 - 04L1
145815712 8107043 R02
MADE IN CHINA


NETGEAR




A
B
C
D
RG — 59

Компьютерные сети

- Сокеты.
- Сетевое оборудование.
- ➔ Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Сетевые протоколы

- Протокол передачи данных (сетевой протокол) — набор правил интерфейса, который определяют поведение блоков данных при обмене данными между различными программами.
- Уровни протоколов (иерархия, абстракции):
  - декомпозиция сложной структуры на меньшие части и уровни,
  - обеспечение стандартного интерфейса между сетевыми функциями.
- Эталон: семиуровневая модель взаимосвязи открытых систем (OSI).
- Используется четырехуровневый стек протоколов TCP/IP (старше OSI).

Модель OSI

Класс уровня	Уровень	Тип данных	Функции	Примеры
Программный (HOST)	7. Прикладной (приложений)	Данные	Доступ пользователей к сетевым службам	HTTP, FTP, SSH
	6. Представительский		Представление и шифрование данных	ASCII, "Binary"
	5.Сеансовый		Управление сеансом связи	RPC, PAP
	4. Транспортный	Сегменты	Прямая связь между пунктами	TCP, UDP
Аппаратный (Media)	3. Сетевой	Пакеты	Маршрутизация и логическая адресация	IPv4, IPv6
	2. Канальный	Биты/кадры	Физическая адресация	PPP, IEEE 802.2, Ethernet, DSL
	1. Физический	Биты	Передача сигнала в физической среде	USB, витая пара, оптоволокно, радиоволны

Компьютерные сети

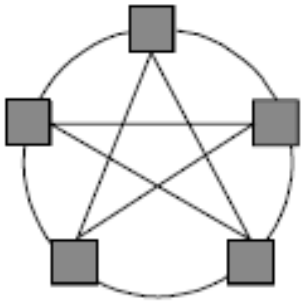
- Сокеты.
- Сетевое оборудование.
- Модель OSI: уровни протоколов.
- ➔ Протокол интернета: адресация и маршрутизация.
- Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Протокол интернета

- Протокол интернета (IP) — протокол децентрализованной адресации компьютеров и маршрутизации в компьютерной сети с динамически меняющейся топологией.
- IP-пакет — блок информации формата, определенного спецификацией протокола интернета.
  - Кроме собственно данных содержит данные об отправителе, получателе, контрольную сумму и др. информацию.
  - Более высокий уровень, чем прямая передача битов и байтов.
- Предназначен для доставки пакетов между узлами, не соединенными напрямую, через произвольное количество промежуточных узлов.
  - Военная цель: пакет должен дойти, даже если часть узлов выведена из строя.

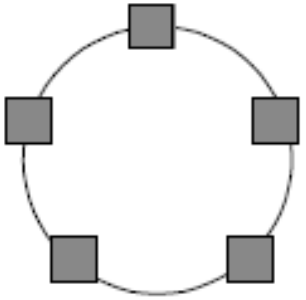
Топология сети (и параллельных вычислительных систем)

- Большее количество ребер связи увеличивает стоимость системы, но увеличивает производительность (уменьшает время передачи информации).
- "Закольцованность" сети (наличие циклов в графе) — потенциальная проблема (сеть может оказаться загружена на 100% блуждающими по кругу пакетами): у всякого пакета есть время жизни (TTL), уменьшающееся при проходе каждого узла.

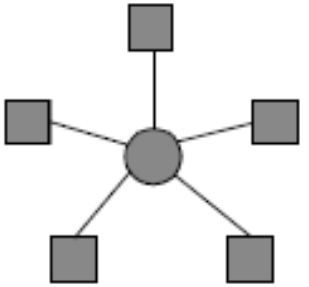


1) Полный граф

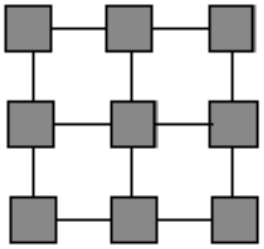
2) Линейка



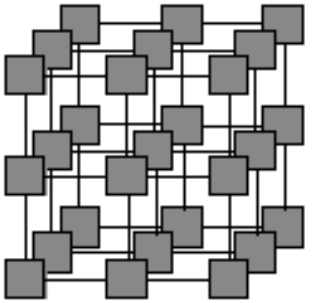
3) Кольцо



4) Звезда



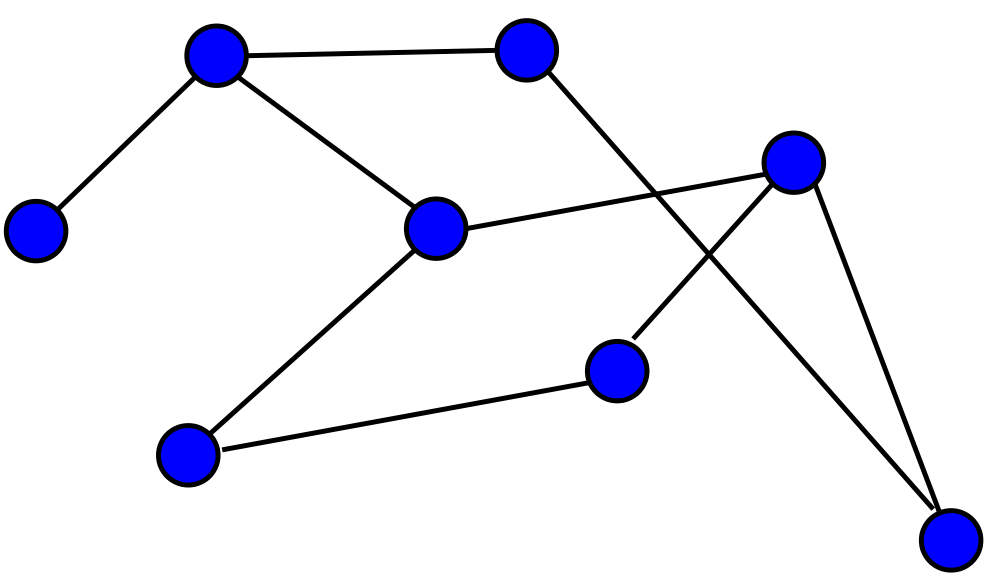
5) 2-мерная решетка

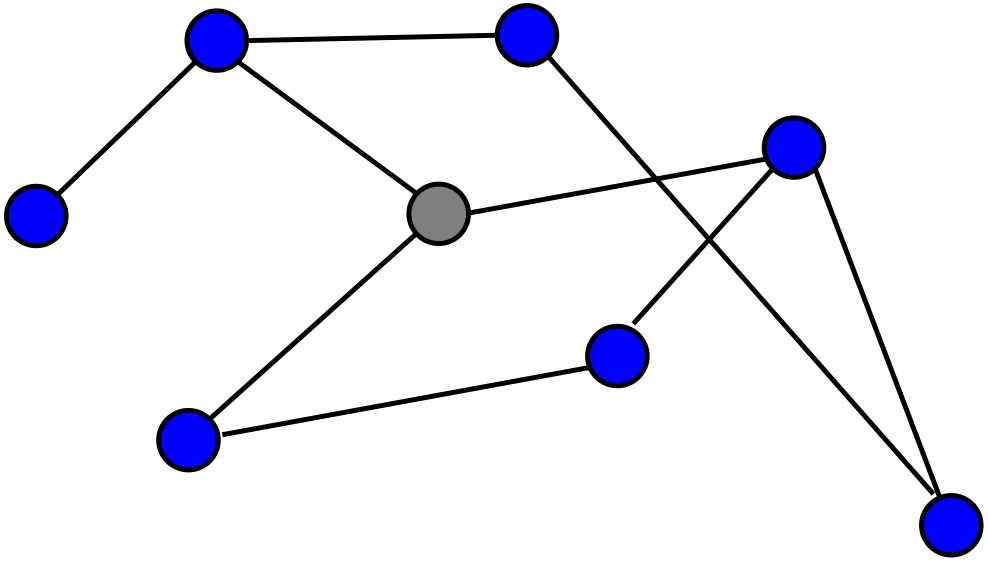


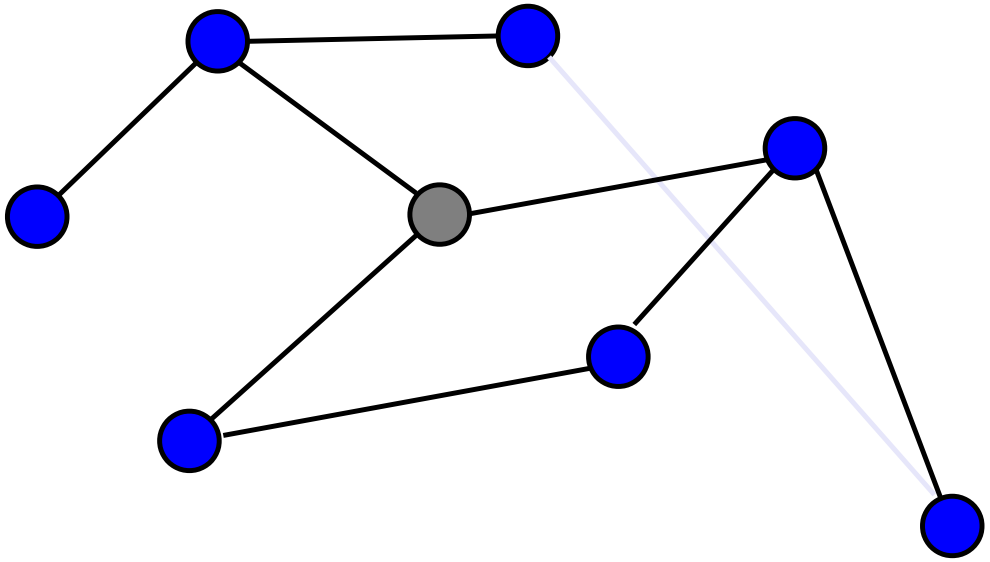
6) 3-мерная решетка

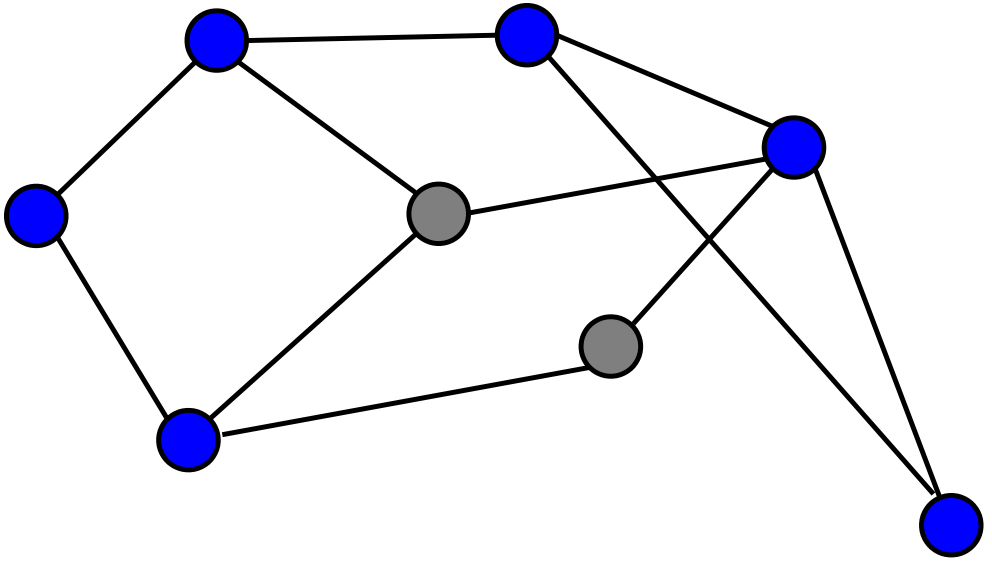
(C) В. Гергель

Топология сети интернет









Задачи: адресация, маршрутизация.

IP-адресация (1)

- Адресация IPv4: 32-х битовый адрес (макс. ~4 млрд. адресов — дефицит).
  - Адреса записываются как десятичные числа (4 октета), разделенные точками (195.19.251.2, 8.8.8.8);
  - Разделяется сетевая часть (логическая сеть, к которой относится адрес) — 1-3 октета — и машинная часть (конкретная машина в сети) — 3-1 октета.
  - Выделенные адреса:
    - 127.0.0.1 — локальный компьютер,
    - 10.x.x.x, 172.16.x.x, 192.168.x.x — локальные подсети,
    - 0.0.0.0 — источник пакетов локальной сети,
    - 169.254.x.x — канальные адреса (нет присвоения штатного адреса),
    - ...
- Адресация IPv6
  - 128-битный IP-адрес ("должно хватить всем"), записывается как 4 четырехзначных шестнадцатиричных числа, разделенных двоеточием, возможны сокращения нулей.
  - используется, но пока не вытеснил IPv4.

IP-адресация (2)

- Присвоение адреса компьютеру
  - вручную в настройках сети (полезно в небольших сетях и для основных узлов),
  - автоматически — выдается DHCP-сервером (Dynamic Host Configuration Protocol)
    - статически ("гарантированный" адрес),
    - динамически ("свободный" адрес).
- Доменное имя — для удобства доступа по символическому, а не цифровому адресу. IP-адрес по доменному имени выдает DNS-сервер (Domain Name System).
- В сети интернет
  - IP-адрес: IANA->RIR->ISP.
  - доменное имя: IANA->DNR->LAN.
- Идентификация компьютера — по IP-адресу, идентификация процесса (сервера) на компьютере — по номеру **порта (адрес:порт)**.
  - 195.19.251.2:80 (math.spbu.ru:80)
  - 192.168.0.1:3128 (localhost:80)
  - ...

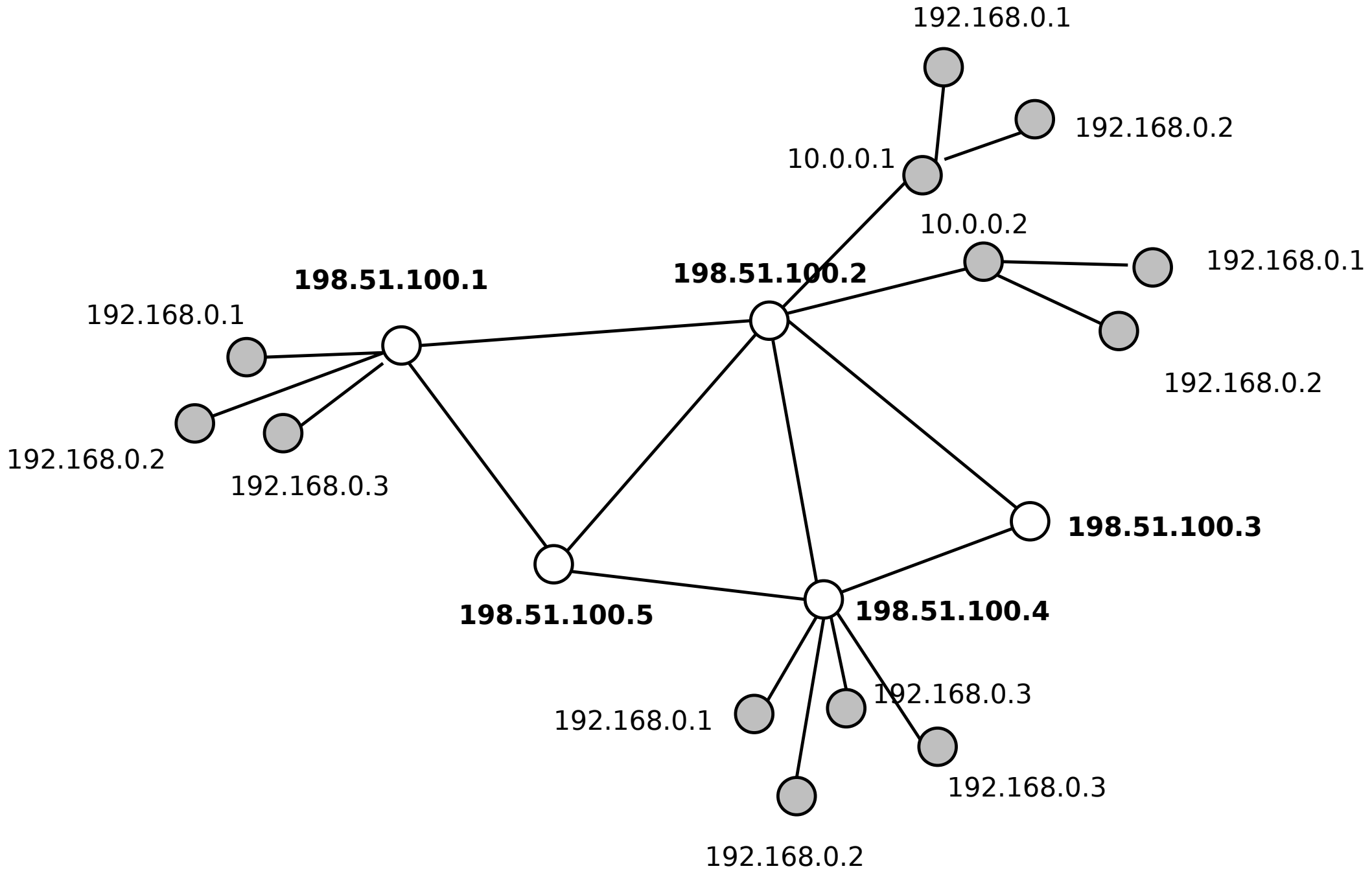
IP-маршрутизация

- Маршрутизация в сетях IP — алгоритм выбора **маршрута** следования информации от отправителя к получателям через коммуникационную сеть
  - Поиск маршрута в ориентированном графе, нагруженном по дугам, **маршрут** — это ориентированный ациклический граф с источником, соответствующим отправителю и стоками, соответствующими получателям (в простейшем случае, путь).
  - Оптимальный маршрут — маршрут, имеющий наименьшую **стоимость (число переходов)**, оптимальная маршрутизация – алгоритм выбора оптимального маршрута.
  - **Протокол маршрутизации** — протоколы обеспечения хотя бы работающей маршрутизации в сети с динамически изменяющейся топологией.
  - Данные маршрутизации сохраняются ОС в таблице, содержащей IP-адрес сети назначения, IP-адрес следующего узла (шлюз, маршрутизатор), стоимость пути до назначения. Алгоритм маршрутизации строит **таблицы маршрутизации**.

Маскарад (NAT)

- Преобразование сетевых адресов (NAT) — доступ с компьютеров внутренней подсети во внешнюю сеть или наоборот и др. задачи.
  - Экономия дефицитных IP-адресов.
  - Защита от обращения "снаружи" к внутренним серверам, сокрытие внутренних сервисов.
- Осуществляется с помощью таблиц маршрутизации:
  - при отправке пакета из локальной сети — замена локального IP-адреса источника на глобальный и замена стандартного порта источника на уникальный с сохранением в таблице;
  - при получении пакета из интернет — замена глобального IP-адреса на локальный по данным порта получателя (по таблице маршрутизации);
  - возможно статическое перенаправление портов.
- Может осуществляться
  - аппаратно ("маршрутизатор"),
  - программно (сервер маршрутизации, межсетевой экран).
- Недостатки:
  - "коллективная ответственность",
  - невидимость внутренних серверов извне.

Маскарад: пример



Компьютерные сети

- Сокеты.
- Сетевое оборудование.
- Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- ➔ Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Транспортные протоколы интернет

- Протокол **UDP** (User Datagram Protocol): передача пакетов между источником и получателем без предварительного установления связи (сообщения независимы друг от друга на уровне протокола):
  - для доставки сообщений используется протокол IP;
  - доставка, порядок и целостность сообщений (надежность) не гарантируется и должна обеспечиваться приложениями;
  - требует меньше накладных расходов, чем TCP;
  - используется некоторыми протоколами верхнего (прикладного) уровня (DNS, NTP).
- Протокол **TCP** (Transmission Control Protocol): надежная передача потока данных с предварительной установкой связи между источником информации и ее получателем
  - контролируется целостность, очередность и доставка пакетов,
  - используется многими протоколами верхнего уровня (TELNET, HTTP, FTP).

Стек TCP/IP

- Четырехуровневая модель: многообразие прикладных протоколов и типов соединения, минимум транспортных протоколов, один протокол интернета.
  - прикладной уровень (HTTP, FTP, DNS): 5-7 в OSI;
  - транспортный уровень (TCP, UDP): 4 в OSI;
  - уровень интернета (IP): 3 в OSI;
  - уровень соединения (Ethernet, IEEE 802.11, физические среды): 1-2 в OSI.
- Для обмена данными между узлами необходимо установить:
  - способ соединения (транспортный протокол);
  - адрес узла-адресата (IP-адрес);
  - номер порта процесса.
- Адресация в сетях TCP/IP:
  - верхний уровень (номера портов процесса),
  - IP-адрес (глобальный, уникальный и не зависящим от аппаратных средств),
  - Сетевые аппаратные MAC-адреса (Media Access Control): сетевым картам задаются уникальные 6-байтные аппаратные адреса), протоколы ARP (Address Resolution Protocol) и RARP (Reverse Address Resolution Protocol) преобразуют IP-адреса в аппаратные.

Компьютерные сети

- Сокеты.
- Сетевое оборудование.
- Модель OSI: уровни протоколов.
- Протокол интернета: адресация и маршрутизация.
- ➔ Транспортные протоколы: доставка пакетов.
- Порты и прикладные протоколы.

Порты и прикладные протоколы

- Порт: 16-битное число (0-65535).
  - Технически процесс может установить любой порт при открытии серверного сокета (**прослушивание порта**), одновременная работа двух серверов на одном порту на одном IP-адресе невозможна.
  - Некоторые порты (особенно с номерами до 1000) стандартизированы де-юре или де-факто (порт по умолчанию для сервера).
- Прикладной протокол — протокол уровня приложения.
- **URL** (Uniform Resource Locator) — универсальный локатор ресурсов в сети (символическая строка):
  - <схема>://[<логин>:<пароль>@]<хост>[:<порт>][/]<URL-путь>[?<параметры>][#<якорь>]
  - `https://user:123456@example.com/path/to/file?par1=v1&par2=v2#anchor`

Номера портов

Порт	Пользовательский протокол	Описание
7	ECHO	отправка сообщения обратно адресату
20	FTP	передача файлов по протоколу передачи файлов
21	FTP	управление протоколом передачи файлов
22	SSH/SFTP	удаленная оболочка/протокол передачи файлов через SSH
25	SMTP	отправка электронной почты
53	DNS	система доменных имен
67/68	DHCP	присвоение IP-адресов
80	HTTP	протокол передачи гипертекста
110	POP3	получение электронной почты
111	RPC	удаленный вызов процедур
123	NTP	протокол синхронизации времени по сети
143	IMAP	управление почтовым ящиком
443	HTTPS	защищенный протокол передачи гипертекста

Безопасность сетевого соединения

- Авторизация по паролю (логин для входа в удаленную систему).
  - Иные методы авторизации при использовании протоколов верхнего уровня.
- Файрвол (брандмаузер): блокировка нежелательных сетевых соединений — открытия сокетов по правилам (для пользователей, программ, процессов, портов, адресов, сетевых интерфейсов, входящий и исходящих).
- TCP/IP-соединение не зашифровано (данные могут быть перехвачены промежуточным узлом).
  - Безопасные прикладные протоколы (s): шифрование с открытым ключом (медленное, требуется сертификат) для передачи одноразового закрытого ключа (шифрование быстрее).
- Риск подмены хоста (узла, сервера) — служба сертификатов.
- Взломы системы
  - ОС,
  - сетевых протоколов,
  - серверов,
  - серверных приложений.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- ➔ Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- Управление памятью.
- Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- Управление памятью.
- Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

Память (1)

- Оперативная память и накопители информации:
  - с точки зрения архитектуры — постоянная память и оперативная память компьютера, плюс устройства ввода-вывода.
- С точки зрения решаемых задач: различные носители информации. Классификация.
  - По способу записи информации:
    - **постоянные** — ROM, компакт-диски (CD);
    - **однократно записываемые** — записываемые компакт-диски (CD-R);
    - **многократно перезаписываемые** — перезаписываемые компакт-диски (CD-RW);
    - **произвольно перезаписываемые** — магнитные диски (HDD), твердотельные накопители (SDD, флеш-карты), магнитная лента.
  - По зависимости от энергии:
    - **энергозависимые** — оперативная память (RAM), CMOS;
    - **энергонезависимые.**

Память (2)

- По способу доступа:
  - **произвольного доступа;**
  - **последовательного доступа** (магнитные ленты, отчасти лазерные диски).
- По скорости:
  - сверхбыстрая (**регистры**);
  - **кеш** (процессора, разных уровней);
  - **оперативная память**;
  - **долговременная** память
    - магнитные ленты, лазерные диски, флеш-карты, магнитные диски, твердотельные накопители,
    - могут иметь свой кеш (ОЗУ у HDD и SSD, SSD у HDD).

- Роль ОС:
  - выделение свободной оперативной памяти процессам;
  - выгрузка неиспользуемых данных процессов из оперативной памяти в долговременную при недостаче свободной оперативной памяти;
  - кеширование данных долговременной памяти в оперативной.

Устройства хранения информации

Память и хранение данных

- Память и устройства хранения информации.
- ➔ Адресация оперативной памяти.
- Управление памятью.
- Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

Оперативная память

- Оперативная память — дефицитный ресурс вычислительной системы.
  - "Программы стремятся занять всю доступную память".
  - На современных ПК недостаток объема ОЗУ ощутимее недостатка скорости ЦПУ.
  - Быстродействие памяти росло в 7 раз медленнее быстродействия ЦПУ.
- В многозадачных средах требуется управление памятью на уровне ОС ("диспетчер памяти"), существует аппаратная поддержка ("блок управления памятью", MMU).

Прямая адресация

- В программе указывается явный (физический) адрес.

  `MOV REG, [1000]`

  - ОС и ПЗУ размещаются в начале или в конце памяти.
- Проблемы прямой адресации:
  - невозможно запустить несколько процессов,
  - затруднены вариации конфигурации ОС.
- Проблемы доступа к физическим адресам:
  - риск вмешательства в код и данные ОС,
  - риск вмешательства в код и данные параллельных процессов (если бы была возможность их запускать).
- Использовалась на первых ПК (Spectrum), может использоваться во встроенных системах.

Сегментная адресация

- 16 битный адрес (указатель): прямая адресация 64Кб данных, режим **сегмент:смещение**.
  - 1 сегмент — 64К, смещение внутри сегмента — 16 бит, сегменты смещены друг относительно друга на 16 байт (физический адрес — 17 бит).
  - 1A21:13BA →0x1A21*0x10 + 0x13BA = 0x1B5CA, максимальный физический адрес FFFF:FFFF →~1088 Кб памяти.
  - IBM PC: 640 Кб памяти, выше — BIOS, видеопамять, доп. память (если установлена) для переноса ОС и резидентных программ.
- Код программы использует 16 разрядные адреса (смещения), сегмент определяется значением регистра.
  - CS (сегмент кода), DS (сегмент данных), SS (сегмент стека), ES (расширенный сегмент) — выделяются программе ОС при запуске или по запросу (ограничение на каждый сегмент 64К).
  - Программе неважно, в какой сегмент она загружена — возможно создание дочерних процессов.
  - Сохраняется прямой доступ ко всей памяти путем изменения регистров с вытекающими последствиями, многозадачность и безопасный режим не предусматривается.
  - Также: короткий (8 бит), ближний (16 бит, в пределах сегмента), дальний (32 бит, вся память) указатели.

Селекторная адресация: 32-бит

- 32-битный адрес, защищенный режим (начиная с 80386).
  - 32-битные регистры смещения: до 4 Гб памяти.
  - **Линейный адрес** (4 Гб) вычисляется как селектор+смещение (**виртуальный адрес**).
  - Программе доступно смещение, но селектор не подлежит изменению.
- **Физический адрес** не обязан совпадать с линейным — **страничная адресация**.
  - Страницы адресации формируются операционной системой, используются 2 таблицы страниц: глобальная (для ядра ОС) и локальная (своя для каждого процесса). Линейный адрес указывает на индекс страницы памяти и смещение внутри страницы.
  - Виртуальное адресное пространство — до 64 Тб, превышает реальный размер памяти (реально меньше, страницы памяти обычно меньше 4 Гб, от 4 Кб). Доступ к страницам других процессов запрещен.
  - Страница снабжается защищающими атрибутами (разрешено чтение/запись/выполнение), адресом начала страницы, размер страницы, куда отображена страница: **страница может находиться не в ОЗУ** — подкачка (вытеснение), сохранение на диск.
- Лимит 4 Гб ОЗУ преодолим (PAE — до 64Гб), лимит АП процесса — нет.
- 64-битный адрес: аналогично, до 16 эксабайт АП процесса и памяти.

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- ➔ Управление памятью.
- Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

Управление свободной памятью

- Представление занятых/свободных блоков: битовая матрица (не используется из-за большого объема), связный список (таблицы).
- **Алгоритм поиска свободного места**: первый доступный, ближайший доступный (от позиции в списке), **наиболее подходящий** (по размеру).

Свопинг и виртуальная память

- **Свопинг**: сбрасывание всех данных приостановленного процесса из ОЗУ на долговременный носитель.
  - Проблема: если процесс занимает гигабайты потребуются секунды или десятки секунд на переключение процессов.
- **Виртуальная память**: сбрасывание части страниц памяти, занятых процессом и не используемых им, восстановление по мере обращения.
  - Проблема: как определить страницу, которую нужно исключить?

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- Управление памятью.
- ➔ Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

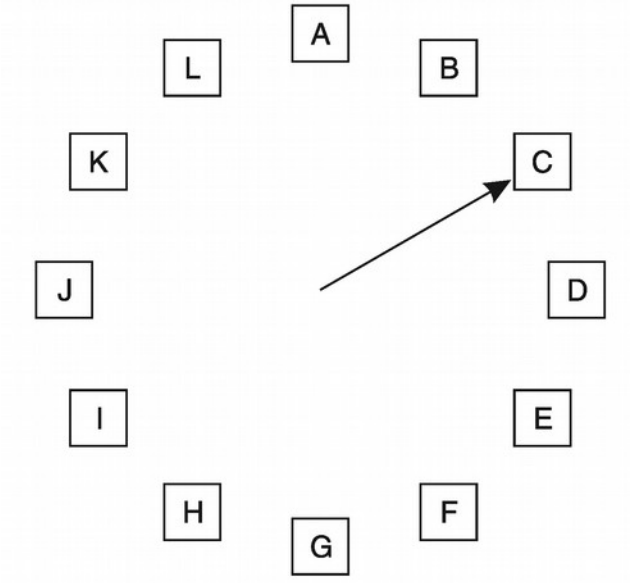
Алгоритмы замещения страниц (1)

Могут рассматриваться как алгоритмы замещения страниц при любой форме кеширования.

- **Оптимальный алгоритм**: удалить страницу, которая не будет использоваться дольше всего.
  - Не реализуем (может использоваться как алгоритм экспериментального сравнения).
- **Случайный выбор**.
  - Легко реализуем, далек от оптимального.
- **NRU** (Not Recently Used): исключение не использовавшейся недавно страницы (поддержан аппаратно).
  - Всякая страница помещается битом **R** в момент обращения (чтения и записи) и **M** в момент модификации (записи). При прерывании по таймеру биты сбрасываются.
  - Классы страниц по последнему кванту: 0 (не было ни чтений, ни модификаций), 1 (чтений не было, но была модификация), 2 (были чтения, но не было модификаций), 3 (были и чтения и модификации).
  - Исключается страница с наименьшим классом.

Алгоритмы замещения страниц (2)

- **FIFO** (Первой пришла — первой ушла).
  - Легко реализуем, далек от оптимального.
- **Второй шанс** (модификация FIFO): удаление первой неиспользуемой недавно страницы (если есть, иначе удаление первой).
- Алгоритм "**часы**": циклический список вместо очереди: проверяется страница, на которую указывает стрелка.
  - Если R=0, страницы выгружается;
  - Если R=1, R сбрасывается, стрелка переходит на следующую страницу.



(C) Э. Таненбаум, Х. Бох

Алгоритмы замещения страниц (3)

- Замещение наименее востребованной, **LRU** (Least Recently Used)
  - Каждая страница снабжается полем для хранения счетчика, увеличивающегося после каждой команды процессора. При обращении к странице записывается значение счетчика, вытесняется страница с наименьшим значением поля.
  - Требуется аппаратная поддержка.
- Алгоритмом нечастого востребования (**NFU**, Not Frequently Used)
  - При каждом прерывании от таймера ОС сканирует все находящиеся в памяти страницы, добавляя к программному счетчику значение бита R (0 или 1), сбрасывая бит. Вытесняется страница с наименьшим значением счетчика.
  - Алгоритм ничего не забывает: если страница использовалась давно, но интенсивно, она будет иметь большое значение счетчика.

Алгоритм старения

- N байтовый счетчик, изначально все биты 0. Каждый квант значение счетчика сдвигается вправо (делится на 2), слева устанавливается значение бита R, который зануляется. Вытесняется страница с наименьшим значением счетчика:
- Значение R по квантам (10 последних квантов) для страниц:

A	B	C
1 1 0 0 1 0 0 1 0 0	0 0 1 0 1 1 0 1 0 0	0 1 0 0 0 0 0 0 1 1

Значение счетчика (1 бит) по тактам:

00000000	00000000	00000000
10000000	00000000	00000000
11000000	00000000	10000000
01100000	10000000	01000000
00110000	01000000	00100000
10011000	10100000	00010000
01001100	11010000	00001000
00100110	01101000	00000100
10010011	10110100	00000010
01001001	01011010	10000001
00100100	00101101	11000000

Вытеснится страница A.

- Отличия от LRU:
  - счетчик вычисляется с точностью до кванта, не до операции;
  - счетчик "быстро забывает прошлое": обращения 8N+1 тактов и 8N+M, M>>1 тактов неразличимы.

Рабочий набор

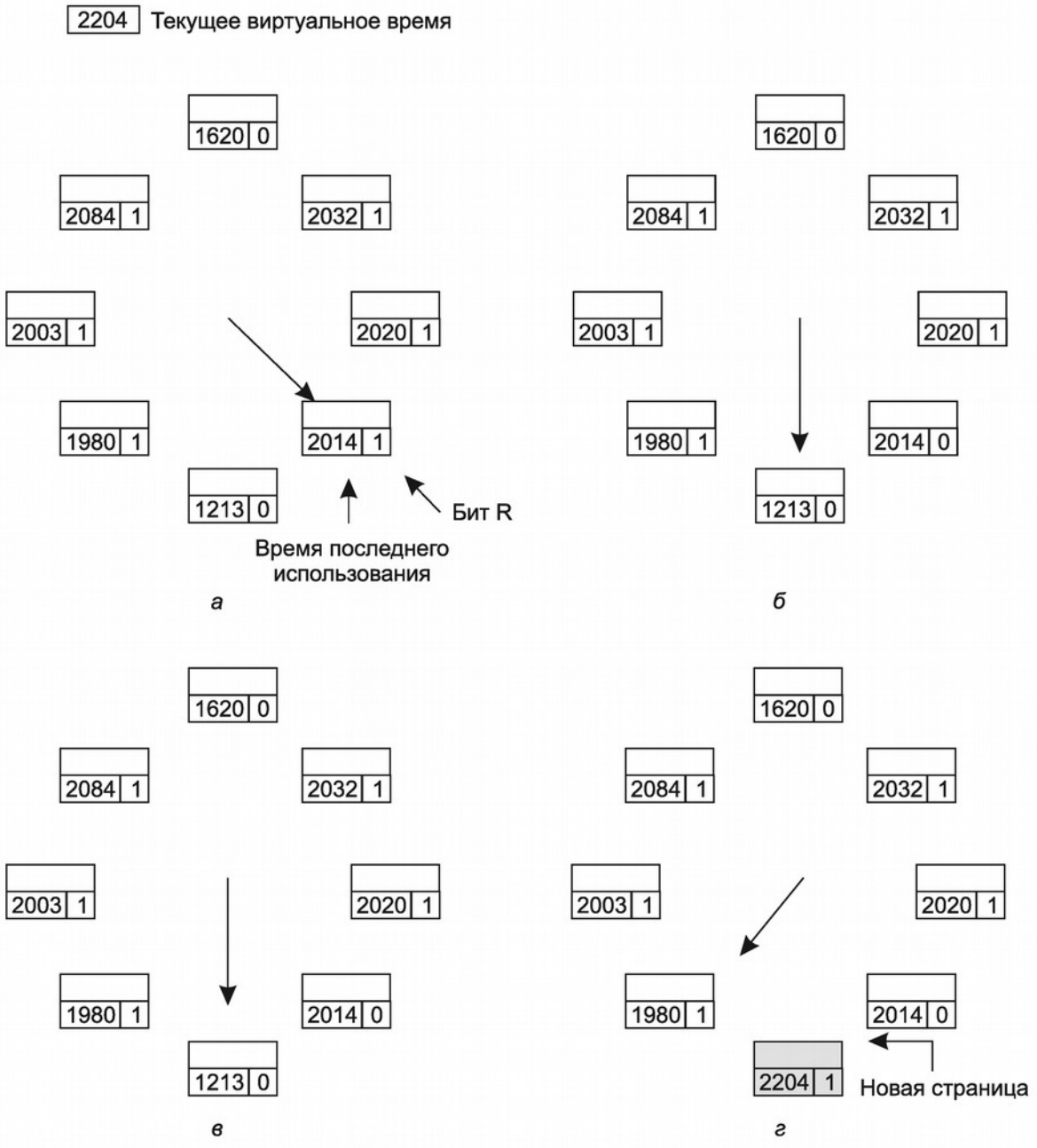
- Рассмотренные алгоритмы подгружают страницы по первому требованию. Хорошая идея — опережающая загрузка страниц. Не всех, а только тех, которые нужны процессу сейчас, т.е. рабочему набору.
  - Существование рабочего набора для большинства программ подтверждается практикой.
- **Рабочий набор** (W), как функция от числа обращений к памяти (k) — набор страниц, к которым было обращение за прошлые k обращений. W(k) растет с k, достигая полное число страниц процесса.
- Приближение: вместо k использовать время (**виртуальное время процесса**, т.е. время, в течении которого процесс задействовал процессор).
  - Каждый квант проверяется бит R, при R=1 сохраняется текущее виртуальное время r. К рабочему набору принадлежат страницы, у которых t-r<k.
  - Вытесняются страницы, не принадлежащие к рабочему набору, если нет — страницы с наименьшим r.
- Слишком трудоемкий алгоритм.

WSClock (1)

- **Часы с информацией о рабочем наборе** — модификация алгоритма "часы". В узлах списка содержится
  - виртуальное процессорное время последнего использования P (обновляется каждый акт прерывания для страниц с R=1);
  - значения битов R и M.
- **Страница относится к рабочему набору, если время ее последнего использования больше, чем некоторое время T. Алгоритм предпочитает вытеснить неизменные страницы, непринадлежащие рабочему набору.**
- Проверяется страница, на которую показывает стрелка.
  - Если R=1 страница не удаляется, R сбрасывается, стрелка переводится.
  - Если R=0 и страница не относится к рабочему набору.
    - Если M=0 копия страницы уже есть на диске — она удаляется и замещается новой.
    - Если M=1 планируется сохранение страницы на диск, стрелка движется дальше.
- Если по окончании круга не была вытеснена страница..
  - Если были запланированы сохранения страниц, они производятся, биты M сбрасываются. Алгоритм повторяется.
  - Если запланированных записей нет, выбирается ближайшая (по стрелке) неизмененная страница, не принадлежащая рабочему набору, если неизмененных страниц нет — выбирается выбирается ближайшая (по стрелке) страница, не принадлежащая рабочему набору.
  - Если страниц, не принадлежащих рабочему набору нет, выбирается ближайшая неизмененная, если нет и таких — измененная страница.

WSClock

- Часы с информацией о рабочем наборе (Working Set).



(C) Э. Таненбаум, Х. Бос.

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- Управление памятью.
- Алгоритмы замещения страниц.
- ➔ Файловые системы.
- Магнитные диски.

Уровни абстракции доступа к информации на долговременном носителе

- **Физический носитель** (диск, лента, твердотельный, др.).
- **Контроллер** (микрокомпьютер, управляющий устройством).
- **Интерфейс (взаимодействия с компьютером)**: IDE, SATA, SCSI, USB, др.: передача команд контроллеру.
- **Драйвер интерфейса** (на уровне прошивки/ОС, HAL): чтение блока, запись блока.
- **Файловая система (драйвер)**
  - предоставление особой абстракции "файл" (поименованная упорядоченная совокупность байт),
  - размещение файлов на носителе,
  - может быть не связана с носителем информации и обращаться к интерфейсу ОС для доступа к данным, фактически находящимся на других ФС (образы дисков, надстройки), компьютерах (сеть) и т.п.
- **Интерфейс ОС**.
  - доступ к файлам на носителях, разграничение доступа пользователей.
- **Функции ЯВУ** (кроссплатформенный доступ?).

Файловые системы

- Разделяют
  - ФС носителей информации,
  - виртуальные ФС (сетевые и др.),
  - ФС операционной системы (совокупность подключенных ФС носителей и виртуальных ФС),
  - не файловые "ФС" (swap).
- Традиционно
  - файлы объединяются в каталоги, **каталог** — особый файл, содержащий файлы и другие каталоги (корневой каталог, подкаталоги →дерево каталогов); в CP/M (DOS/Windows) добавляется "буква диска";
  - **полный путь** к файлу:
    - от корневого каталога, разделить / (*nix), разделитель > (MULTICS): /path/to/file
    - от буквы "диска" (D:), разделитель \ (CP/M, DOS, Windows) d:\path\file
  - всякий процесс имеет текущий (рабочий) каталог
    - **относительный путь**: **..** (на уровень выше), **.** (текущий каталог).
  - файл имеет имя и расширение (условное обозначение формата содержимого).

Разделы

- Разбиение "диска": лимиты ФС, размещение ФС различных ОС.
- Первый сектор диска содержит информацию о разделах, загрузчик ОС: MBR (не более 4 разделов, не более 2 Тб), GPT (10^{22} байт) или является первым сектором ФС (нет разделов).
- Разделы: разделы ФС, расширенный раздел (MBR).
  - ФС носителя информации → ФС раздела.

Доступ к файлам

- Последовательный доступ (быстрее), произвольный доступ (медленнее).
- Кеширование: на уровне носителя (кеш ОЗУ, SSD кеш в HDD), на уровне ОС (проблема потери кеша записи при отказе ПО, сбое питания).

Файловые операции

- Операции с файлами: создание, удаление, переименование, открытие, закрытие, чтение, запись, переход по ссылке, получение атрибутов, установка атрибутов.
- Операции с каталогами: создание, удаление, переименование, открытие, закрытие, чтение, получение атрибутов, установка атрибутов.

ФС носителей информации

- Последовательное размещение файлов (первые компьютеры, стримеры, CD-DA, DVD Video, ...).
  - Проблемы: параллельная запись, необходимость уплотнения при удалении.
- Размещение файлов фрагментами по блокам/кластерам. Проблемы: расход места на неполные кластеры, **фрагментация**.
  - Таблица размещения файлов (FAT-12, FAT-16, FAT-32): число записей равно числу кластеров, в каждой записи — либо следующий кластер файла, либо признак конца файла.
    - Проблема: большой размер кластера или большая таблица размещения файлов (макс. размер FAT-12 — 2 Гб, лимит на размер файла 2 Гб).
  - Размещение в i-узлах (**inode** — ufs, ext*): списки блоков, занятых данным файлом, **MFT** (NTFS, могут содержать небольшие файлы), аналогичные структуры.

Другие особенности ФС

- **Атрибуты** файлов и каталогов:
  - размер файла (у каталога — размер записи);
  - времени создания, изменения, обращения;
  - информация о владельце (пользователе) файла;
  - информация правах доступа (владельца, группы, конкретных пользователей, остальных пользователей), информация о способе доступа (только чтение, возможно выполнение, "скрытый", "системный");
  - иная информация (сжатый, зашифрованный, др.)
- Размещение информации в каталоге: записи фиксированной длины, записи переменной длины. (Файловые системы DOS: имя 8.3, один штамп времени, первый кластер, 6 атрибутов).
- Иные ситуации:
  - **жесткие ссылки** (один файл в нескольких каталогах: в каталоге ссылка на inode, в inode счетчик ссылок), **символические ссылки** (файлы, каталоги);
  - многопоточные файлы (NTFS);
  - не обычные файлы: файлы устройств, каналы и др.

Журналируемые ФС

- Проблема отказа оборудования и сбоев электропитания: нарушение структуры ФС.
  - Необходимость проводить проверку ФС на наличие ошибок — исправление ошибок ведет к потере данных.
- Журнал ФС: сохранение информации о запланированных и выполненных действиях (транзакции).
  - Повышает устойчивость ФС (ext4, NTFS).

Примеры (классификация) ФС

- По ОС: linux (**ext4, btrfs**), DOS (FAT-12, FAT-16), Windows 95/98/Me (FAT-12, FAT-32), Windows NT/... (FAT-32, **NTFS**, **exFAT**), Mac OS (**HFS+**), Solaris (zfs).
- По носителям: CD/DVD (**ISO-9660, UDF**), Flash (**F2FS**).
- Серверные (**ReiserFS**, **xfs**).
- По именам файлов: стандарта 8.3 (**FAT**, ISO-9660), длинные имена файлов (FAT-16, FAT-32, RockBridge, Joliet), до 256+ символов Unicode (ext*, NTFS), регистронезависимые (DOS/Windows), регистрозависимые (*nix).
- Виртуальные (**nfs**, webDAV, MTP, **sshfs**, smbfs), FUSE, архивные файловые системы (**squashfs**), экзотические (WikipediaFS).

Память и хранение данных

- Память и устройства хранения информации.
- Адресация оперативной памяти.
- Управление памятью.
- Алгоритмы замещения страниц.
- Файловые системы.
- Магнитные диски.

Магнитные диски (1)

- Представляют собой диски, снабженные головками, осуществляющие позиционирование по дорожкам ("цилиндрам") и секторам диска.
- Каждый сектор содержит собственно информацию и память коррекции ошибок (ECC-память), позволяет проверять целостность данных при чтении (контрольная сумма) и восстанавливать незначительные повреждения (биты четности, коды Хэмминга и др.) — низкоуровневое форматирование (высокоуровневое форматирование — разделы, файловые системы).
- Имеется внутренний кеш и буфер контроллера.
  - **Буферизация** — накопление операций вывода данных небольшого объема (побайтовых) в памяти с последующим проведением вывода поблочно/посекторно (например, ~512 байт) и наоборот.
- **Алгоритм планирования операций ввода-вывода (перемещения головок HDD)**:
  - **первый пришел первый обслужен** (большое количество перемещений, но подходит для **SSD**);
  - запрос на ближайший цилиндр (риск бесконечного ожидания запросов на дальние цилиндры);
  - **алгоритм лифта** (перемещение от края к центру до самого близкого к центру цилиндру из очереди запросов с обслуживанием запросов на промежуточных цилиндрах, в т.ч. вновь приходящих, и обратно, удачен для **HDD**).

Магнитные диски (2)

- Быстродействие носителя определяется
  - свойствами оборудования:
    - скоростью вращения диска,
    - плотностью записи данных,
    - временем позиционирования головок,
    - размером и скоростью кеша;
    - качеством алгоритма кеширования (ПО контроллера).
- Скорость операции ввода-вывода определяется еще и
  - устройством фактического проведения операции:
    - в аппаратном кеше процессора (в "кеше кеша"),
    - в программном кеше или в буфере в ОЗУ,
    - в кеше носителя,
    - непосредственно на физическом носителе,
    - места на физическом носителе (линейная скорость HDD), заполненностью носителя (запись на SSD);
  - способом проведения операции:
    - последовательный или произвольный доступ к блокам на физическом носителе,
    - алгоритм ввода ввода-вывода в ОС (буферизация, кеш, планирования операций).

Магнитные диски (3)

- Нечитаемые сектора
  - логические сбои (устраняются перезаписью);
  - физические сбои (невозможность восстановления):
    - обработка на уровне ОС (старые диски),
    - обработка на уровне контроллера (перемещение сбойных секторов).
- **S.M.A.R.T.** (self-monitoring, analysis and reporting technology) — оценка состояния жесткого диска.
- **RAID** (Redundant Array of Independent Disks) — избыточный массив дисков, параллелизм на уровне накопителей информации.
  - Уровень 0: один виртуальный диск большого объема.
    Уровень 1: каждый диск имеет дубль (ускорение чтения, повышение надежности).
    Уровень 2: отдельные диски для ECC.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- ➔ Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Ввод и вывод

- Устройства ввода-вывода.
- Способы ввода-вывода.
- Программное обеспечение ввода-вывода.

Ввод и вывод

- Устройства ввода-вывода.
- Способы ввода-вывода.
- Программное обеспечение ввода-вывода.

Ввод и вывод

- Управление устройствами ввода-вывода — одна из задач ОС:
  - выдача команд устройствам;
  - обработка ошибок;
  - предоставление удобного и унифицированного интерфейса приложениям;
  - управление устройствами как ресурсами;
  - обеспечение безопасности (прямое обращение к устройствам — только в режиме ядра).
- Не всегда решается на уровне ОС:
  - прямое обращение приложений к значительной части устройств в DOS;
  - прямое обращение к некоторым устройствам в современных ОС (специфические устройства, отладка устройств).
- Значительная часть устройств (ключевых, "стандартных") управляется практически исключительно ОС, точнее — **подсистемой ввода-вывода**.

Устройства ввода-вывода

- Две основные категории устройств
  - блочные (хранят информацию в адресуемых блоках фиксированной длины) — носители информации (реальные, виртуальные, разделы);
  - символьные — обрабатывают поток символов (мышь, аудиосистема, принтер, сетевые устройства и др.).
- Разделение условно:
  - является ли стример блочным устройством?
  - являются ли часы символьным или блочным устройством?
  - выступает ли жесткий диск как символьное устройство при передаче управляющих команд?

Особенности устройств ввода-вывода

- Взгляд на устройства ввода-вывода
  - инженера: физическая сторона (микросхемы, питание, двигатели и др.);
  - программиста: интерфейс (команды, воспринимаемые устройством).
- Большой диапазон рабочих скоростей:
  - клавиатура 10 байт/с
  - мышь 100 байт/с
  - лазерные диски 20 Мбайт/с
  - SATA 3, USB 3.0 600 Мбайт/с
  - Сеть 1 Гбайт/с
- Скорости совершенствуются, но не всегда это необходимо.
  - Скорость интерфейса может значительно превышать реальную скорость передачи данных устройства.

Контроллер устройства

- Устройства ввода-вывода состоят из механической и электронной части — **контроллера** (плата, например, с разъемом для подключения к компьютеру).
  - Задача контроллера — **преобразование последовательности байтов в аппаратные сигналы и обратно**.
- Интерфейс между контролером и устройством — интерфейс очень низкого уровня, контролер м.б. индивидуален для каждого устройства (зависит не только от производителя, но и модели, серии).
  - Интерфейс между компьютером и контролером — интерфейс низкого уровня — часто попадает под какой-то стандарт — ANSI, ISO, IEEE или соглашение, ставшее стандартом де-факто — например, ATA, USB, SCSI, IEEE 1394, производители устройств подстраивают контролеры под стандарт.

Ввод и вывод

- Устройства ввода-вывода.
- ➔ Способы ввода-вывода.
- Программное обеспечение ввода-вывода.

Связь контроллера и ЦПУ

- У контролера могут быть:
  - регистры управления;
  - буфер данных (видеопамять и т.п.).
- Вариант 1 — использование портов ввода-вывода.
  - Каждому устройству (регистру управления) сопоставляется уникальный номер порта.
  - Для передачи и получения данных используются специальные машинные команды (обычно доступны только в режиме ядра).

```
IN  REGISTER, PORT
OUT REGISTER, PORT
```

- Вариант 2 — отображение ввода-вывода на адресное пространство (памяти).
  - преимущество: использование обычных команд доступа к памяти (в т.ч. команд типа TSL), облегчение контроля со стороны ОС за доступом;
  - проблемы: кеширование, разделение адресных пространств.

Прямой доступ к памяти (DMA)

- Побайтный обмен данными: нерационально.
- Прямой доступ устройств ввода-вывода к блоками памяти минуя центральный процессор: эффективнее.
  - Осуществляется с помощью особого DMA-контроллера.
  - DMA-контроллер имеет прямой доступ к системной шине независимо от ЦПУ (прямой доступ к памяти и устройствам).
  - В регистры DMA-контроллера записывается адрес памяти, порт устройства, счетчик байт и дается команда на ввод или вывод указанного числа байт, по окончании которого генерируется прерывание.
  - Во время операции ввода-вывода ЦПУ продолжает работать над другой задачей.
  - DMA-контроллер работает с физическими адресами, задача ОС и MMU обеспечить получение данных приложением в нужных виртуальных адресах.
- Отказ от DMA: если ЦПУ быстрее и обычно простаивает.

Аппаратные прерывания (1)

- Инициируются устройствами ввода-вывода по завершении операции (запрошенной системой, автоматической или инициированной пользователем), выставляется особый бит шины.
- Проверку выставленных сигналов осуществляет **контроллер прерываний**. Если система занята обработкой другого прерывания или прерывания запрещены, то устройство какое-то время игнорируется.
- Если прерывание возможно, то контроллер передает соответствующий устройству номер прерывания ЦПУ.
- ЦПУ по номеру прерывания в особой таблице — **векторе прерываний** — находит адрес обработчика прерываний и передает управление ему.
  - В той же таблице могут быть реализованы и **программные прерывания** (системные вызовы).
- Проблема: где сохранить состояние (регистры) текущего процесса:
  - в стеке процесса (проблема, так как в процессе может быть ошибка);
  - в стеке ядра (более безопасно, но несет доп. накладные расходы).

Аппаратные прерывания (2)

- Проблема: конвейерезация и суперскалярность. Если команды выполняются последовательно, после каждой из них можно проверить на наличие прерывания. А если часть команд еще не завершилась?
- **Точное прерывание**: требуется, чтобы все команды, до счетчика команд, были выполнены, после счетчика — не выполнены (а все изменения, которые произошли, если они уже начали выполнятся, были откачены перед прерыванием). Требуют время на выполнение данного требования.
- **Неточное прерывание**: допускаются прерывания когда команды имеют разную степень выполнения. Требуют записи большого объема данных в стек, сложный процесс прерывания. Может оказаться более медленным вариантом.
- Гибридный вариант: часть прерываний точная (например, ввода-вывода), часть — неточная (например, системные).

Ввод и вывод

- Устройства ввода-вывода.
- Способы ввода-вывода.
- ➔ Программное обеспечение ввода-вывода.

ПО ввода-вывода

- Способы осуществления операций ввода-вывода:
  - программный ввод-вывод (прямая работа с регистрами устройства);
  - ввод-вывод с помощью прерываний;
  - ввод-вывод с использованием DMA.
- Уровни программного обеспечения ввода-вывода:
  - аппаратура (прошивка);
  - обработчики прерываний;
  - драйверы устройств;
  - устройство-независимое ПО операционной системы (файловая система, спулер принтера);
  - библиотеки ЯВУ и пользовательское ПО.
- Способ передачи данных:
  - синхронный (блокирующий) — физические операции ввода-вывода;
  - асинхронный (неблокирующий) — системные вызовы.

ПО аппаратуры ввода-вывода

- Исполняется на микросхеме контроллера.
- В некоторых случаях может быть обновлено пользователем с помощью специальных утилит.
- Устройство ввода-вывода — отдельная вычислительная система, архитектура многих из них не документируется производителями (взломы и реверсивная инженерия: например, про контроллеры жестких дисков известно, что там используется многоядерный ARM-процессор, ресурсов хватает на запуск linux).

Обработчик прерываний

- Является частью ядра ОС или драйвера, должен быть "спрятан как можно глубже".
- Должен быть короткой процедурой — как можно быстрее освободить ЦПУ и разрешить другие прерывания.
- Основная задача — извлечь информацию из регистров контроллера устройства, вызвавшего прерывание.

Драйверы устройств (1)

- Индивидуальны для каждого типа устройства и каждой ОС. Предоставляются
  - производителем оборудования;
  - создателями ОС;
  - создателями ЯВУ, системный и прикладных программ (в настоящее время крайне редко).
- Могут внедряться как часть ядра ОС (работают в пространстве ядра, большинство современных ОС) или работать в пространстве пользователя (микроядро, позволяет изолировать сбои в драйверах от ядра ОС).
- ОС должна учитывать возможность подключения к ней драйверов: унифицированный **интерфейс подключения драйвера**, обычно драйвер относят к одному из немногочисленных типов (символьные устройства, блочные устройства и т.п.).
- Обычно для драйверов запрещены системные вызовы, но возможно взаимодействие с ядром посредством вызова функций ядра.

Драйверы устройств (2)

- Классы драйверов устройств:
  - драйверы физического устройства (оборудования);
  - драйверы логического устройства (дисковых разделов, файловых систем, виртуальных дисков и др.);
  - драйверы псевдоустройств (/dev/null, /dev/random).
- Большинство ОС запрещает прямое взаимодействие с устройством — прямой доступ имеют только драйвера.
- Драйверы также могут решать задачу оптимизации работы устройства.

Устройствонезависимое ПО (1)

- Возможность создания программ, работающих с любыми устройствами (например, файл на жестком диске, файл на лазерном диске).
- Единообразие именования устройств ("все есть файл" или строка-имя устройство).
- Обработка ошибок (при вводе-выводе распространены больше, чем в других сферах компьютерных устройств):
  - различают
    - ошибки программирования (запись в устройство ввода, неверный адрес буфера и др);
    - фактические ошибки (поврежденный дисковый блок, иной отказ оборудования).
  - фактическая ошибка должна быть обработана как можно ближе к аппаратуре (хорошо, если ее может исправить контроллер, но это не всегда так);
  - реакция: игнорирование, попытка исправить, вывод сообщения, вывод диалогового окна, прерывание программы и др.

Устройствонезависимое ПО (2)

- **Буферизация**.
  - Прямой ввод-вывод: слишком частые прерывания.
  - Буферизация в пользовательской программе: необходимо фиксировать буфер в памяти (снижается эффективность выгрузки страниц);
  - Буферизация в ядре и в пространстве пользователя (копирование данных из буфера ядра в буфер пользователя по заполнении первого): что делать с данными, поступающими в момент копирования?
  - Двойная буферизация в ядре: пока данные одного копируются, заполняется другой.
  - Кольцевая буферизация.

  Буферизация не то же самое, что кеширование.
- Блокирование и высвобождение ресурсов.
- Предоставление устройствонезависимого размера блока.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- ➔ Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Пользовательский интерфейс

- Совокупность средств, методов и правил взаимодействия между компонентами системы.
  - программный интерфейс (API, ABI) —взаимодействие программных продуктов;
  - аппаратный интерфейс — взаимодействие компьютера и подключаемых устройств;
  - пользовательский интерфейс — взаимодействие пользователя и компьютера.
- Пользовательский интерфейс.
  - Осуществляется с помощью **интерфейсного оборудования** (относится к устройствам ввода-вывода, но не все устройства ввода-вывода являются таковым).
  - Один из предметов **человеко-машинного взаимодействия** (HCI, человеко-компьютерное взаимодействие).

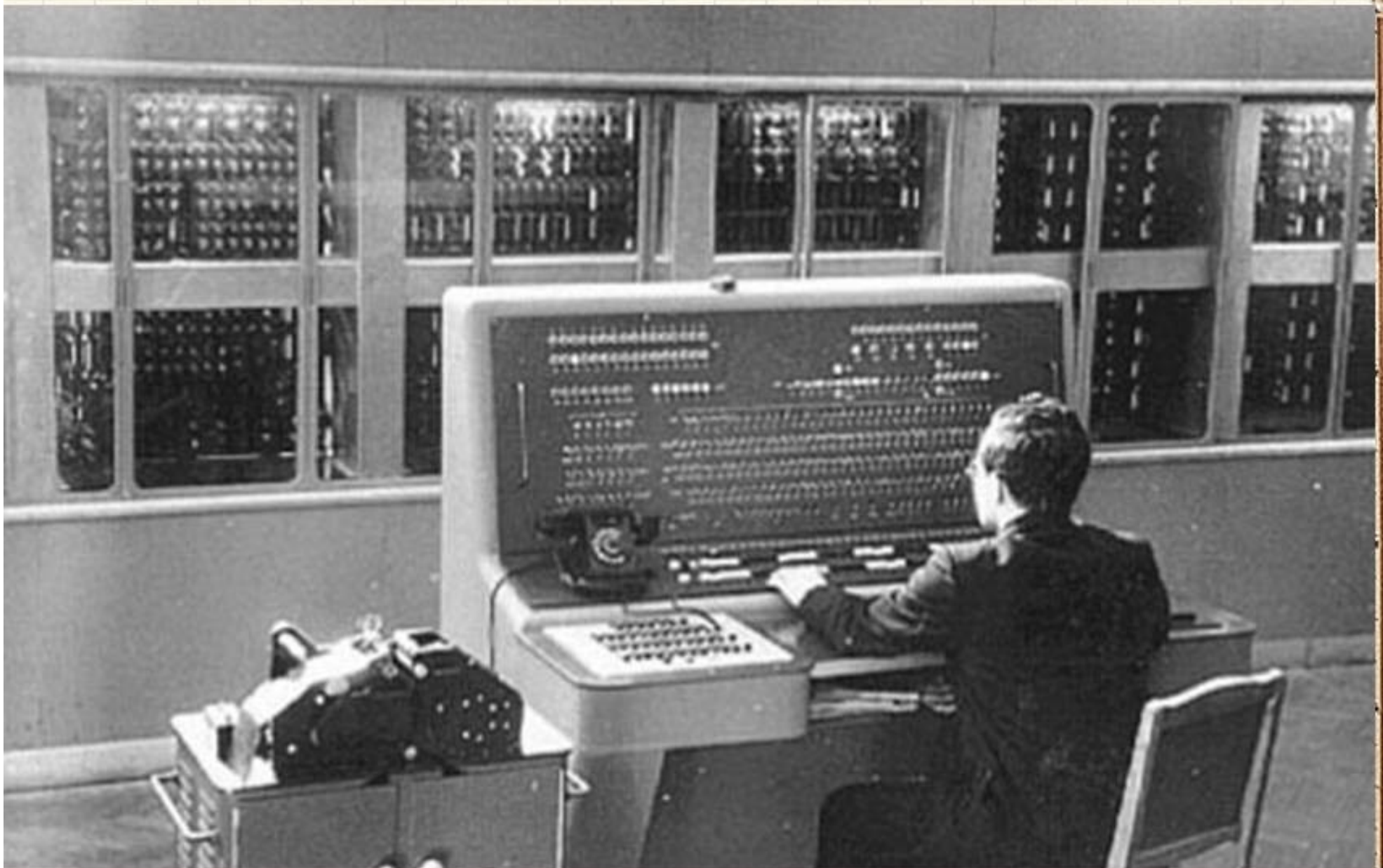
Человеко-компьютерное взаимодействие

- Две основные задачи:
  - передача команд компьютеру (ввод);
  - получение результатов работы компьютера (вывод);
  - кроме того: получение компьютером информации о человеке (в интересах здоровья людей, с целью лучшего решения поставленной перед компьютером задачи).
- Осуществляется разными способами:
  - классическим образом ("работа" с компьютером: "обычный" компьютер, смартфон, навигатор, др.);
  - управление устройств посредством встроенного компьютера (телевизор, стиральная машина);
  - поддержка работы устройств со стороны компьютера (бортовой компьютер автомобилей, ВС).

Оборудование для человеко-компьютерного взаимодействия

- Основные виды устройств:
  - пульты, индикаторы, простые дисплеи;
  - экраны (в т.ч. сенсорные), клавиатуры, мыши (тачпады, джойстики, рули), аудиосистема, видеокамера.
  - устройства управления техникой (штурвал).
- Эволюция устройств ввода:
  - переключатели, кабели, перфокарты, пульты, клавиатура;
  - мышь, микрофон;
  - сенсорные экраны, видеокамеры;
  - ???
- Эволюция устройств вывода:
  - индикаторы, телетайп, терминалы, принтеры;
  - дисплей, аудио;
  - 3D-очки
  - ???

ЭВМ БЭСМ-2 (1959 год, Институт точной механики и оптики)



Виды интерфейса взаимодействия с компьютером

- Командный интерфейс.
- Текстовый (псевдографический) интерфейс.
- Графический интерфейс.
- Естественно-интуитивный интерфейс.
  - Языковой интерфейс (в т.ч. голосовой).
  - Жестовый интерфейс.
  - Виртуальная реальность.
- Нейрокомпьютерный интерфейс.
  - Управление компьютером, виртуальная реальность.
  - Нейропротезирование.

Роль ОС в Ч-К взаимодействии

- Обеспечение пользовательского интерфейса является одной из задач ОС.
  - ПО, осуществляющее взаимодействие ОС и человека считается **оболочкой ОС**.
- В универсальных пользовательских компьютерах значительная часть пользовательского интерфейса реализуется на уровне прикладным программам.
  - Программы, обеспечивающие или меняющие ключевые, системно-значимые особенности интерфейса, тоже относятся к **оболочкам**.
- Оболочка ОС — надстройка над ОС, обеспечивающая (облегчающая) работу пользователя с ОС (компьютером), интерпретатор команд ОС, обеспечивающий интерфейс взаимодействия пользователя и компьютера.

Оболочки ОС

- **Интерфейсные системы** – модифицируют как пользовательский, так и программный интерфейсы, а также реализуют дополнительные возможности по управлению ресурсами.
- **Программы-оболочки (командные интерпретаторы)** — модифицируют только пользовательский интерфейс, предоставляя пользователю качественно новый интерфейс по сравнению с реализуемым операционной системой.
  - Упрощают выполнение часто требуемых действий функций (запуск программ, файловые операции и др.)
- **Утилиты** — средства обслуживания программного и аппаратного обеспечения компьютера.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- ➔ Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Язык: понятие

- Язык — средство представления информации.
- Язык — средство абстрагирования информации от носителя (возможность передачи информации).
- **Язык** — сопоставление объектам некоторого множества ("первичной реальности"), объектов множества обозначений ("имен").
  - Также могут рассматриваться множество объектов первичной реальности со множеством состояний.
  - По отношению к имени объект первичной реальности называется значением.
- Язык:
  - **синтаксис** — правила образования слов (сочетаний, предложений и т.д.).
  - **семантика** — интерпретация слов (сочетаний, предложений и т.д.).
  - **прагматика** — правила использования слов (сочетаний, предложений и т.д.): в контексте, с учетом внешних факторов и др.

Языки: классификация

- По происхождению
  - **естественные** (возникли в процессе эволюции: человеческие, животных);
  - **искусственные** (созданы целенаправленно)
    - языки общения (эсперанто, эльфийский),
    - **формальные**
      - знаковые (дорожные знаки),
      - наук (терминология, нотация),
      - теорий (**формализованные** — исчисления, теорий),
      - **программирования** (командные).
- По способу представления информации
  - письменные,
  - вербальные (устные),
  - невербальные (жесты).
- Универсальный язык человеческого общения не создан.
  - Гипотеза: родной язык влияет на восприятие мира.

Языки Ч-К взаимодействия

- Язык ввода информации и управления:
  - система управления
  - командный (программирования)
  - естественный
  - нейрокомпьютерный
- Язык вывода информации
  - знаковый (сигнальный)
  - формальный (сообщений)
  - естественный
  - нейрокомпьютерный
- На уровне других интерфейсов: тоже языки (стандартны, протоколы).

Языки программирования

- **Трансляция**: перевод с языка программирования на язык машинных команд.
  - **Компиляция**: перевод всего исходного кода программы исполняемый файл.
  - **Интерпретация**: пошаговое исполнение команд языка программирования.
  - Смешанная форма (байт-код).
- Для всякого компилируемого языка можно создать интерпретатор. Обратное неверно.
  - Конечно, можно прозрачно включать интерпретатор непосредственно в исполняемый файл программы.
- Универсального языка программирования не существует и вряд ли он будет создан.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- ➔ Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Командные интерпретаторы

- "Первичная" оболочка операционной системы.
- Представляет собой интерпретатор некоторого языка (**языка командной оболочки, языка сценариев**): исполняет команды непосредственно по их вводу в **командную строку — интерфейс командной строки (консольный интерфейс)** или из **командного файла (пакетный файл, файла сценария)**.
  - Формат команд регламентирован, формат вывода обычно жестко не определен, программисты стараются поддерживать единообразие в рамках ОС.
  - Различают команды внутренние (исполняются оболочкой) и внешние (запускаемые консольные приложения).
- Преимущества CLI:
  - быстрота набора команд,
  - возможность пакетной обработки,
  - низкий расход ресурсов.

Примеры командных интерпретаторов

- *nix:
  - bash
  - sh
  - zsh
  - cs
- DOS
  - command.com
  - 4dos.com
- Windows
  - cmd.exe

Пример команды (bash)

- Вставить в имена всех текстовых файлов суффикс перед расширением

```
$ for file in *.txt; do mv $file ${file%.*}.suffix.txt; done
```

- Преобразовать все файлы каталога и подкаталогов из одной кодировки в другую

```
$ find . -type f | while read file
  do iconv -f cp1251 -t utf8 "$file" >"$file.new"
     mv -f "$file.new" "$file"
  done
```

- Рекурсивно удалить файлы, содержащие в себе строку "Hello!"

```
$ find . -type f | while read file
  do grep -Fxq "Hello" $file || rm -f $file
  done
```

Человеко-компьютерное взаимодействие

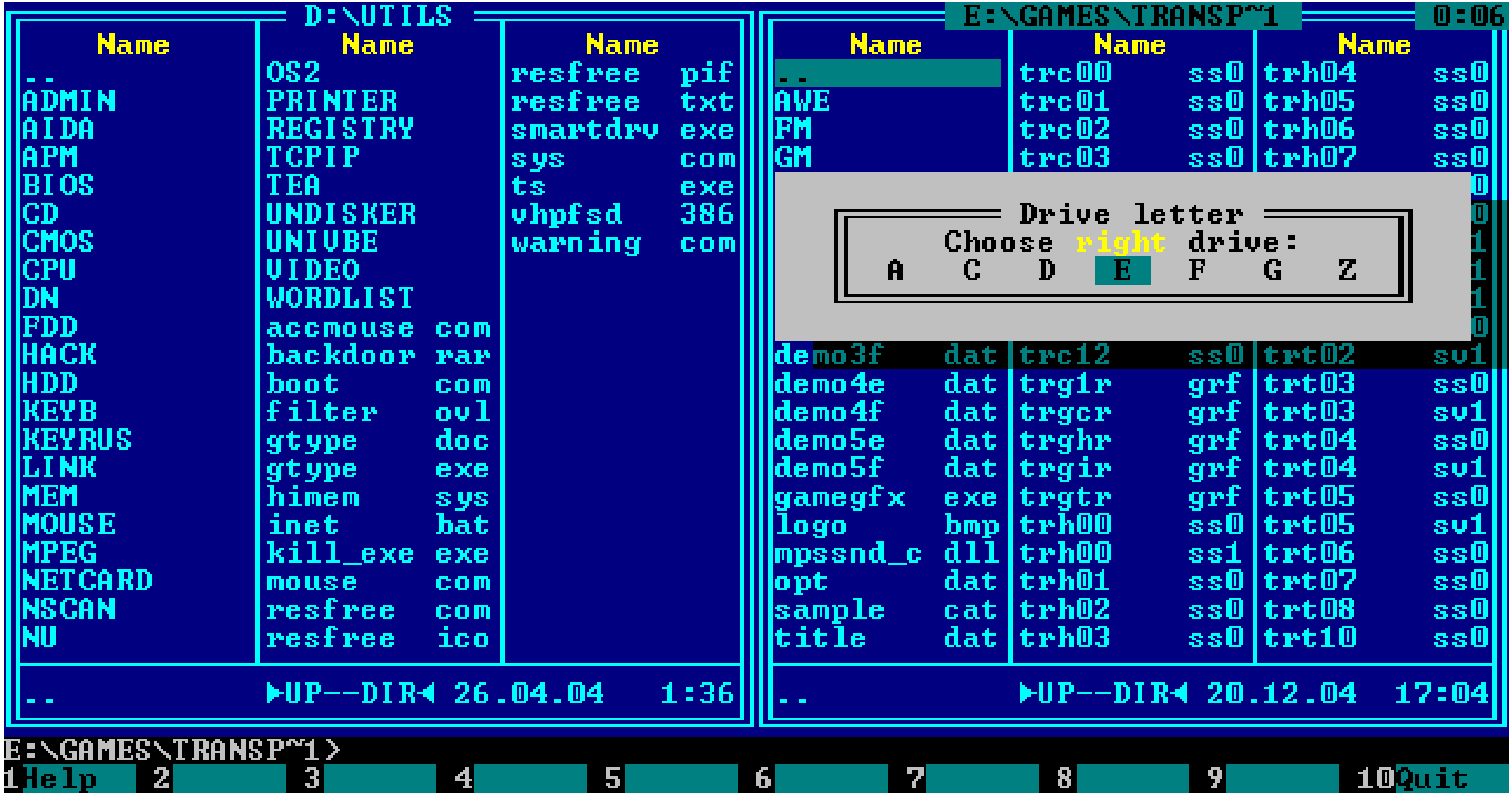
- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- ➔ Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Текстовый интерфейс

- Оконный или подобный интерфейс **псевдографического режима**.
  - Линии и другие элементы прорисовываются с помощью символов псевдографики — особых ASCII-символов.
- Norton Commander: первый двухпанельный файловый менеджер (DOS)
  - современные: Midnight Commander, Far Manager.
- Tetris: первые версия текстовые.

Codepage 437

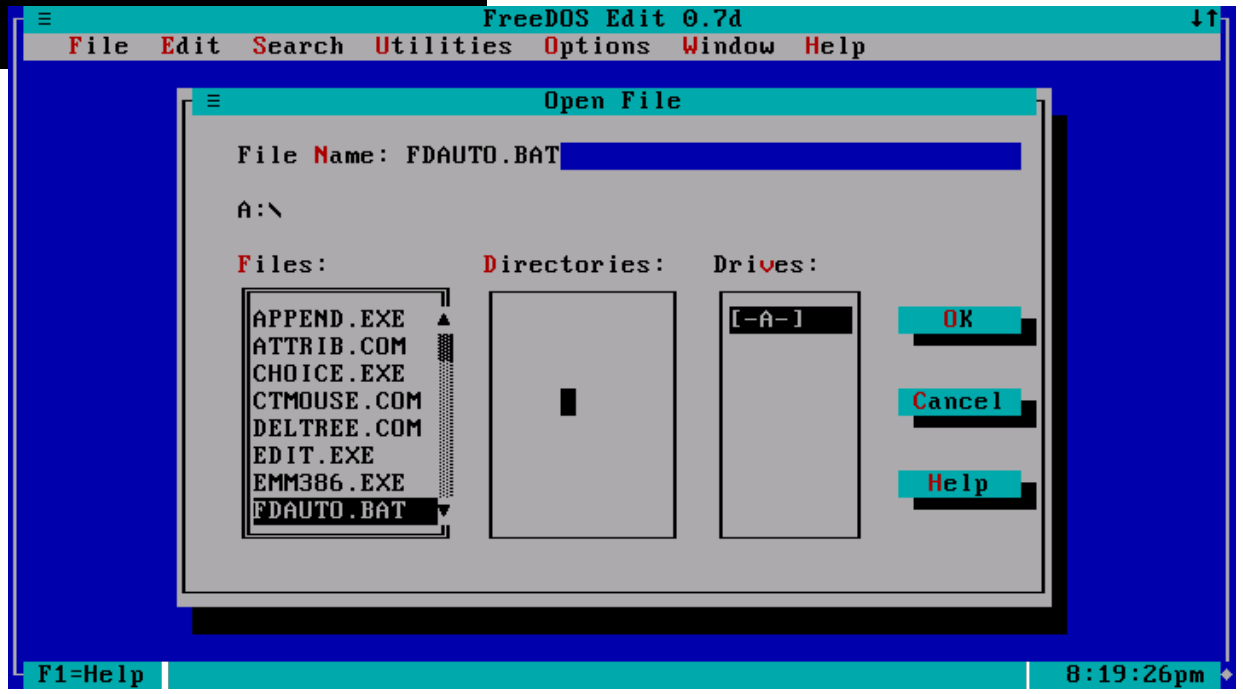
Norton Commander, 1986



Tetris, 1986 г.

Еще примеры...


Synchronet Main Menu
Read/Post Messages
N New message scan
R Read message prompt
Z Continuous new scan
B Browse new scan
Q QWK packet transfer
P Post a message
A Post auto-message
Message Search
F Find text in messages
S Scan for msgs to you
Message Area Selection
J Jump to new msg area
* List sub-boards
/* List groups
{ } # Select sub-board
[ ] /# Select group
Go to
T File Transfer section
G Text file section
C Chat section
X External programs
Electronic Mail
E Read/Send E-mail
Other Commands
D Default user config
& Message scan config
U User lists
I Information
M Minute Bank
/L Node activity
^K Ctrl-key Menu
O Logoff BBS (or /O)
Anytime: Ctrl-U Who's online Ctrl-P Send private msg Ctrl-C Abort cmd/text
• Main • 0:00:14 [1] Main [1] Notices:


FreeDOS Edit 0.7d
File Edit Search Utilities Options Window Help
Open File
File Name: FDAUTO.BAT
A:\
Files:
Directories:
Drives:
APPEND.EXE
ATTRIB.COM
CHOICE.EXE
CTMOUSE.COM
DELTREE.COM
EDIT.EXE
EMM386.EXE
FDAUTO.BAT
[-A-]
OK
Cancel
Help
F1=Help
8:19:26pm

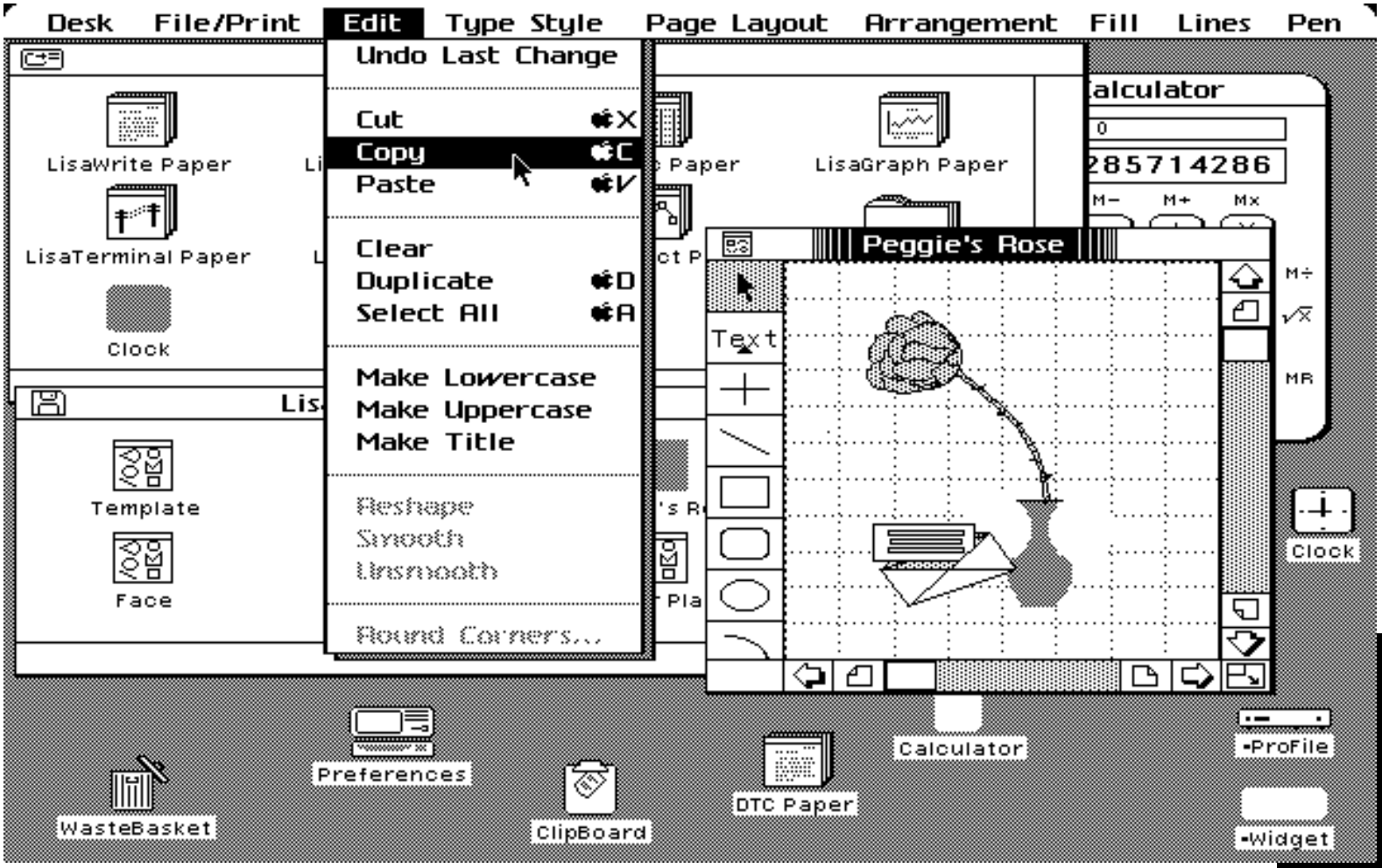
Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Графический интерфейс

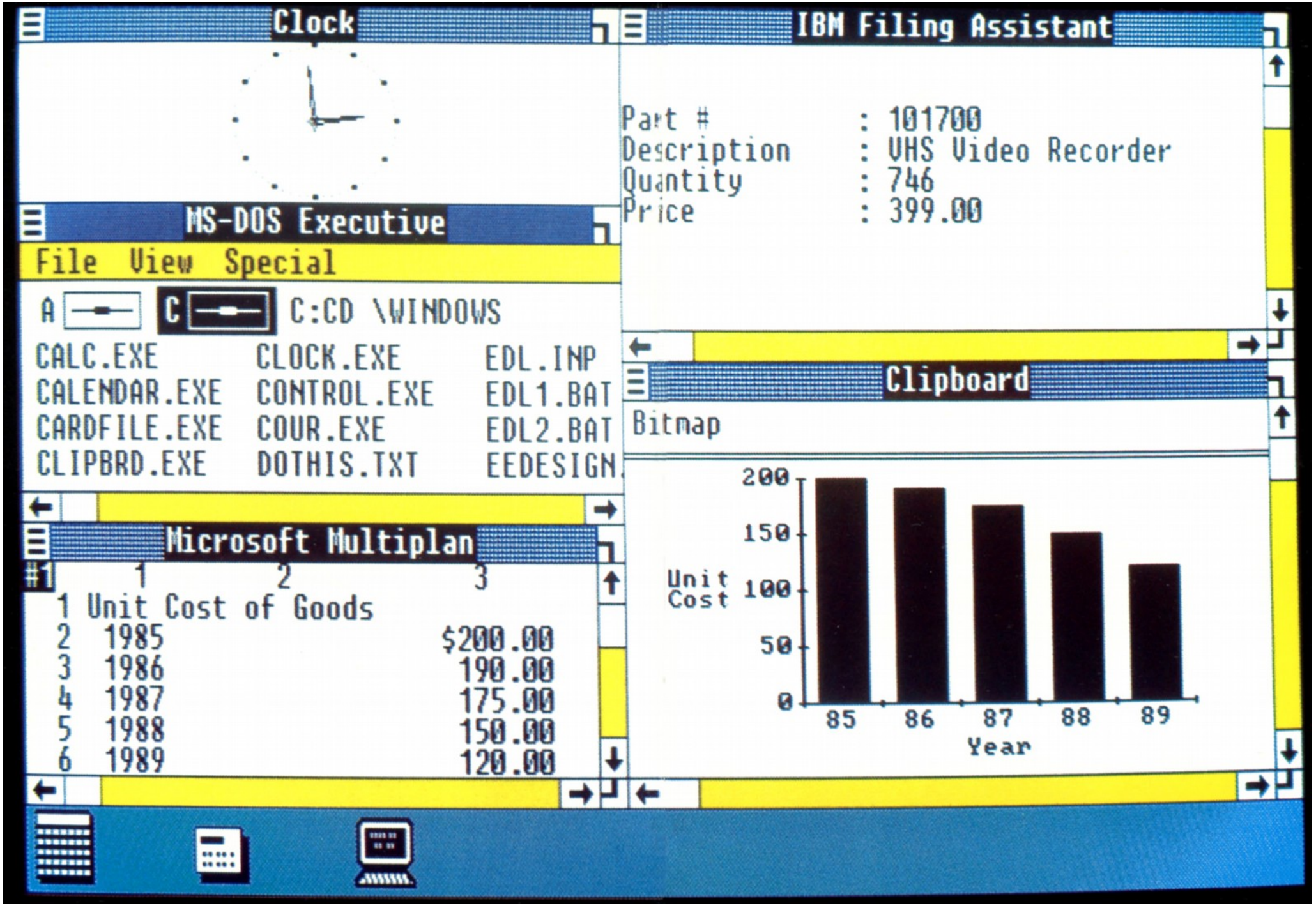
- Индивидуальный (приложений) — переключение при запуске.
  - игры
  - просмотр DVI в emTeX
  - программы просмотра изображений в DOS
- Стандарт современных ОС.
  - WIMP (Window, Icon, Menu, Pointer) — окно, значок, меню, указатель.
- Графические оболочки ("графические окружения рабочего стола") и оконные менеджеры:
  - Windows Explorer
  - GNOME (metacity)
  - MATE (marco)
  - KDE (KWin)
  - LXDE (openbox)
  - др.
- "Плиточный" интерфейс

Lisa, 1983 г.

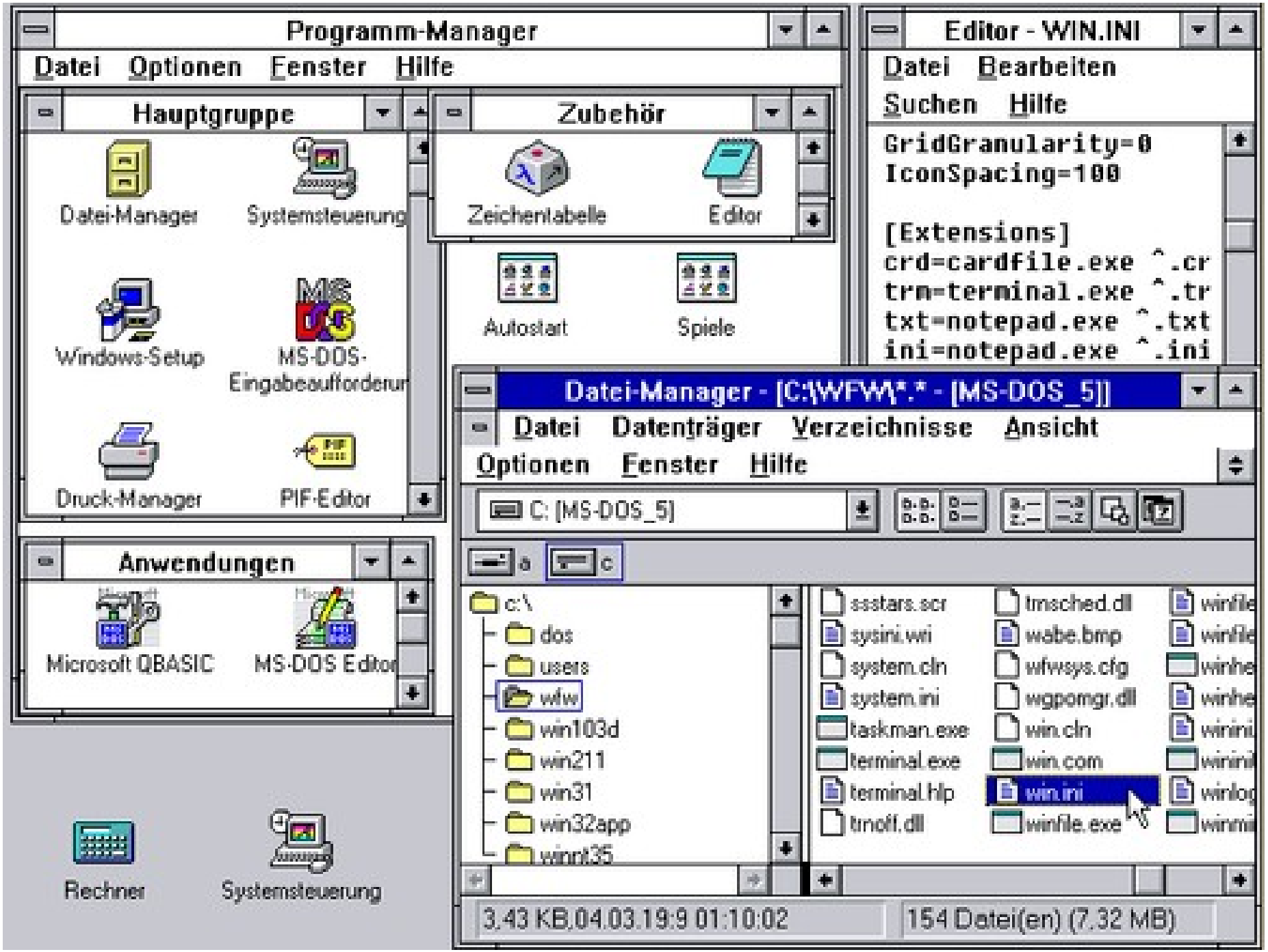


Дисплей: черно-белый
12" 720 x 364

Windows 1.0, 1985 г.



Windows 3.x, 1993 г.



X Window System (1984)

- Обеспечивает базовые функции графической среды:
  - взаимодействие с устройствами ввода-вывода;
  - отображение курсора мыши, отображение и перемещение окон.
- Модель клиент-сервер (возможно удаленное подключение).
  - X server (X-сервер, X.Org Server, X11): работает на "графическом терминале";
  - Клиент: любое графическое приложение. Сервер для подключения определяется переменой окружения.
    `DISPLAY=192.168.0.2:1.2`
    `DISPLAY=`Host`:`Server`.`Screen

    (порт = 6000 + Server)
- История:
  - Консорциум X MIT (~1987, современный протокол X11) и X Consortium (1993).
  - XFree86 (1992): X server для i386 (большая закрытость проекта).
  - The Open Group и X.Org (1996-1997)
  - X.Org Foundation (2004)

Удаленный доступ к графическому интерфейсу

- Подключение к X-серверу другого компьютера не дает возможности управления этим компьютером (фактически осуществляется обратное управление, инициированное управляемым компьютером), средствами X-сервера невозможно подсоединиться к имеющейся сессии.
- С помощью **SSH** можно перенаправить X-соединение (**X Forwarding**).
  - Необходим работающий X-сервер на управляющем компьютере, окна будут отображаться на нем (существуют X серверы для Windows и других платформ).
- **VNC** (Virtual Network Computing) — система удаленного доступа.
  - VNC-сервер запускается на управляемой машине, содержит X сервер (не связанный с физическим интерфейсным оборудованием) и сервер для подключения клиента.
  - VNC-клиент может подключиться к серверу (с любого компьютера к имеющейся на VNC-сервере X-сессии).
- **RD**P (Remote Desktop Protocol) — протокол удаленного рабочего стола Windows. Также состоит из клиентской и серверной части, есть реализации клиентов и серверов для других ОС.

Тонкий клиент

- Пользовательский терминал (компьютер с минимальной конфигурацией, легковесная программа), обеспечивающий доступ к удаленному компьютеру (или даже облаку), на который выносится бОльшая часть обработки информации.
  - В частности: веб-браузер при работе с веб-приложениями.
  - Веб-приложение — еще один способ организовать GUI.

Насыщенные интернет-приложения

- RIA (Rich Internet Application) бОльную часть вычислений производит на клиентском компьютере.
- Преимущества:
  - доступны с любого компьютера, подключенного к интернет;
  - не требуют установки и обновления приложений (кроме браузера).
- Недостатки:
  - ресурсоемкость;
  - зависимость от подключения к интернету;
  - неподконтрольность фундаментальной конфигурации и выбора версии;
  - доступность пользовательских данных третьим лицам.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- ➔ Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Элементы управления

- **Элементы управления** (controls, widgets — windows gadgets, оконные приспособления) — доступные для манипулирования объекты графического интерфейса, посредством которых человек взаимодействуют с программным продуктом.
- Классификация элементов управления:
  - **командные элементы управления** (выполнение функций);
  - **элементы выбора** (выбор данных или настроек);
  - **элементы ввода** (ввод данных);
  - **элементы отображения** (представление данных и непосредственное манипулирование).

Командные элементы управления

- Немедленно выполняют действие.
  - иногда выделяются кнопки по умолчанию;
  - следует изменять внешний вид нажатой кнопки.
- Варианты: текстовые кнопки, кнопки-значки, текстовые гиперссылки, др..
  - кнопки, помещенные на панель инструментов постоянно на виду и легко запоминаются;
  - проблема: неоднозначность понимания пиктограмм.

Элементы выбора

- Позволяют пользователю выбрать из группы доступных объектов тот, с которым будет совершено действие, или вариант настройки.
- Может одновременно быть командным.
- Варианты:
  - флажки, радиокнопки (нуждается в поясняющем тексте);
  - списки, раскрывающиеся списки;
  - выключатели;
  - переключатели, триггеры (обычно одновременно командные, могут сбить пользователя);
  - ...

Элементы ввода

- Дают возможность выбирать существующую и вводить новую информацию.
- Ограничивающий элемент ввода ограничивает доступные для ввода пользователем значения (должен четко информировать о допустимых границах).
- Варианты:
  - счетчики (можно увеличивать/уменьшать значение или вводить непосредственно);
  - рукоятки и ползунки (экономят пространство);
  - поле ввода текста (однострочное, многострочное);
  - ...

Элементы отображения

- Используются для управления визуальным представлением информации на экране
- Варианты:
  - элемент вывода текстовой информации;
  - полосы прокрутки (экономят пространство);
  - разделители (подвижные, неподвижные);
  - битовые карты (вывод изображений и др);
  - ...

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- ➔ Естественно-интуитивный и нейрокомпьютерный интерфейс.
- Дизайн пользовательского интерфейса.

Естественно-интуитивный интерфейс

- Основные типы поступающей информации:
  - руки (движения, положение пальцев, жесты);
  - лицо (распознание лиц);
  - голос (распознание речи);
  - окружение.
- Области применения:
  - взаимодействие с трехмерным миром приложения, распознание жестов и манипулирований — моделирование захвата, перемещение, указывание;
  - идентификация людей, определение моргания, улыбки, пола, возрастной группы;
  - разделение фона и переднего плана;
  - комбинирование объектов реального мира и комбинирование с виртуальным контентом;
  - языковые поисковые системы;
  - разговор с компьютером в автомобиле;
  - компьютерный перевод.

Нейрокомпьютерный интерфейс

- Система для обмена информацией между мозгом и компьютером.
  - Однонаправленная: только принимает сигналы от мозга (протез руки, управление игрой) или только посылает сигналы (протез уха, глаза);
  - Двунаправленная: обмен информации в обоих направлениях.

Человеко-компьютерное взаимодействие

- Пользовательский интерфейс и человеко-компьютерное взаимодействие.
- Язык и его роль в человеко-компьютерном взаимодействии.
- Интерфейс командной строки.
- Текстовый интерфейс.
- Графический интерфейс.
- Элементы управления пользовательского интерфейса.
- Естественно-интуитивный и нейрокомпьютерный интерфейс.
- ➔ Дизайн пользовательского интерфейса.

Интерфейс пользователя: дизайн

- Цель дизайнера: представление информации в понятном и удобном виде.
- Дизайнер — не (совсем) художник:
  - удобство vs эстетический отклик
  - понятность vs своеобразность
- Дизайнер:
  - типографика, композиция;
  - принципы взаимодействия пользователей и компьютеров, идиомы интерфейса.
- Проектировщик интерфейса (планирование "интерфейса в целом")
  - психология, аналитика, практичность, простота использования.

Дизайн интерфейса

- Графический дизайн:
  - внешняя красота интерфейса;
  - общая композиция и стиль;
  - фирменный стиль.
- Визуально-информационный дизайн:
  - визуализация данных (текст, графики, диаграммы и др.) и средств управления;
  - цвет, форма, расположение, масштаб.

Строительные блоки дизайна интерфейсов (1)

- **Форма**
  - главный признак сущности объекта для человека;
  - удобна для указания связи между объектами или их однотипности;
  - плохо подходит для указания контраста;
  - форма (начертание) шрифта имеет значение для выделения (проблема: злоупотребление разнообразием шрифтов обесценивает форму шрифта).
- **Размер**
  - различаться по размеру могут информационные объекты, элементы управления, шрифт и т.д.;
  - более крупные элементы привлекают больше внимания и наоборот;
  - человек автоматически упорядочивает объекты по размеру.

Строительные блоки дизайна интерфейсов (2)

- **Цвет**
  - привлекает внимание;
  - может иметь собственное значение (нести информацию);
  - проблема: цветовая слепота.
- **Яркость**
  - привлекает внимание;
  - подчеркивает контраст между объектами.
- **Текстура**
  - признаки перетаскивания, масштабируемости: засечки, выпуклости;
  - признаки нажимаемости: фаски, тени.

Строительные блоки дизайна интерфейсов (3)

- **Расположение**
  - передача иерархии, отношений между объектами;
  - может зависеть от вида устройства, ориентации экрана;
  - важны разделители (линии, значки, др), может использоваться цветовое разделение.
- **Направление**
  - "Естественное" направление:
    - для европейца – слева направо и сверху вниз;
    - не во всех странах это так.
  - Вторичный, но важный признак.

Ориентированность на пользователя

- **Программирование — сфера услуг**.
  - Программы создаются для пользователей.
  - "Заказчик всегда прав".
- Важность понимания желаний (и нежеланий) пользователя:
  - до 90% новых продуктов провальны,
    до 1/4 провалов связано с неадекватным анализом потребностей пользователя (!).
  - 1/3 — недоработки маркетинга и реакция конкурентов;
    1/5 — дефекты продукта и производственные проблемы.
- Важно понимание пользователя (потенциальных пользователей)
  - обратная связь;
  - модели пользователей.

Понимание пользователя

- Что может предъявить пользователь?
  - **Пользовательские истории** — способ описания требований к разрабатываемой системе, сформулированных на естественном языке пользователя;
  - **Варианты использования** — исчерпывающее описание функциональных требований к системе (фактически, техническое задание).
- Сбор и анализ информации о пользователе:
  - исследование пользователей (интервью, наблюдение, анализ деятельности, задач и т.д.);
  - экспертиза предметной области;
  - аудит конкурирующих и аналогичных продуктов.

→ Определение **профилей пользователей**.
→ Создание **сценариев действия пользователей**.

Для пользователей

- **Принцип наименьшей неожиданности** ("без сюрпризов"): программа должна действовать как следует из ее вида.
  - Аналог в программировании: функция должна делать то, что следует из ее имени.
- **Принцип прозрачности**: интерфейс не должен напоминать компьютерную программу:
  - интерфейс — тоже уровень абстракции (скрывает факт обращения к компьютеру);
  - сразу понятен, не нуждается в руководстве.
- **Производительность интерфейса**:
  - число действий ("правило трех кликов");
  - время поиска элемента ("правило трех секунд").

Для пользователей?

- **Простота в обучении != простота в использовании**.
- **Всем не угодишь**
  - широкая аудитория vs узкая аудитория: большая клиентура vs сложность выхода на рынок?
  - много небольших программ для разных задач vs одна программа для большого класса задач?
  - пользователи разные (программа "для себя" может оказаться удобной для других программистов, но не для широкого класса пользователей);
  - реальность: пользователи не всегда выбирают, крупные производители диктуют условия пользователям, пользователи привыкают к "стандартам";
  - **коммерческий интерес vs интерес пользователя**.
- Что естественно, что интуитивно?
  - Зависимость от среды формирования личности ("культурный слой"), в том числе от образования (профессии) и опыта, экстремальная ситуация: "Маугли".
  - Зависимость от привычек.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- ➔ Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Виртуализация

- Общие понятия
- Виртуальные машины (системные)
- Виртуализация CPU
- Виртуализация памяти
- Другие аспекты виртуализации

Виртуализация

- ➔ Общие понятия
- Виртуальные машины (системные)
- Виртуализация CPU
- Виртуализация памяти
- Другие аспекты виртуализации

Виртуализация

- **Виртуализация** — предоставление набора вычислительных ресурсов, абстрагированное от аппаратной реализации.
- Требования:
  - безопасность;
  - эквивалентность;
  - эффективность.
- Назначение:
  - абстрагирование от оборудования;
  - программное представление недоступного (отсутствующего, несуществующего) аппаратного оборудования;
  - предоставление недоступных (отсутствующих, несуществующих) аппаратных и программных ресурсов и средств исполнения процессов;
  - одновременная работа нескольких ОС на одном оборудовании.
  - ...

Виды виртуализации

- **эмуляция**: DOSbox, эмуляторы NES, QEMU, др**:**
  - наиболее полное предоставление виртуальных ресурсов одной вычислительной системы в другой — высокое качество;
  - машинные команды эмулируемой системы интерпретируются — низкая эффективность.
- **виртуализация оборудования:** HAL, эмуляторы устройств и др.;
- **программная виртуализация (системные виртуальные машины):** VMWare, VirtualBox, Xen, QEMU
  - выполнение большинства команд на физическом оборудовании, эмуляция некоторых команд и ресурсов.
- **виртуализация на уровне процесса** — абстрактная машина, исполнение команд виртуальной архитектуры
  - **виртуальные машины процесса (сред выполнения)**: JVM, CLR;
- **аппаратная поддержка виртуализации**
  - поддержка виртуализации на уровне физического оборудования.
- **слой совместимости** (не эмуляция)
  - альтернативная реализация API: wine, cygwin;

Виртуализация

- Общие понятия
- ➔ Виртуальные машины (системные)
- Виртуализация CPU
- Виртуализация памяти
- Другие аспекты виртуализации

Виртуальные машины

- Использования нескольких компьютеров вместо одного (когда достаточно мощности одного).
  - Преимущества:
    - возможность одновременной работы в разных ОС;
    - запуск разных серверов на разных компьютерах: сбой или взлом одного из них не остановит другие.
  - Недостатки:
    - дороговизна.
- Выход: использование **виртуальных машин** — создание иллюзии работы нескольких компьютеров на одном оборудовании.
  - Дополнительные преимущества:
    - возможность сохранения и отката состояния машины;
    - облегчение миграции и клонирования;
  - Недостатки:
    - сбой оборудования нарушит работу всех виртуальных машин.

Связанные понятия (1)

- **Host (хозяин)**: платформа-хозяин, host-платформа, host-машина, host-система — платформа или ОС, в которой запускается виртуальная среда.
- **Guest (гость)**: Целевая платформа, гостевая платформа, гостевая машина, гостевая ОС — создаваемая среда, запускаемая ОС.
- **Гипервизор (Монитор виртуальных машин)** — программное (или аппаратное) обеспечение, предоставляющее средства запуска виртуальных машин и гостевых ОС.
  - **гипервизор первого типа** является самостоятельной ОС, работающей на оборудовании, гостевые ОС имеют ("почти") прямой доступ к управлению оборудованием (VMware ESX, XenServer) → **паравиртуализация**.
  - **гипервизор второго типа** запускается как пользовательский процесс внутри ОС хоста, ОС хоста отвечает за работу оборудования, гипервизор эмулирует значительную часть оборудования гостевых машин → **полная виртуализация**.

Связанные понятия (2)

- **Снимок** — сохраненное состояние гостевой машины, позволяющее возобновить ее работу с точки сохранения (приостановка гостевой ОС с освобождением ресурсов и/или отключением хоста, откат изменений в гостевой ОС).
- **Модуль ядра гипервизора** (второго типа): устанавливается в ОС хоста, чтобы дать гипервизору возможность выполнять код в режиме ядра, создавать виртуальные устройства и т.д.
- **Модуль ядра и утилиты гостевой ОС**: устанавливаются в гостевую ОС для общения с гипервизором, поддержки обмена данными между гостевой ОС и ОС хоста, создания виртуальных устройств и т.д.
- **Песочница —** среда безопасного исполнения программ **(изоляция).**

Применение виртуальных машин

- изоляция программ и серверов;
- запуск программ, требующих отсутствующего (устаревшего, дорогостоящего) оборудования или операционной системы, отличной от основной рабочей;
- аренда вычислительных ресурсов;
- тестирование программ разработчиками на ОС, отличных от ОС разработки;
- исследование вредоносного кода;
- тестирование сетевых приложений;
- разработка ОС;
- ознакомление с новыми (для пользователя) ОС, учебные цели;
- др.

Проблемы виртуализации

- Эмулировать ли ЦПУ и до какой степени?
  - Пошаговая интерпретация команд: низкая эффективность, используется как вынужденная мера, когда необходимо эмулировать архитектуру целиком.
  - Прямое исполнение всех команд на ЦПУ хоста проблемно:
    - как быть с командами гостевой ОС режима ядра, если гипервизор 2 типа запущен в режиме пользователя? (они не выполнятся);
    - как быть с командами гостевой ОС режима ядра, если гипервизор запущен в режиме ядра? (они могут нарушить гипервизор или ОС хоста);
    - как быть с адресацией памяти?
- Как эмулировать память?
  - таблица страниц памяти гостевой ОС != таблице страниц памяти ОС хоста.
- Как эмулировать другие устройства?
- Важна аппаратная поддержка виртуализации:
  - виртуализация эффективнее, если ЦПУ имеет технологии поддержки запуска гостевых ОС.

Виртуализация

- Общие понятия
- Виртуальные машины (системные)
- ➔ Виртуализация CPU
- Виртуализация памяти
- Другие аспекты виртуализации

Виртуализация ЦПУ (1)

- Ситуация:
  - виртуальная машина запущена в пользовательском режиме, ей нельзя выполнять служебные инструкции;
  - на виртуальной машине работает гостевая ОС, которая считает, что она запущена в режиме ядра, но это не так ("**виртуальный режимом ядра**").
  - в гостевой ОС запущены пользовательские процессы, которые считают, что они выполняются в пользовательском режиме, это действительно так.
- Проблема: что будет, если гостевая ОС выполнит инструкцию ядра?
  - Если процессор не имеет аппаратной поддержки виртуализации, произойдет ошибка и крах гостевой ОС.
  - Если процессор имеет аппаратную поддержу виртуализации, произойдет прерывание, передаваемое гипервизору.

Виртуализация ЦПУ (2)

- Аппаратная поддержка виртуализации: технологии Intel VT (2005), AMD-V (2006).
  - Теоретически скорость работы гостевых ОС сопоставима с прямой работой ОС на хосте.
  - Практически прерывания "тормозят" исполнение гостевых ОС, программная виртуализация может быть и не медленнее.
- Виртуализация существовала и раньше, при отсутствии аппаратной поддержки на архитектуре x86:
  - динамическая трансляция: "проблемные" команды кода гостевой ОС заменяются на вызовы процедур гипервизора;
  - исполнение гостевых ОС "в кольце 1" (гипервизоры 1 типа).

Виртуализация

- Общие понятия
- Виртуальные машины (системные)
- Виртуализация CPU
- ➔ Виртуализация памяти
- Другие аспекты виртуализации

Виртуализация памяти (1)

- Гостевая ОС думает, что работает с реальной таблицей страниц, но это не так. Как отобразить виртуальные физические страницы памяти на реальные физические страницы памяти?
  - Использование **теневой таблицы страниц** гипервизором. Когда вносить изменения?
    - при каждом изменении таблицы страниц гостевой ОС (трудоемко т.к. это просто запись в память): необходимо "найти" и перехватывать изменение соответствующей страницы памяти;
    - синхронизировать при возникновении ошибки отсутствия страницы памяти;
  - Аппаратная поддержка: вложенные таблицы страниц (nested page tables, AMD), расширенные таблицы страниц (Extended Page Tables, Intel).

→ для гипервизора 2 типа нужен модуль ядра, исполняющийся в режиме ядра.

Виртуализация памяти (2)

- Проблема перерасхода памяти, подкачки и вытеснения страниц:
  - подкачка гостевой ОС и подкачка хоста — разные алгоритмы и разные области диска: есть риск, что гостевая ОС запросит выгруженную хостом страницу только для того, чтобы тут же выгрузить ее своими средствами;
  - на хосте может быть запущено несколько гостевых ОС, суммарный объем виртуальной памяти которых превышает объем реальной памяти хоста: бОльшую часть времени гостевые ОС не будут использовать всю предоставленную им память (перерасход памяти).

Виртуализация памяти (3)

- Как гипервизору определить, какие страницы выгружать — нет информации о востребованности приложениями гостевой ОС, получается неэффективный алгоритм.
  - Выход — "раздувание": загрузка в гостевую ОС модуля ядра, который взаимодействует с гипервизором, если хосту не хватает памяти, то модуль требует от гостевой ОС выделения невыгружаемых страниц ("раздувается"), вынуждая гостевую ОС выгружать страницы приложений по собственному алгоритму, и, наоборот, модуль "сдувается", если памяти хосту достаточно;
- Технология дедупликации: определение идентичных страниц у разных гостевых машин (например, запуск одинаковых ОС на разных виртуальных машинах даст одинаковые страницы кода ядра и многих библиотек).

Виртуализация

- Общие понятия
- Виртуальные машины (системные)
- Виртуализация CPU
- Виртуализация памяти
- ➔ Другие аспекты виртуализации

Другие аспекты виртуализации

- Виртуализация ввода-вывода
  - эмуляция накопителей (разделы дисков хоста, файлы файловой системы ОС хоста);
  - перехват, передача и эмуляция аппаратных прерываний;
  - виртуализация и DMA.
- Эмуляция сетевых устройств
  - NAT, Bridge, виртуальный адаптер хоста.
- Общие папки.
- Создание "снимков", откат, миграция.
- Передача устройств на прямое управление гостевой ОС (USB, PCI — требуется аппаратная поддержка).

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- ➔ Безопасность и уязвимость ОС.
- Особенности реальных ОС: примеры и проблемы разработки.

Безопасность ОС

- Общие сведения
- Конструктивная безопасность
- Действия пользователей
- Программные источники уязвимостей
- Вредоносное ПО

Безопасность ОС

- ➔ Общие сведения
- Конструктивная безопасность
- Действия пользователей
- Программные источники уязвимостей
- Вредоносное ПО

Проблемы безопасности ОС (1)

- Эксплуатирование небезопасности:
  - атаки злоумышленников;
  - случайные действия пользователей;
  - работа спецслужб;
  - развлечение.
- Последствия:
  - получение конфиденциальных данных (личная информация, коммерческая тайна, данные доступа, государственная тайна);
  - нанесение ущерба (материального, морального, репутационного);
  - кража вычислительных ресурсов;
  - ...

Проблемы безопасности ОС (2)

- Источники проблем безопасности вычислительных систем:
  - конструктивные особенности;
  - ошибки и недоработки в коде (уязвимости);
  - действия пользователей.
- Важность учета безопасности в конструкции ОС:
  - выше, чем для обычных прикладных программ;
  - проблема может быть сопоставимой или даже острее
    - в веб-программировании;
    - в управляющих системах с многопользовательским и сетевым доступом.

Почему нет безопасной системы?

- Количество ошибок в программном коде прямо пропорционально размеру кода → безопасность системы обратно пропорциональная количеству реализованных возможностей.
- Безопасная система — меньше возможностей при той же цене, более дорогая при том же количестве возможностей.
- Безопасность требует дополнительных ресурсов (безопасный код работает медленнее).
- Практически невозможно обеспечить обратную совместимость напрямую.
- Проблема выхода на рынок: он уже заполонен дешевыми небезопасными системами, снабженными огромным количеством прикладных программ.
- Проблема уязвимости пользователя никуда не денется.
- Проблема "черной королевы" (гонка вооружений).

Безопасность ОС

- Общие сведения
- ➔ Конструктивная безопасность
- Действия пользователей
- Программные источники уязвимостей
- Вредоносное ПО

Конструктивная (не) безопасность

- Важность конструктивной безопасности ОС, программ и оборудования: по умолчанию имеются возможности
  - программного доступа к данным и оборудованию;
  - физического доступа к оборудованию;
  - перехвата передаваемых данных (сигналов в пространстве, пакетов в промежуточном узле).
- Дополнительные конструктивные проблемы
  - несанкционированное исполнение кода (автозапуск);
  - "черные ходы";
  - ...

Инструменты

- Использование пользовательского режима ЦПУ;
- Использование аутентификации и разграничения прав пользователей (многоразовый пароль, одноразовый пароль, физический ключ, биометрический доступ);
- шифрование данных (с закрытым ключом, с открытым ключом, одностороннее);
- разграничение прав программ (возможность исполнения, доступ к данным, доступ к оборудованию);
- публичное исследование кода (открытого) и "белый хакинг";
- физическое ограничение доступа;
- правовые меры.

Безопасность ОС

- Общие сведения
- Конструктивная безопасность
- ➔ Действия пользователей
- Программные источники уязвимостей
- Вредоносное ПО

Действия пользователей (1)

- Пароли:
  - слабые (угадываются, подбираются), борьба — установка требований к сложности (длина, алфавит, несловарность);
  - сильные (не запоминаются), борьба — ???;
  - долго не меняющиеся, борьба — установка лимита на время действия;
  - общие для разных ресурсов, борьба — просвещение пользователей;
  - ...
- Установка и запуск вредоносных программ, борьба — ограничение прав, просвещение, информирование, использование средств защиты.
  - осознанный (скачанное ПО)
  - полуосознанный (подмененное ПО, ловушки и др.);
  - неосознанный (запуск замаскирован под другое действие или запуск другой программы).

Действия пользователей (2)

- Атака на пользователя (социальная инженерия, фишинг и др).
- Проблема пользователей: низкая квалификация, неосведомленность об устройстве вычислительных систем, ПО и математических основах информатики.
- Отдельный вопрос: действия инсайдеров:
  - черные ходы (несанкционированные разработчиком);
  - часовые бомбы (срабатывает в нужный момент);
  - логические бомбы (срабатывает при выполнении определенных условий).

Безопасность ОС

- Общие сведения
- Конструктивная безопасность
- Действия пользователей
- Программные источники уязвимостей
- ➔ Вредоносное ПО

Источники уязвимостей (1)

- Переполнение буфера

```
char s[128];
gets (s);
```

- Использование форматирующей строки

```
char s[128];
fgets (s, 127, stdin);
printf (s); /* вместо printf ("%s", s) */
```

- Указатели на несуществующие объекты

```
type *p = malloc (N);
/* ... */
free (p);
/* ... */
*p = value;
```

- Разыменование нулевого указателя (проблема, если в пространстве ядра).

Источники уязвимостей (2)

- Переполнение целочисленных значений
- Внедрение команд

```
system ("ls >file_list");
```

- Атаки, проводимые с момента проверки до момента использования (проблема, если программа имеет возможность запросить административный доступ)

```
/* программа с setuid root */
int fd;
/* проверка права пользователя, запустившего
программу */
if (access ("./file", W OK) != 0) exit (1);
/* здесь вне программы: подмена ./file
 * на симв. ссылку на системный файл
 * далее: действия программы от root
 */
fd = open ("./file", O_WRONLY)
write (fd, data, sizeof (data));
```

Борьба с уязвимостями

- "Канарейки" (проверка целостности стека);
- предотвращение выполнения данных (защита страниц памяти и др.);
- рандомизация распределения адресного пространства;
- запрет на mmap_zero (и др. помещение данных по адресу NULL);
- поиск и недопущение ошибок в коде: отказ от небезопасных инструментов, статический анализ кода, верификация кода и др.;
- публичное исследование кода и "белый хакинг";
- обновления безопаснсти.

Безопасность ОС

- Общие сведения
- Конструктивная безопасность
- Действия пользователей
- Программные источники уязвимостей
- ➔ Вредоносное ПО

Вредоносное ПО (malware)

- Распространение:
  - прямой запуск (троянские кони);
  - использование уязвимостей (черви);
- Размещение вирусов
  - отдельная программа;
  - "инфицирование" исполняемых файлов;
  - оперативная память (резидентные);
  - загрузочные сектора;
  - драйверы;
  - макровирусы;
  - ...
- А также (терминология):
  - эксплоиты;
  - программы-шпионы;
  - руткиты (программы, скрывающие свое пристуствие)
    - во встроенном ПО, гипервизоры, в ядре, в библиотеках, в приложениях.

Типы вредоносного ПО (malware)

- Троянские кони (запускаются пользователем);
- Черви (распространяются с использованием уязвимостей);
- Эксплоиты (эксплуатируют уязвимости);
- Программы-шпионы (воруют данные: с диска, с клавиатуры, с камеры, с микрофона и т.д.);
- Программы-вымогатели (блокираторы ОС, шифровальщики и т.п.)
- Руткиты (программы, скрывающие свое присутствие или присутствие иного вредоносного ПО) — во встроенном ПО, в виде гипервизора, в ядре, в библиотеках, в компиляторах, в приложениях.

Средства защиты

- Межсетевые экраны (разграничение прав программ на действия в качестве клиента и сервера).
- Антивирусные программы
  - поиск известного вредоносного ПО;
  - проверка целостности файлов (контрольная сумма, "вакцина");
  - проверка поведения (эвристика).
- Средства восстановления целостности системы и программ (обнаружение и восстановление поврежденных файлов).
- Информирование и просвещение пользователей.

Общий план курса

- Введение: компьютеры и операционные системы.
- Процессы: выполнение, планирование, взаимодействие.
- Компьютерные сети: протоколы, адресация, маршрутизация.
- Память: адресация, управление, хранение данных и носители информации.
- Ввод и вывод: аппаратное и программное обеспечение.
- Человеко-компьютерное взаимодействие: устройства, языки, пользовательский интерфейс.
- Виртуализация: виртуальные машины и эмуляторы.
- Безопасность и уязвимость ОС.
- ➔ Особенности реальных ОС: примеры и проблемы разработки.

ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- Ядро ОС linux.
- ОС Windows.
- Разработка ОС
- Критика.

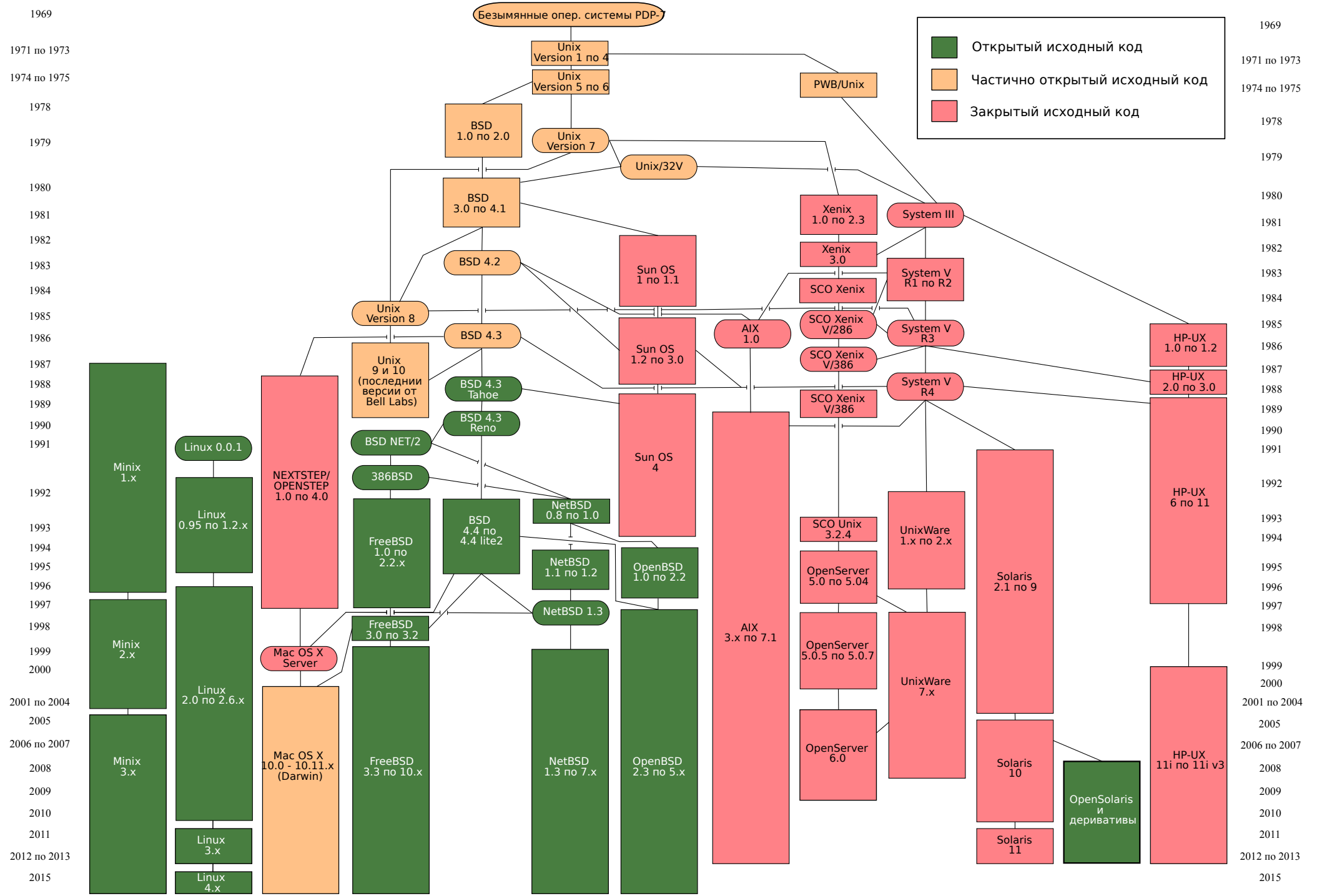
ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- Ядро ОС linux.
- ОС Windows.
- Разработка ОС
- Критика.

ОС UNIX и семейство *NIX

- Корни: операционная система **UNIX** (AT&T, 1970-е)
  - основные принципы (философия), язык программирования Си, оболочка, ряд стандартных команд и утилит оболочки, ...
  - проблемы: проприетарная, реализация на устаревшем оборудовании.
- "Альтернативная реализация UNIX" (с 1980-х)
  - BSD, GNU, Linux, Solaric, MacOS ...
  - проприетарные, свободные, частично свободные...
- Стандарт **POSIX** (Portable Operating System Interface, 1990-е)
  - стандартизация **системных вызовов** и **команд оболочки**;
  - реальные системы могут поддерживать POSIX частично и расширять его;
  - строгое следование POSIX — удобство переносимости программ, в т.ч. сценариев оболочки, привычность интерфейса (пользовательского и программного).

Генеалогия *nix



Философия UNIX (1)

- Существует реально ли или фольклор?
  - Появилась после UNIX?
  - Можно встретить различные формулировки, различные правила и принципы, разных авторов.
  - Некоторые принципы стали общими принципами программирования.
- Принцип "**Делайте что-то одно, но делайте это хорошо**".
  - простота, ясность, экономичность, модульность, взаимодействие программ и т.д.
- Принцип "**Все есть файл**"
  - устройства и другие ресурсы ОС и аппаратной среды отображаются в файловую систему.
- Еще о файлах:
  - **текстовые файлы** — универсальный формат;
  - программы должны работать как **фильтры**.
- **Принцип наименьшей неожиданности**.

Философия UNIX (2)

- Подход "для программистов" / "для администраторов":
  `$ grep text *.txt`
  вместо запуска программы поиска, ответа на запрос какие файлы искать, что в них искать и т.д.
- Небольшое количество малых программы, выполняющих малые задачи — возможность комбинировать их.
- Наименьшая неожиданность — в т.ч. для согласованности разных программ
  - Если команда
    `$ ls A*`
    выводит все файлы, чьи имена начинаются с A, команда то
    `$ rm A*`
    должна удалить именно эти файлы.
- UNIX не предназначен для ограждения своих пользователей от глупостей, поскольку это оградило бы их и от умных вещей.
  - **минимум вопросов**;
  - **правило тишины**.

ОС: решения и проблемы

- UNIX и семейство *nix.
- ➔ Стандарт POSIX.
- Ядро ОС linux.
- ОС Windows.
- Разработка ОС.
- Критика.

Стандарт POSIX

- Включает в себя
  - Основные определения и обоснование принципов стандарта.
  - Системные интерфейсы: список системных вызовов, список заголовочных файлов языка Си и системных библиотечных функций;
  - Оболочка и утилиты: описание утилит и командной оболочки sh, стандарты регулярных выражений.
- Поддержка POSIX в ОС
  - POSIX-сертифицированные: MacOS, Solaris, ...
  - POSIX-совместимые: FreeBSD, MINIX, Android, Darwin, OpenSolaris, ...
  - POSIX для Windows: Microsoft POSIX subsystem, Cygwin, MinGW, ...
  - POSIX для OS/2, DOS и др.

Некоторые системные вызовы POSIX

Вызов	Описание
open	открытие файла
read	чтение из файла
write	запись в файл
creat	создание файла
close	закрытие файла
chmod	изменение прав доступа к файлу
chown	изменение владельца файла
opendir	открытие каталога
readdir	чтение каталога
mkdir	создание каталога
rmdir	удаление каталога
fork	создание процесса
exec	замена образа процесса
exit	завершение процесса
wait	ожидание процесса
sigaction	определение действий при получении сигнала
kill	отправка сигнала процессам

Оболочка POSIX

- Запуск внутренних и внешних команд
  - регистрозависимые имена файлов и команд, предпочтение нижнему регистру;
  - разделяют параметры команды (имена файлов и т.п., указываемые просто через пробел) и опции (ключи), обычно начинающиеся с `-` (однобуквенные, `-r`, `-w` и т.п.) или с `--` (многобуквенные, --version), со значениями или без;
  - кавычки и апострофы подавляют пробелы как разделители параметров, апострофы подавляют раскрытие переменных, обратный слеш подавляет специальные символы;
  - всякой программе назначается стандартный поток ввода, вывода и сообщений об ошибках.
- Средства программирования сценариев:
  - использование переменных окружения в качестве переменных;
  - проверка условий, циклы по условию, по набору значений;
  - широкий набор команд по обработке строк символов, текстовых файлов.
- Использование групповых символов (wildcard) для выбора файлов (передача списков файлов и каталогов в команду):
  - `*` — любая последовательность символов,
  - `?` — один непустой символ;
  - `[...]` — один из символов списка.
- Широкое использование фильтров и конвееров
  - `$ grep foo *.txt | sort | head -20 | tail -5 >bar`
  - `$ rm `grep foo *.txt``

Некоторые команды оболочки POSIX

Команда	Описание
echo	вывод сообщения
cat	конкатенация файлов
head	извлечение первых строк файла
tail	извлечение последних строк файла
cp	копирование файла
mv	перенос (переименование) файла или каталога
rm	удаление файла
mkdir	создание каталога
rmdir	удаление каталога
chmod	изменение прав доступа
chown	изменение владельца файла
cd	смена текущего каталога
grep	поиск регулярных выражений в файлах
sort	сортировка строк в файлах
find	поиск фалов
ls	вывод списка файлов каталога (каталогов)
ps	вывод списка процессов
kill	отправка сигнала процессам
man	просмотр справки по команде

Пользователи и защита

- Каждый пользователь имеет уникальный номер (**uid**) и номер группы пользователей (**gid**).
  - Идентификация пользователей проводится по логину и паролю.
  - Пользователь 0 является привилегированным (логин "**root**", группа "root"), имеет полный доступ к системе.
  - Для всякого процесса установлены uid и gid пользователя, от имени которого процесс выполняется, что и определяет права доступа для данного процесса.
- Всякий файл снабжается атрибутами доступа
  - у файла есть владелец (uid) и группа владельца (gid);
  - модель **rwx**: Read, Write, eXecute (исполнение для файлов, переход для каталогов);
  - разделяют права владельца, права группы, права всех остальных;
  - индивидуальные права доступа — POSIX ACL (тоже rwx);
  - бит setuid и setgid — право на запуск от имени владельца и группы исполняемого файла, а не от имени запускающего процесс пользователя.

ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- ➔ Ядро ОС linux.
- ОС Windows.
- Разработка ОС.
- Критика.

Общая структура linux

- Уровни организации:
  - аппаратное обеспечение
  - операционная система linux
  - стандартная библиотека
  - стандартные обслуживающие программы
  - пользователи
- Загрузка linux
  - Загрузчик компьютера (встроенное ПО — BIOS, EFI)
  - Загрузчик ОС (MBR, GPT linux не запускают, запускают GRUB, Lilo, syslinux и др.)
  - Запуск ядра
    - инициализация начального RAM диска (initrd/initramfs) — базовая проверка целостности, оболочка с возможностью восстановления системы;
    - монтирование корневой файловой системы, переход в корневую файловую систему;
    - запуск процесса инициализации системы (init, systemd) — запуск демонов (служб) и выполнение сценариев загрузки.

ОС, основанные на Linux

- Linux (Л. Торвальдс, с 1991) — только ядро ОС (управление ресурсами, драйвера): монолитно-модульное ядро.
- Операционная система GNU (Р. Столлман, с 1983) — GNU’s Not UNIX, ядро GNU Hurd (микроядро) не написано.
  - Коллекция GNU-утилит (расширение POSIX, оболочки);
  - Коллекция GNU-компиляторов (Си и другие языки, расширение языков);
  - Коллекция программ;
  - Принцип свободного ПО.
- Дистрибутивы linux: ОС GNU/linux
  - ядро linux;
  - приложения GNU;
  - графический интерфейс: система X Windows;
  - графические окружения рабочего стола (от GNU: Gnome);
  - приложения (пользовательские, серверные, системные и др.).
- Android: ядро linux, но не GNU.
- FreeBSD, MacOS, Solaris и др *nix системы: не linux, но могут содержать части GNU или собственную реализацию POSIX-приложений.

Файловая система

- Подключение ФС носителей:
  - корневой каталог /
  - монтирование файловых систем в любой (пустой?) каталог.
- Не все, но многое есть файл:
  - устройства (/dev/device)
  - элементы системы (/sys/kernel/irq/0/actions)
  - процессы (/proc/1/maps)
  - средства защиты (/sys/fs/selinux/enforce)
  - ...
- Условно-фиксированная структура файловой системы
  - /bin основные исполняемые файлы
  - /boot загрузчик ОС
  - /dev устройства
  - /etc файлы настроек
  - /home домашние каталоги пользователей
  - /lib библиотека
  - /mnt прочие подмонтированные файловые системы
  - /opt программы сторонних производителей
  - /root домашний каталог пользователя root
  - /sbin системные программы
  - /tmp временные файлы
  - /usr пользовательские программы
  - /var прочие файлы

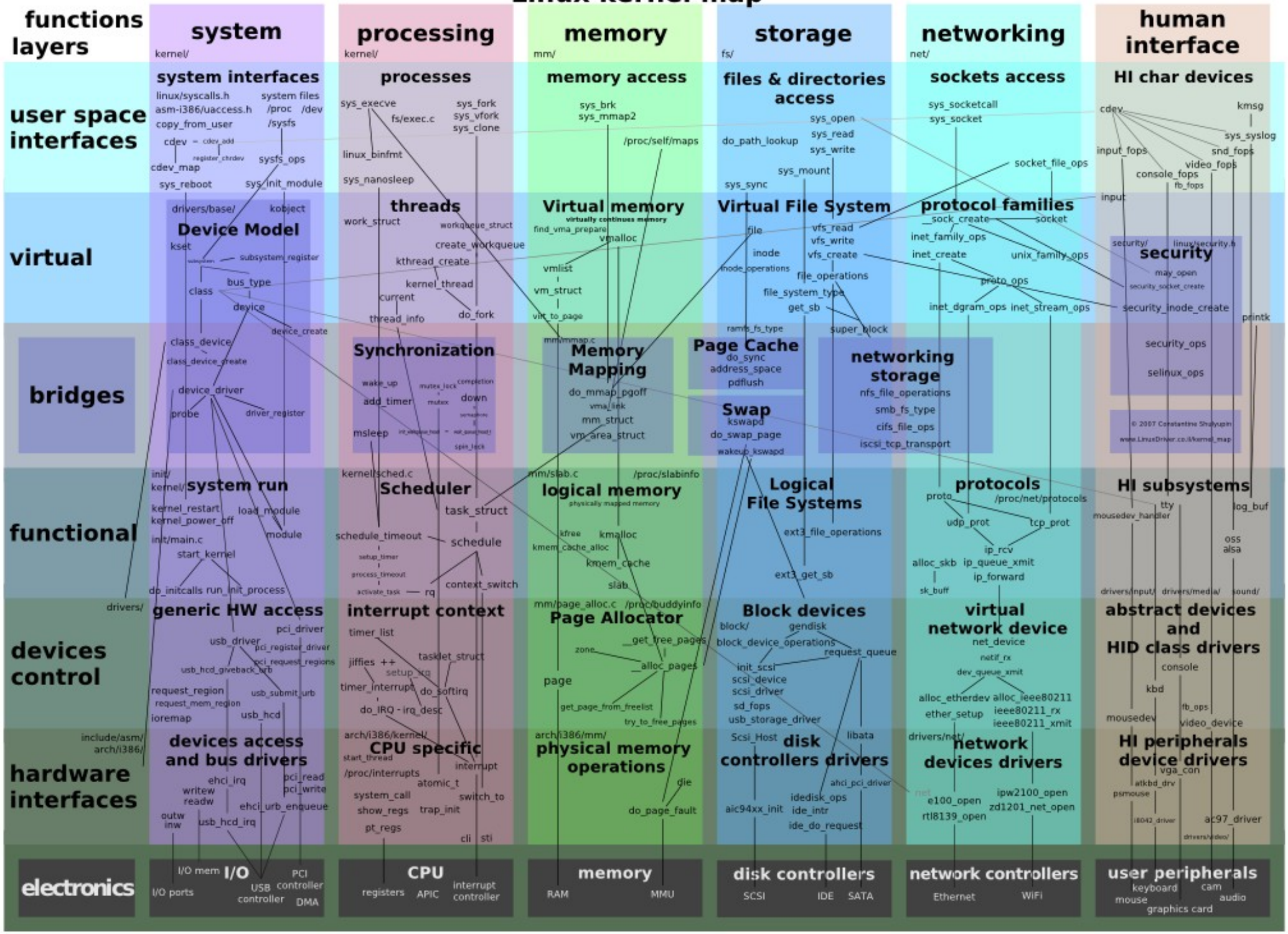
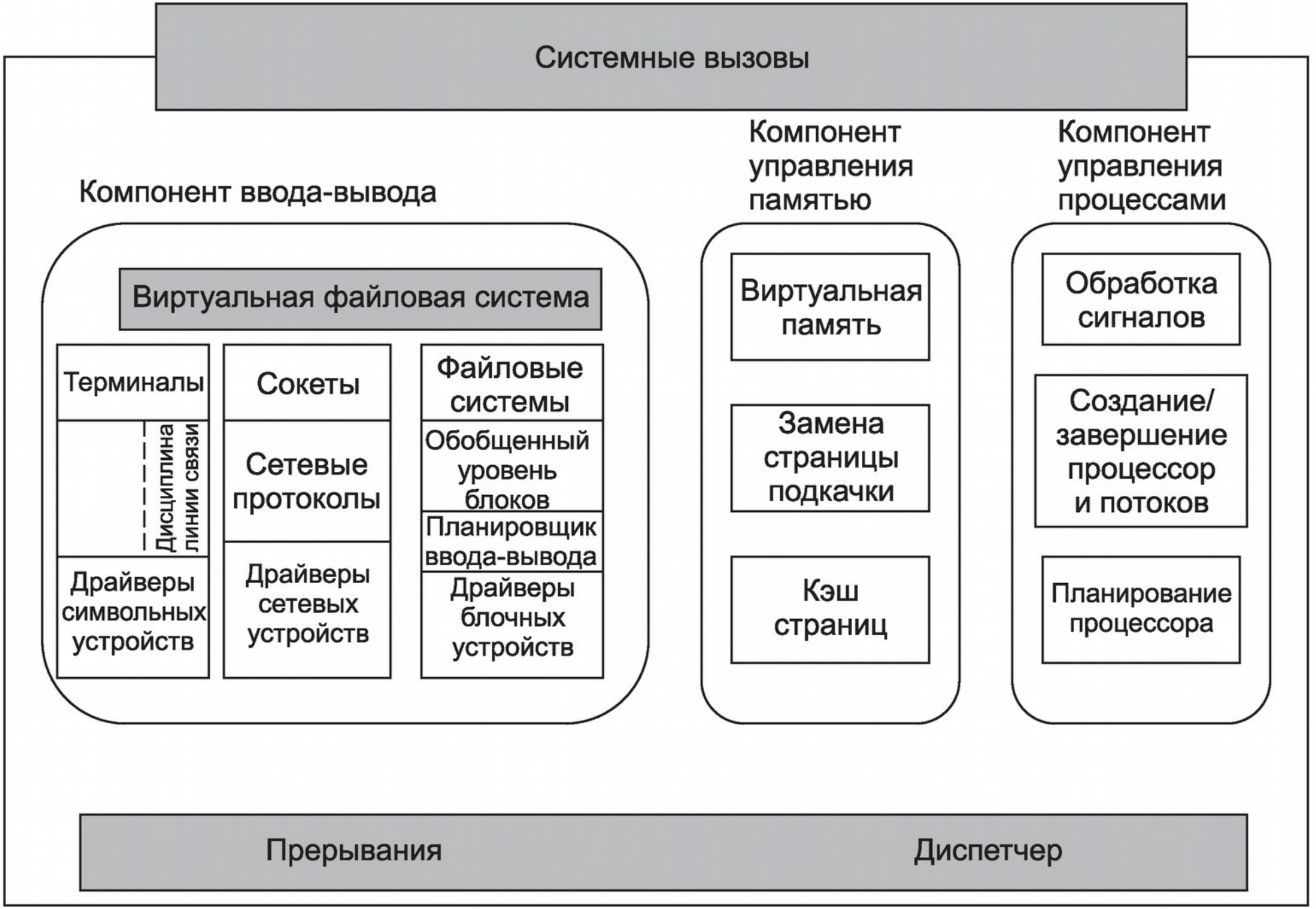

Linux kernel map
functions
layers
system
kernel/
processing
kernel/
memory
mm/
storage
fs/
networking
net/
human
interface
user space
interfaces
system interfaces
linux/syscalls.h
asm-i386/uaccess.h
copy_from_user
system files
/proc
/dev
/sysfs
cdev
cdev_add
register_chrdev
cdev_map
sysfs_ops
sys_reboot
sys_init_module
processes
sys_execve
fs/exec.c
sys_fork
sys_vfork
sys_clone
linux_binfmt
sys_nanosleep
memory access
sys_brk
sys_mmap2
/proc/self/maps
files & directories
access
sys_open
sys_read
sys_write
do_path_lookup
sys_mount
sys_sync
sockets access
sys_socketcall
sys_socket
socket_file_ops
HI char devices
cdev
kmsg
sys_syslog
input_fops
snd_fops
video_fops
console_fops
fb_fops
input
virtual
drivers/base/
kobject
Device Model
kset
subsystem_register
bus_type
class
device
device_create
threads
work_struct
workqueue_struct
create_workqueue
kthread_create
kernel_thread
current
thread_info
do_fork
Virtual memory
virtually continues memory
find_vma_prepare
vmalloc
vmlist
vm_struct
virt_to_page
mm/mmap.c
Virtual File System
file
inode
inode_operations
vfs_read
vfs_write
vfs_create
file_operations
file_system_type
get_sb
ramfs_fs_type
super_block
protocol families
_sock_create
socket
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops
security
security_ops
selinux_ops
bridges
class_device
class_device_create
device_driver
probe
driver_register
Synchronization
wake_up
add_timer
msleep
mutex_lock
completion
mutex
down
spin_lock
Memory
Mapping
do_mmap_pgoff
vma_link
mm_struct
vm_area_struct
Page Cache
do_sync
address_space
pdflush
Swap
kswapd
do_swap_page
wakeup_kswapd
networking
storage
nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport
security_inode_create
printk
functional
init/
kernel/
system run
kernel_restart
kernel_power_off
load_module
module
init/main.c
start_kernel
do_initcalls
run_init_process
kernel/sched.c
Scheduler
task_struct
schedule_timeout
schedule
setup_timer
process_timeout
activate_task
rq
context_switch
mm/slab.c
/proc/slabinfo
logical memory
physically mapped memory
kfree
kmalloc
kmem_cache_alloc
kmem_cache
slab
Logical
File Systems
ext3_file_operations
ext3_get_sb
protocols
proto
/proc/net/protocols
udp_prot
tcp_prot
ip_rcv
alloc_skb
ip_queue_xmit
ip_forward
sk_buff
HI subsystems
tty
mousedev_handler
log_buf
oss
alsa
drivers/input/
drivers/media/
sound/
devices
control
drivers/
generic HW access
pci_driver
usb_driver
pci_register_driver
pci_request_regions
usb_hcd_giveback_urb
request_region
request_mem_region
ioremap
usb_submit_urb
usb_hcd
interrupt context
timer_list
jiffies ++
tasklet_struct
setup_irq
timer_interrupt
do_softirq
do_IRQ
irq_desc
mm/page_alloc.c
/proc/buddyinfo
Page Allocator
zone
__get_free_pages
__alloc_pages
page
get_page_from_freelist
try_to_free_pages
Block devices
block/
gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
virtual
network device
net_device
netif_rx
dev_queue_xmit
alloc_etherdev
alloc_ieee80211
ether_setup
ieee80211_rx
ieee80211_xmit
abstract devices
and
HID class drivers
console
kbd
mousedev
fb_ops
video_device
hardware
interfaces
include/asm/
arch/i386/
devices access
and bus drivers
ehci_irq
writew
readw
ehci_urb_enqueue
pci_read
pci_write
outw
inw
usb_hcd_irq
arch/i386/kernel/
CPU specific
start_thread
/proc/interrupts
interrupt
atomic_t
system_call
switch_to
show_regs
trap_init
pt_regs
cli
sti
arch/i386/mm/
physical memory
operations
die
do_page_fault
Scsi_Host
disk
controllers drivers
libata
ahci_pci_driver
idedisk_ops
ide_intr
aic94xx_init
ide_do_request
drivers/net/
network
devices drivers
net
e100_open
rtl8139_open
ipw2100_open
zd1201_net_open
HI peripherals
device drivers
vga_con
atkbd_drv
psmouse
i8042_driver
ac97_driver
drivers/video/
electronics
I/O mem
I/O
PCI
I/O ports
USB
controller
controller
DMA
CPU
registers
APIC
interrupt
controller
memory
RAM
MMU
disk controllers
SCSI
IDE
SATA
network controllers
Ethernet
WiFi
user peripherals
keyboard
cam
mouse
audio
graphics card

Схема ядра linux



Процессы в linux

- Каждый процесс снабжается уникальным идентификатором (**pid**), с которым ассоциированы данные процесса.
- Процессы работают в пространстве пользователя
  - при системном вызове процесс переводится в пространство ядра со своим собственным стеком пространства ядра.
- Алгоритмы планирования процессов.
  - Три класса приоритета:
    - реального времени, обслуживаемые по алгоритму **FIFO** (непрерываемые);
    - реального времени, обслуживаемые **циклически**;
    - разделения времени (различный приоритет, число **nice** от -20 до +19).
  - Алгоритм планирования **Linux O(1)**:
    - две очереди выполнения: активных задач и задач, чье время истекло;
    - если процесс блокируется до истечения кванта времени, он остается активным, если квант времени истек переносится во второй массив;
    - если активные задачи закончились, массивы меняются местами;
    - эвристика интерактивности: блокировка ожидания клавиатурного ввода vs блокировка дисковой операции.
  - Абсолютно справедливый планировщик (**CFS**)
    - очередь выполнения — красно-черное дерево, ключ — виртуальное время выполнения;
    - для более приоритетных задач время течет медленнее;

Управление памятью в linux

- Всякий процесс имеет
  - **текстовый сегмент** (сегмент кода);
  - **сегмент данных** (две части: инициализированные и неинициализированные данные);
  - **сегмент стека**.
- Экономия памяти
  - текстовые сегменты совместного использования (запуск одной и той же программы дважды);
  - **копирование при записи** (дублирующиеся страницы при создании процесса копируются только при изменении);
  - **отложенное выделение памяти** (выделение физической памяти при обращении);
  - **OOM** (Out Of Memory) **Killer** (уничтожение вредных процессов при нехватке памяти).
- Подкачка (выгрузка страниц)
  - в отдельный раздел (**swap**);
  - в файл подкачки (используется реже);
  - раздел подкачки используется для гибернизации.
- Алгоритм замещения страниц
  - основан на алгоритме "часы" с информацией о недавно использующихся страницах (без учета рабочего набора);
  - сейчас используются комбинации различных подходов.

Ввод и вывод в linux

- Устройства — специальные файлы.
- При работе с символьными устройствами может использоваться дисциплина линии связи (клавиатура: ввод строками, а не символами).
- Файловая система:
  - дисковые файловые системы: ext4, btrfs, ...
  - другие файловые системы: поддерживаются почти все современные файловые системы на уровне ядра или модулей сторонних файловых производителей, разработано множество недисковых файловых систем.
  - **виртуальная файловая система**: унификация файловых операций (устройство-независимое ПО ввода-вывода).
- Планировщик ввода-вывода: оптимизация работы устройств.

Драйвера устройств в linux

- Могут подключаться как
  - часть ядра;
  - модуль ядра (модули могут загружаться и отключаться при работе).
- Связаны с ядром через особый интерфейс (интерфейс ядро-драйвер)
  - интерфейс зависит не только от версии ядра, но и от конфигурации сборки (под каждое ядро модули собираются индивидуально).

Еще о реализации linux

- Демон — фоновая программа, работает без взаимодействия с пользователем.
- D-Bus — система межпроцессного взаимодействия над POSIX (основана на сообщениях).
- HAL (Hardware Abstraction Layer) — слой аппаратных абстракций для Linux, заменен на udev.
- udev — современный менеджер устройств.
- dconf — система межпрограммной конфигурации (нетекстовая).
- journald — система протоколирования (нетекстовая).

Использование linux

- Суперкомпьютеры и некоторые мейнфреймы.
- Серверы.
- Нетбуки и лэптопы (поставляется производителем).
- Встроенные системы.
- Смартфоны и коммуникаторы.
- Настольные компьютеры (использование ограничено?).

Безопасность в linux

- Пароли пользователей хранятся с использованием одностороннего шифрования и "соли".
- Программа **sudo** используется для предоставления пользователям административного доступа без сообщения пароля root.
- Мандатное управление доступом (**MAC**): контроль, основанный на метках, ролях пользователей и программ.
  - SElinux, TOMOYO, AppArmor, TrustedBSD, grsecurity и др.
- Межсетевой экран: **iptables**.
- Средства борьбы с вредоносным ПО: сторонние, в основном сканирование по запросу.
- Защищенность от вирусов — ?
  - Система конструктивно более защищена (чем DOS/Windows):
    - сложнее исполнить код, сложнее получить административный доступ и др.
  - Открытый код и быстрое обнаружение уязвимостей сообществом?
  - Разнообразие дистрибутивов при небольшом количестве пользователей
    - "невогодная мишень", цель — сервера (проблема актуальна).
  - Средний пользователь linux более квалифицирован.

Особенности Android

- Отсутствие подкачки: агрессивный OOM Killer (для подкачки может использоваться zram).
- Большинство пользовательских программ работает на JAVA.
- Dalvik — реализация среды JAVA (особый исполняемый код).
- zygote — предзапущенный процесс программы JAVA (fork и копирование образа быстрее инициализации JAVA).
- Binder — система межпроцессного взаимодействия.
- Безопасность: каждая программа работает как отдельный POSIX-пользователь (со своим uid и gid).
- Отсутствие доступа от имени root во многих устройствах.
- Проблема обновления: сочетание свободного ядра и проприетарных драйверов, предоставляемых производителем устройства.
- Уязвимость для вирусов — ?

ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- Ядро ОС linux.
- ➔ ОС Windows.
- Разработка ОС.
- Критика.

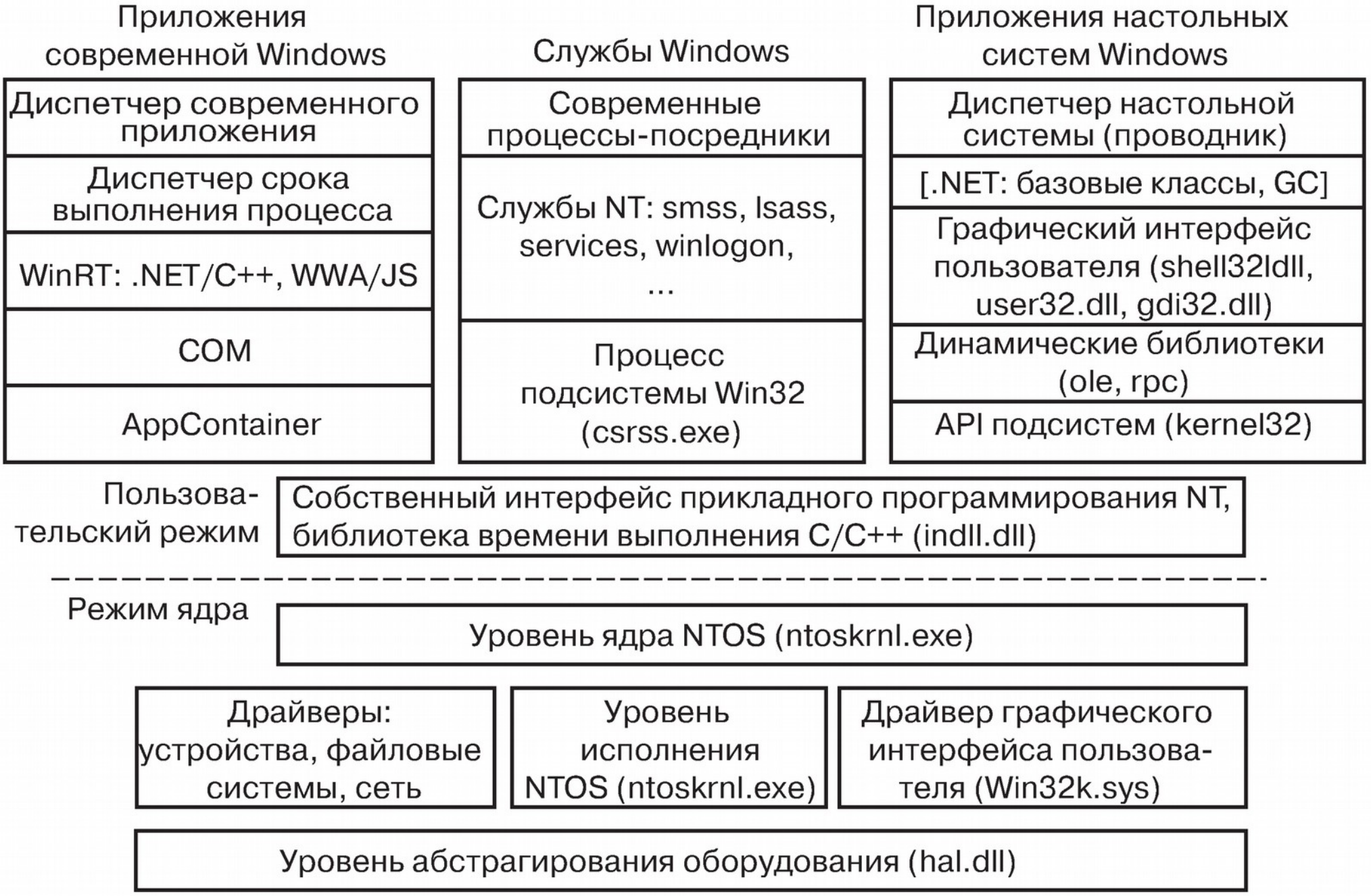
ОС Windows

- История
  - CP/M (1974-1983)
  - MS-DOS (1981-1993)
  - Windows-оболочка для DOS (1985-1993)
  - OS/2 (1987-2001)
  - Windows над DOS (1995-2000)
  - Windows на базе NT (с 1993)
- Виды
  - Для персональных компьютеров (Windows)
  - Серверные (Windows Server)
  - Встроенные (Windows Embedded)
  - Для мобильных устройств (Windows Phone/Mobile)
- Ключевые моменты
  - изначально ориентирована на графический интерфейс, "дружественный пользователю";
  - различные версии и типы версий оперируют различными технологиями, архитектурой, интерфейсом и др. элементами;
  - является стандартом де-факто для ПК.

Приложения Windows

- Системные вызовы:
  - уровня ядра (различны для разных семейств);
  - уровня пользователя (WinAPI).
- Подсистемы:
  - WinAPI (Win16, Win32, Win64, WinCE)
  - POSIX subsystem for Windows
  - OS/2 subsystem for Windows
  - UNIX/Linux subsystem for Windows
- Поддержка
  - Windows On Windows: поддержка Win16 на Win32 (WoW), Win32 на Win64 (WoW64);
  - VDM: поддержка приложений DOS.
- Windows Runtime (Windows RT, приложения в стиле "метро", "современные" приложения).

Уровни программирования Windows



Структура Windows

Особенности Windows (1)

- Системные вызовы режима ядра и режима пользователя.

Вызов WinAPI	Вызов ядра NT
CreateProcess	NtCreateProcess
ReadFile	NtReadFile
VirtualAlloc	NtAllocateVrtualMemory

- Реестр Windows
  - является общей системой конфигурации для ОС и приложений (заменил разрозненные конфигурационные .ini файлы);
  - содержит системную (в т.ч. оборудование, безопасность, аккаунты, безопасность) и пользовательские конфигурации (настройки интерфейса, приложений и д.р.);
  - записывается в нескольких файлах в системной области и в профайлах пользователей;
  - Представляет собой дерево разделов, содержащих ключи вида имя (текстовое) = значение (различных типов); создание, получение и изменение разделов доступно посредством вызовов WinAPI.
- Ядро Windows NT считается гибридным.

Особенности Windows (2)

- Динамически связываемые библиотеки (DLL)
  - аналог библиотек совместного использования *nix (.so).
- Загрузка Windows
  - встроенное ПО
  - MBR/GPT/UEFI
  - BootMgr
  - WinResume (восстановление из гибернизации) или WinLoad (загрузка)
- Объекты — унифицированный интерфейс для управления ресурсами системы, управляются диспетчером объектов.
  - Являются структурами данных ядра, доступны из пользовательского режима посредством дескрипторов (handler).
  - Для каждого объекта доступны стандартные (обязательные, Open, Parse, Delete, Close, Security, QueryName) и собственные методы.
  - Имеется отдельное пространство имен объектов с иерархической структурой.

Объекты Windows

- Некоторые элементы пространства имен объектов

Каталог	Содержимое
\?? (\DosDevices)	Имена дисков MS-DOS
\Device	Устройства ввода-вывода
\Windows	Отправка сообщений окнам GUI
\BaseNamedObjects	Создаваемые пользователем объекты
\FileSystem	Объекты драйверов файловых систем
\KnownDLLs	Известные библиотеки

- Некоторые типы объектов

Тип	Описание
Process	Пользовательский процесс
Semaphore	Семафор
Mutex	Мьютекс
Event	Событие (объект сигнализации)
Open file	Открытый файл
Key	Ключ реестра
Device	Устройство

Процессы и потоки в Windows

- Процесс
  - виртуальное адресное пространство;
  - дескрипторы объектов ядра;
  - потоки;
  - системные данные пользовательского режима (PEB, Process Environment Block): список загруженных модулей, переменные окружения, рабочий каталог и др.
- Поток
  - приоритет;
  - аффинизация (распределение по процессорам);
  - два стека вызовов (пользовательский, ядра);
  - блок среды потока (TEB, Thread Environment Block): данные пользовательского режима, области хранения для потока (Thread Local Storage) и поля для WinAPI, локализации языка и культуры и др.
- Задания
  - группы процессов с общим управлением и квотированием ресурсами;
- Волокна
  - части потоков, связь 1 ко многим.

Планирование потоков

- Инициируется самим потоком:
  - блокировка на мьютексе/семафоре и т.п.;
  - подача сигнала на объекте;
  - исчерпание кванта времени.
- Классы приоритетов
  - реального времени; высокий; выше среднего; средний; ниже среднего; фоновый.
- Класс приоритета потока может быть установлен ниже класса приоритета процесса.
- Планировщик подбирает "идеальный" процессор для потока, может повышать приоритет (после ожидания ввода-вывода, блокировки на семафоре и др.).

Замещение страниц памяти

- Использование файла подкачки для выгрузки страниц.
- Принцип рабочего набора
  - динамическая модификация размера набора при дефиците и избытке памяти.

Ввод и вывод

- Диспетчер ввода-вывода.
  - Plug-and-Play: запрос самоидентификации всех устройств шины.
  - Назначает аппаратные ресурсы (прерывания и др.)
  - Загружает драйвер (создается объект драйвера и как минимум один объект устройства).
  - Для некоторых шин (PCI) поиск устройств возможен только во время загрузки, для некоторых (USB) может произойти в любой момент работы.
- Драйвера
  - концептуально должна поставляться производителем оборудования;
  - являются частыми причинами сбоев на уровне ядра (BSOD);
  - предоставляются различные инструменты проверки и верификации драйверов.

Файловая система

- Поддерживаются ФС FAT и NTFS (а также ISO 9660, UDF и др).
- FAT сейчас используется в основном для сменных носителей
- NTFS
  - имеет поддержу Unicode, регистрозависимые имена файлов (не поддерживается WinAPI), символические и жесткие ссылки, многопоточные файлы;
  - имеет дескрипторы безопасности файлов
    - индивидуальные права для каждого пользователя и группы;
    - права могут наследоваться от родительского каталога;
    - есть права разрешающие и запрещающие, запрет имеет более высокий приоритет;
  - поддерживает прозрачное сжатие и шифрование отдельных файлов.
- Некоторые элементы файловой системы ОС Windows фиксированы (хотя меняются от версии к версии).

Безопасность

- Разделение пользователей и администраторов (частичное?)
- Наличие защиты от исполнения данных, использование различных технологий защиты от атак при компиляции и исполнении (перемешивание адресного пространства, неисполняемые стеки и др.)
- Встроенные средства защиты: межсетевой экран, защитник (антивирус).
- В современных Windows система безопасности основана на списках управления доступом и уровнях целостности.
  - Каждый процесс имеет маркер аутентификации (пользователь, привилегии).
  - Каждый объект имеет связанный с ним дескриптор безопасности (список управления доступом, разрешающий или запрещающий доступ отдельным пользователям или группам).
- Уязвимость для вирусов - ?

ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- Ядро ОС linux.
- ОС Windows.
- ➔ Разработка ОС
- Критика.

Проблемы разработки ОС

- Проблемы разработки ОС, особенно на фоне простых программ:
  - **огромный объем кода** (15 млн. строк linux, 50-100 млн в Windows): невозможно понять одному человеку, но отдельные части ОС сильно взаимосвязаны ("мифический человекогод");
  - **длительный жизненный цикл** ОС: обратная совместимость плюс незнание, где будет применяться ОС в начале разработки;
  - необходимость учета параллелизма, одновременного многопользовательского доступа, набора одновременно работающих устройств, переносимость;
  - наличие потенциально враждебных пользователей.
- Основные моменты при разработке ОС: **цель** и **парадигма**.

Задача разработки ОС

- Основные подзадачи:
  - Определение абстракций.
  - Предоставление примитивных операций.
  - Обеспечение изоляции.
  - Управление аппаратурой.
- Разработка интерфейса: руководящие принципы.
  - Простота (KISS): лучше меньше, чем больше.
  - Полнота: все, что требуется, должно быть выполнимым.
  - Эффективность: интуитивное понимание стоимости системных вызовов программистом.

  Пример: минимализм системных вызовов Minix (всего три системных вызова: send, receive, sendrec), Amoeba (единственный системный вызов: удаленный вызов процедуры), а POSIX-вызовы — библиотечные.
- Интерфейс: системных вызовов, пользовательский, драйверов.

Понимание стоимости вызовов

C	C++	Сложность

`int i, j, k;`
`/* … */`
`k = i + j;`
	`int i, j, k;`
	`/* … */`
	`k = i + j;`
		Θ(1)

`char s1[N], s2[N],`
`buffer[N];`
`/* … */`
`strcpy (buffer, s1);`
`strcat (buffer, s2);`
	`string s1, s2, buffer;`
	`/* … */`
	
	`buffer = s1 + s2;`
		Θ(n)

- Для динамических массивов и иных структур (**C++**): непредсказуемость момента переноса данных в памяти.
- **ООП**: накладные расходы на таблицу виртуальных методов, избыточность кода.
- **JAVA/C#**: еще бОльшая непредсказуемость (когда сработает сборщик мусора?)
- **C**: наличие структур (копирование при передаче в функцию и возврате).
- Безопасность: выше в языках более высокого уровня (?).

Парадигмы ОС

- **Парадигма пользовательского интерфейса.**
  - Что первично: пользовательский или программный интерфейс?
  - GUI: WIMP, сенсорный экран, естественно-интуитивный, специализированный.
- **Парадигма исполнения:**
  - алгоритмическая;
  - событийная.
- **Парадигма данных** (плюс унификация устройств и ресурсов?)
  - магнитная лента;
  - файлы;
  - объекты;
  - документы (URI).

Еще вопросы организации ОС

- Структура ОС:
  - Уровни организации? Микроядра?
  - Структурная согласованность?
- Механизм и политика: необходимо разделение.
  - Изменение политики не должно затрагивать механизм.
- Ортогональность: возможность комбинировать независимые концепции.
- Сокрытие аппаратуры.
- Проверка на ошибки.
- Производительность.
  - Plug'n'play: долгая загрузка.
  - Слишком много функций (т.е. кода)?
  - Что оптимизировать (профайлинг, память vs время)?

Некоторые проблемы существующих ОС (и не только ОС)

- Недоработанность, большое количество ошибок и уязвимостей, необходимость частых обновлений.
- Низкая производительность, относительное падение производительности: кажущаяся скорость работы компьютеров почти не растет, но "качество и количество" возможностей растет явно медленнее, чем по закону Мура.
- Недостаточная обратная совместимость
  - проблема поддержки старых программ;
  - проблема поддержки старого оборудования.

Проблема создания новой ОС

- Высокая стоимость и значительное время разработки.
- Как и с чем выйти на уже занятый и развитый рынок?
- Совместимость и поддержка оборудования.
- Привычки пользователей.

Проблема создания новой ОС

- Источники проблем:
  - высокая стоимость и значительное время разработки;
  - как и с чем выйти на уже занятый и развитый рынок?
  - совместимость и поддержка оборудования;
  - привычки пользователей.
- Создавались ли принципиально новые ОС?
  - Практически все ОС базируются на предшественниках (кони в 60-70 годах):
    - заимствование идей и принципов;
    - заимствование функционала (системные вызовы, команды оболочки);
    - заимствование кода (открытого).
  - Новые идеи, принципы и парадигмы регулярно добавлялись, в т.ч. с развитием оборудования (накопители, экраны).
  - Потенциально значительная часть кода современных ОС может оказаться избыточной (дублируемой, наслоенной, неиспользуемой, устаревшей), особенно если отказаться от обратной совместимости: сложность задачи создания ОС переоценена?
    - Проблема выхода на рынок остается.

ОС: решения и проблемы

- UNIX и семейство *nix.
- Стандарт POSIX.
- Ядро ОС linux.
- ОС Windows.
- Разработка ОС

➔ Критика.

Некоторые проблемы существующих ОС (и не только ОС)

- Недоработанность, большое количество ошибок и уязвимостей, необходимость частых обновлений, низкая или сложная настраиваемость, плохая документированность, принудительные обновления (недоступность старых версий) в т.ч. привычного интерфейса и др.
- Низкая производительность, относительное падение производительности: кажущаяся скорость работы компьютеров почти не растет, но "качество и количество" возможностей растет явно медленнее, чем по закону Мура.
- Недостаточная обратная совместимость
  - проблема поддержки старых форматов;
  - проблема поддержки старых программ;
  - проблема поддержки старого оборудования.

"Глас народа" (1)

- opennet.ru, Аноним, 29-июн-2014:

  26 лет назад у меня была проблема - текстовый процессинг немного тормозил. Казалось, добавить немножко мегагерц, несколько сотен килобайт памяти, и станет хорошо, и наконец-то можно будет нормально работать.

  Но нет, текстовые редакторы стали графическими, мегагерцы не помогли. Чёртова скрепка смотрела на меня в ворде и тормозила. Всё жрало память. Казалось бы, добавить немножко гигагерц и мегабайт памяти, и станет хорошо, и наконец-то можно будет нормально работать.

  Но нет, теперь документооборот только в сети. Открывай в браузере google.docs, фигач там. Всё написано на JS, всё в браузере, и поэтому тормозит. Но это потому, что вкладка делит ядро с другими задачами (и памяти маловато). Казалось бы, надо добавить немножко 3-гигагерцевых ядер, ещё несколько гигов памяти, и станет хорошо, и наконец-то можно будет нормально работать.

  Но нет, теперь ОС будут писать на яваскрипте тоже. И драйвера. Присвятой Торвальдс, они пишут драйвера на JS, этот мир сошел с ума. Хотя понятно - через 20 лет Интелу надо будет как-то продавать новые процессоры на 1024 ядра (ведь на 512 ядрах текст будет нормально не поредактировать - они придумают что-нибудь и для этого), надо начинать решать проблему заранее.

  А мне надо было ещё на первом шаге (26 лет назад) освоить vim и послать всех к чёрту.

Действительно...

- **Word 6.0 для Windows 3.1: 100 МГц процессор, 4 Мб памяти, 128 Мб жесткий диск**

  vs

  современные версии Word: 3 ГГц процесор, 8 Гб памяти, 1 Тб жесткий диск

→ производительность примерно одинакова;

→ какие возможности добавились? шрифты? полноцветная графика (нужная ли в каждом документе?) интерфейс? анимация в тексте?

→ отсутствие обратной совместимости форматов, частичная прямая совместимость форматов.

- **LaTeX: EmTeX 3.14159 для DOS vs совр. версии TeXLive**

→ скорость компиляции и объем обрабатывамых файлов выше

→ существенный прирост возможностей (тысячи пакетов расширений из которых десятки используются в реальных документах для конфигурации структуры документа, создания таблиц, диаграмм и др.)

→ частичная обратная совместимость форматов, почти полная прямая совместимость форматов.

"Глас народа" (2)

- В 2014 году интернет пестрел сообщениями о том, что аудиоплеер для ПК Winamp прекращает свое существование. Первая версия Winamp вышла в 1997 году, а разработка прекращена в 2013. **Что же это получается - за 16 лет авторы так и не смогли написать** — отладить, избавить от ошибок, насытить необходимыми возможностями (KISS+полнота) — **аудиоплеер? Написать так, чтобы он служил долгие годы без обновлений.**

  Молоток, разработанный 100-500 лет назад, работает до сих пор. Программа — тоже инструмент, почему же без поддержки она умирает? В отличие от молотка она не изнашивается. Если уж человечество не может написать плеер, то что говорить об операционных системах и более сложных программах?

  Говорят, что разработка Winamp была прекращена потому, что новая версия вышла с сильно измененным интерфейсом и не была принята пользователями. А зачем выпускать новую версию с радикально отличным интерфейсом, а не новую программу?

"Глас народа" (3)

- Bash.im 2015-12-14 09:12 #437065

  yyy: Заметил я такую странную вещь, многие программы (в том числе и ОС) имеют два этапа развития: сперва этап улучшения, когда каждая новая версия становится менее багнутой и более комфортной. Потом - "украшения", когда, казалось бы, программа уже достигла совершенства, но создатели начинают прикручивать к ней всякие свисто<...>ки (попутно оставляя и новые баги) и зачем-то полностью перекраивают интерфейс, так что вся комфортность улетучивается.

  Вот и получается, что я до сих пор сижу с windows XP, пользуюсь 2003 офисом, слушаю музыку в winamp 2.9, записываю диски в Nero 6, качаю старым uTorrent и т.д. и т.п. А на другом разделе жесткого у меня стоит Ubuntu 10.04, которая так же полностью меня устраивает…

"Глас народа" (4)

- Алан Голуб, "C&C++ правила программирования"

  Эта проблема когда-то касалась почти исключительно машин Macintosh, но Windows и здесь в последнее время выходит вперед. Компьютер Mac был спроектирован так, чтобы прежде всего быть простым в освоении. Положим, что тетушка Матильда Мак-Гиликатти часто заходила в компьютерный магазин, чтобы пользоваться их услугой по моментальной печати кулинарных рецептов. В итоге Матильда забирает компьютер домой и успешно вводит рецепты в течение нескольких месяцев. Теперь она хочет взять эти рецепты, проанализировать их химический состав и написать статью в научный журнал о коллоидных свойствах продуктов питания на основе альбумина. Доктор Мак-Гиликатти - хорошая машинистка, печатающая обычно около 100 слов в минуту, но эта ужасная мышь ее постоянно тормозит. Каждый раз, когда ее руки отрываются от клавиатуры, она теряет несколько секунд. Она пытается найти слово в своем документе и обнаруживает, что для этого должна открыть меню, ввести текст в диалоговое окно и щелкнуть по нескольким экранным кнопкам. В конце файла она должна явно указать утилите поиска возвратиться к его началу. (Ее версия редактора vi 15-летней давности позволяет выполнить все это при помощи двух нажатий клавиш - без необходимости отрываться от клавиатуры). Наконец, она обнаруживает, что на выполнение обычной работы - подготовки статьи в журнал - уходит в два раза больше времени, чем раньше, в основном из-за проблем с пользовательским интерфейсом. Ей не понадобилось руководство, чтобы пользоваться этой программой, - ну и что?

"Глас народа" (5)

- (lurkmore.to, вольная адаптация)
- Если бы методы ручного управления устройствами, ресурсами и многозадачности на уровне прерываний до сих пор были бы в почете — продолжилось бы развитие технологий расширения DOS (DOS/16M, DOS/4G и др) — система бы загружалась за секунды, а документы открывались до того, как отпустишь Enter.
- Однако, из-за недостатка программистов и программонстров такого уровня от "гламура" пришлось бы отказаться.