Федеральное государственное автономное учреждение
высшего образования

Московский физико-технический институт
(национальный исследовательский университет)
КЛУБ ТЕХА ЛЕКЦИЙ

АРХИТЕКТУРА КОМПЬЮТЕРОВ И ОПЕРАЦИОННЫЕ СИСТЕМЫ

III СЕМЕСТР
Физтех-школа: *ФПМИ*
Направление: *ПМИ*
Лектор: *Яковлев Виктор Вадимович*

Автор: *Савелий Романов*
Проект на github

Долгопрудный, Осень 2020 год.

Содержание

1 Unix системы

1.1 Компоненты операционной системы

Определение 1. Ядро — программа, которая запускается самой первой. Имеет привилегированное положение на процессоре и может взаимодействовать с «железом» напрямую.

Определение 2. Базовые библиотеки: зачастую входят в состав операционной системы, потому что привязаны к определенному ядру (версии ядра), чтобы максимально использовать его функциональность.

Определение 3. Служебные сервисы — ведение логов, обработка сетевых соединения, и т.д.

Определение 4. Минимальная пользовательская оболочка — чтобы пользователь мог взаимодействовать с системой.

1.2 Процессы

Определение 5. Процесс — любой экземпляр программы, работающей в операционной системе.

Примеры:

- Приложения
- Фоновые сервисы (демоны)
- Команды в терминале

Главные команды для мониторинга процессов:

- ps -A – список процессов
- pstree – иерархия процессов
- top

1.3 UNIX программы

Программы:

- **Бинарные исполняемые файлы.**
- **Текстовый файл, начинающийся с #!.** После #! указывается путь к интерпретатору (например, #!/bin/bash). Это могут быть скрипты на Python/Perl/Escript или Shell скрипт.

Название файла и его расширение не имеет значения. Любой файл, имеющий атрибут «исполняемый» является программой.

1.4 UNIX пользователи

Единственные привилегированный пользователь: root (UID = 0).

Непривилегерованные пользователи:

- Реальные пользователи, которые могут войти в систему (UID $\geq$ 1000)
- Фейковые пользователи, привязанные к сервисам (UID $<$ 1000).

Зачем сервисам нужны фейковые пользователи? Допустим, у вас есть веб-сервер и сервер баз данных. Теоретически какой-то пользователь может подключиться к веб-серверу и найти там уязвимость. Для того, чтобы минимизировать возможный ущерб от

уязвимости в одном из процессов, процессы запускаются под разными пользователями и не имеют права общаться друг с другом.

Непривилегерованные пользователи могут временно получить дополнительные привилегии:

- su - запускает терминал от имени root
- sudo - запускает любую программу от имени root
- Запустить программу с атрибутом SUID и владельцем root. SUID атрибут означает, что при запуске программа исполняется от того пользователя, кто является владельцем файла. Кстати, так и работает команда sudo, владелец sudo - root, а также у sudo проставлен атрибут SUID.
- Запустить программу с «capabilities» флагами. Эти флаги позволяют тонко настроить, что можно делать программе, а что - нельзя.

1.5 UNIX межпроцессорное взаимодействие

Процессы работают изолированно друг от друга. Единственный способ взаимодействовать друг с другом — использовать методы межпроцессорного взаимодействия ядра (Inter-Process Communication, IPC):

- Сигналы для коротких сообщений
- Каналы и сокеты для последовательных данных
- Разделяемая память, в которой хранятся большие куски данных

1.6 Процесс запуска

1. Загрузчик с диска или UEFI материнской платы выполняется в привилегированном режиме
2. Загрузчик запускает ядро, тоже в привилегированном режиме.
3. Ядро запускает первый непривелигерованный процесс: init или systemd.

1.7 Сервисы (“Демоны”)

Определение 6. Сервисы — это специальные системные службы, которые работают в фоновом режиме.

Минимальный сет во многих системах:

- **dhclient** – держит активным полученный IP адрес в сети
- **getty** – переключение между графическим/консольным входом в систему
- **sshd** – подключение по ssh
- **ntpd или chronyd** – синхронизация времени
- **syslogd** – системный лог

1.8 #!/bin/sh

В терминале вы работаете в интерпретаторе `shell`. `shell` предоставляет возможность выполнять команды и последовательности команд.

Бывают разные интерпретаторы `shell`:

- самодостаточный `shell` program: **FreeBSD**
- основной в `Linux`: **bash**
- debian: **dash**
- alpine: **busybox**
- maxOS X: **zsh**

1.9 Системный демон (systemd)

Определение 7. Процесс, который управляет всеми остальными процессами. При этом сам является обычным процессом. `Systemd` пришел на замену INIT-процессу.

Чем systemd лучше?

- Устранение интерпретации `shell` скриптов.
- Запуск сервисов параллельно

1.9.1 Концепты systemd

- Сервисы
- Цели:
  - Стандартные цели. Например, уровни `init` в System-V:
    - poweroff.target (init 0)
    - rescue.target (init 1)
    - multi-user.target (init 3)
    - graphical.target (init 5)
    - reboot.target (init 6)
    - sleep.target и hibertate.target (не присутствуют в System-V)
  - Специальные цели

1.10 Фоновые задачи

- **Нет привязанного к задаче терминала.** Для того, чтобы сигнал `SIGHUP` не останавливал исполнение задачи используется команда `nohup`.
- **Пишут в лог.** Иногда пишут данные в какую-то старинную локальную почтовую систему.
- **Запускается в единственном экземпляре.** Обычно это достигается с помощью file lock.

1.11 Дистрибутивы `Linux`

Ядро называется Linux. Базовая система — минимальная юзабельная система.

Дистрибутив:

- Ядро
- Базовая система
- GUI оболочка
- Программное обеспечение

1.11.1 Какие есть дистрибутивы?

Общего назначения:

- OpenSUSE
- Fedora
- Debian
- Ubuntu

Специфические дистрибутивы:

- Alpine Linux — очень легковесный, поэтому его хорошо ставить на виртуальную машину.
- Kali Linux — используется для тестов на безопасность.

2 Введение в Unix системы

2.1 Пользовательская сессия

Определение 8. Пользователь — это некоторое целое число, UID. У него даже может не быть имени.

Пользователи могут обьединяться в **группы**. Причем один пользователь может быть в нескольких группах. Группы нужны, чтобы организовывать совместный доступ к файлам и другим ресурсам.

Авторизация пользователя:

- Текстовый вход (TTY console)
- Графический вход (sddm, kdm, gdm)
- Удаленное подключение (ssh, telnet)

Расширить права введением пароля:

- su и sudo
- графические настройки

2.2 Подключаемые модули аунтефикации (PAM)

Определение 9. PAM - это набор библиотек, которые предоставляют API для авторизации пользователя и проверки прав.

Способы авторизации:

- /etc/passwd и /etc/shadow/

  Информацию о пользователях можно найти в файле **/etc/passwd**. Там для каждого пользователя указаны имя пользователя, UID, группы пользователей, полное имя, путь до домашнего каталога и используемый интерпретатор.

  !Некоторые пользователи впринципе не могут войти в систему. /sbin/nologin — тот shell, который сразу после входа моментально разлогонивает пользователя. Это нужно для того, чтобы иметь виртуальных пользователей. То есть преднозначенных для выполнения ровно одного процесса.

  Пароли хранятся в файле **/etc/shadow/** в хешированном виде.

- Биометрия

  Возможена авторизация не только по паролю. Можно также, например, по отпечатку пальца или по инфракрасной камере. Хотя многие бытовые ноутбуки последних годов выпуска не имеют драйверов под Linux, поэтому с этим возникают проблемы.

- LDAP сервис

  Если есть большое количество компьтерных организаций, то удобно **хранить все данные о пользователях централизованно**. В Unix системах для этого используется LDAP сервис.

2.3 Создание пользователей

Конечно, в графическом режиме можно зайти в настройки и нажать кнопочку добавить пользователя. Но как это сделать, если мы находимся на удаленном сервере?

Инструменты для создания пользователей:

- `useradd` - стандартная Linux команда. Ей достаточно передать параметры пользователя.

- `adduser` - интерактивный инструмент командной строки. Shell скрипт, которым позволяет сделать useradd в интерактивном режиме.

Параметры создания пользователя:

- Имя, группы, пароль, shell
- Изначальный контент домашней директории (обычно копируется из `/etc/skel`)

В `/etc/skel` в основном хранятся настройки пользователя:

- **.profile** - настройки отдельного пользователя, включая переменные окружения
- **.bashrc** - настройки командного интерпретатора bash.

2.4 Пользователь залогинился. Что дальше?

У каждого пользователя есть свой `shell`.

Определение 10. `Shell пользователя` — это команда, которая исполняется при логине пользователя. Ее можно найти в `/etc/passwd`

Shell программа инициализирует переменные окружения из `~/.profile` и `~/.bashrc`.

Общие для всех пользователей переменные могут быть указаны в `/etc/profile` и `/etc/bashrc`.

2.5 Более подробно про интерпретаторы `shell`'а

- самодостаточный `shell` program: **FreeBSD**
- основной в `Linux`: **bash**
- debian: **dash**
- alpine: **busybox**
- maxOS X: **zsh**

2.5.1 zsh

используется в macOS с недавнего времени по лицензионным соображениям. Также его используют и некоторые пользователи Linux из *соображений безопасности*, поскольку поддерживает некоторую дополнительную функциональность. Например, запрет на принципы переноса строки при копировании. Это контролируется, чтобы случайно не выполнить потенциально опасную команду.

2.5.2 bash

наиболее распространенный. У него есть много модулей для автодополнения.

2.5.3 busybox и dash

Более простые. Кстати, у busybox есть своя особенность: в Alpine Linux все стандартные линуксовые команды являются символическими ссылками на некоторый файл busybox размером 800КБ.

Определение 11. Busybox — это один большой бинарник, который в зависимости от именни программы, которую запускают, выполняет разную функциональность.

В системе `Android`, которая является Linux, тоже используется `busybox`. Такой подход нужен, чтобы **уменьшить количество программ и сэкономить место на диске**. Многие программы имеют дублирующую функциональность (например, разбор аргументов командной строки), поэтому логично сделать одну большую программу, которая ведет себя по-разному в зависимости от того, с каким именем вызывается.

2.6 Переменные окружения

- **PATH** — Тут прописано, *где можно искать файлы*, которые вы запускаете без указания полного пути

  (PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin)

- **EDITOR** — *редактор по умолчанию.* Например, он вызывается при git commit. (EDITOR=emacs)

- **LANG** — нужна, чтобы понять, на каком языке реализовывать интерфейс пользователя.

- **LC_ALL** — описывает, какую локаль использовать.

2.6.1 Принципиальное отличие между LANG и LC_ALL

`LANG` используется прикладными приложениями и **высокоуровневыми фреймворками** для того, чтобы понять, на каком языке отображать интерфейс. `LC_ALL` используется **стандартной C библиотекой**. В частности, в функциях ввода/вывода, форматировании дат и т.д. То есть это более **низкоуровневая** вещь.

2.6.2 Сокращения (Aliases)

alias ..='cd ..'
alias ды='ls'

Если вы обьявили переменную, она существует в рамках того `shell`, который сейчас запущен. Как только `shell` завершит свою работу или запустит дочерние процесс, никому эти переменные не будут известны. Команда **export** говорит, что эта *переменная должна быть доступна* ***всем*** *дочерним процессам*, а не только в текущем сеансе.

2.7 Текстовые кодировки

Есть два способа кодирования символов:

- Использовать несколько байт на один символ (wchar_t, std::wstring).
- Последовательность байт (char, std::string).

2.7.1 Пример: Русский язык

- `ANSI (7 bits)`: только US/UK Английский язык
- `KOI8-R и KOI8-U`: один бит для переключения языка
- `OEM Encoding`: поддержка на уровне «железа»
- `ANSI Encoding`: все символы кириллицы от Microsoft.

2.8 UTF-8

Определение 12. UTF-8 — универсальный способ, который **покрывает все языки** и при этом является **обратно совместимым** с кодировкой `ASCII`. Сейчас стал стандартном как в интернете, так и в написании исходных текстов.

2.8.1 В чем идея?

Если старший бит равен 0, то младшие 7 бит в точности совпадают с кодом символов в кодировке `ASCII`.

Если есть символы, которые не покрываются `ASCII`, то они кодируются последовательностью байт, где в старших байтах каждый из байт — единица.

Большинство европейских символов покрываются 2-мя байтами на один символ. Азиатские — 3 байта на символ. Смайлики, эмодзи — 4 байта на символ.

2.9 Вход пользователя

- Login + shell
- startx + xinitrc // что-то из 90-х ...

В современных системах, если это не серверная конфигурация, сразу запускается сеанс с графическим рабочим столом.

Графическая сессия состоит из:

- **X11 сервер**: отвечает за взаимодействие с видеокартой, клавиатурой и всем остальным.
- **Менеджер дисплея**: отвечает за авторизацию пользователя, ввод пароля и запуск нужного сеанса.
- **Менеджер окон**: отвечает за отрисовку окон и управлением жизненных циклов процессов.
- **Приложение рабочего стола**: приложение, которое рисует рабочий стол

2.10 X сервер

Определение 13. Сервер — это процесс, который принимает входящие подключения (от одного или нескольких клиентов) и как-то их обрабатывает.

Определение 14. X сервер — это некоторая программа, которая имеет доступ к видеокарте, клавиатуре, мыши.

Когда мы запускаем некоторое графическое приложение, оно устанавливает соединение через локальный сокет с сервером `DISPLAY`.

В переменной окружения `DISPLAY` указан адрес сервера, номер терминала (DISPLAY=:0.0 значит, что к `DISPLAY` можно подключиться по адресу `localhost`, номер терминала – 0).

Соединение можно пробросить через протокол ssh: `ssh -X user@hostname`

2.11 Приложения

Приложения - это некоторые пакеты:

- **обычные архивы** (FreeBSD)
- **RPM** (rpm to install, rpmbuild to build)
- **DEV** (dpkg to install, debuild to build)

Пакеты:

- Набор файлов
- Установочные скрипты и скрипты деинсталяции.
- Метаинформация: название, версия, описание, зависимости

Пакеты зависят друг от друга:

- Библиотеки конкретных версий
- Продукт может быть разбит на несколько частей: отдельные части под конкретные архитектуры и общие для всех архитектур.

Для того, чтобы управлять зависимостями и скачивать пакеты из удаленных репозиториев существуют **пакетные менеджеры**:

- **apt-get** (apt) для Debian/Ubuntu (dpkg)
- **yum** (dnf) для Fedora (rpm)
- **apt-get** для AltLinux (rpm)

2.12 Установка из исходного кода

Большинство программ в Linux имеют открытый исходный код. Исходный код запаковывают в архивы. Исходный код требует каких-то дополнительных библиотек. Стандартный способ установки:

./configure
make
make install

Иерархия каталогов:

1. **Корневой каталог** / имеет bin, lib, ...
2. /**usr** директория имеет очень похожую структуру
3. /**usr**/**local** содержит кастомные конфигурации. То есть то, что не покрывается софтом, установленным из пакетов.

Такая структура имеет исторические причины. В старых UNIX-системах была привязка к железу. Обычно был один маленький, быстрый диск и другой, большой, но медленнее. Большой монтировался к `/usr`.

2.12.1 Как сейчас устроено распределение?

- Основной каталог — `/usr`
- В корневой каталог помещается все, что востребовано в качестве минимальной системы. То есть это минимальный набор файлов, чтобы посмотреть состояние системы, список файлов, что-то подмонтировать и т.д.

  Хотя в некоторых дистрибутивах, например, Fedora, `/bin` и `/lib` являются просто символическими ссылками на соответствующие разделы `/usr`. При этом если вы запускаете какую-то утилиту, то **поиск происходит от корня в глубину**. То есть сначала /bin, потом /usr/bin, а уже затем - /usr/local/bin. Поэтому чтобы запустить правильную версию, надо либо указать полный путь к программе, либо поменять переменную `PATH`.

2.13 Исполняемые файлы и библиотеки

1. **Статические библиотеки**
   - Индексированный архив обьектных файлов
   - Сейчас редко используются
2. **Динамические библиотеки**
   - Как обычные исполняемые файлы
   - Нет точки входа
   - Есть таблица символов

2.14 Генерация Makefile'ов

Зачем нужны конфигурационные скрипты? Почему бы в поставке софта не использовать сразу готовый Makefile? Синтаксис Makefile'ов может сильно отличаться. Кроме того, необходимые нам библиотеки могут располагаться в разных путях.

Решения:

- **./configure script**: требует только shell и базовые инструменты, но трудно поддерживать (решение для этого: `autotools`)
- **qmake/jam/scons/custom scripts**: обычно привязаны к конкретному фреймворку.
- **cmake**

2.15 CMake

Это универсальный `C` и `C++` инструмент для генерации `makefiles`. Имеет много вспомогательных модулей для пакетов с открытым исходным кодом.

2.16 Распространение софта

- Предоставить файлы для компиляции: ./configure, CMakeLists.txt, ...
- Предоставить ./debian/ файлы (debuild) для Ubuntu/Debian
- Предоставить package.spec (rpmbuild) для разных дистрибутивов
- Ubuntu LaunchPad
- openSUSE Build Service

3 Целочисленная арифметика

3.1 Уровни абстракции «железа»

- «p» и «n» полупроводники, проводники, конденсаторы и резисторы
- Транзисторы и диоды
- Логические элементы
- Арифметическо-логические юниты
- Центральный процессор
- Компьютерная система

Компьютер состоит из элементарных блоков, из которых строятся электрические элементы. Они в свою очередь образуют логические элементы.

Ключевая часть компьютера - центральный процессор. А главная часть ЦП - арифметическо-логический блок, который выполняет все вычисления.

3.2 Типы сигналов

Определение 15. Сигналы — это изменения напряжения в электрической сети. Их можно регистрировать и дальше обрабатывать.

Сигналы бывают двух видов:

- **Аналоговый.** Сила сигнала изменяется непрерывно с изменением напряжения.
- **Цифровой.** Есть всего два состояния - высокий, либо низкий.

3.3 Дискретные сигналы

- **Логический «0».** Все, что ниже определенного значения напряжения — это нулевой уровень.
- **Логическая «1».**
- **Неопределенное состояние.** Напряжение между 0 и 1, не приводящее к изменению состояния.

- **Hi-Z состояние.** Физически какой-то контакт внутри процессора отключается от общей схемы, чтобы не оказывать влияние на все остальные компоненты.

3.4 Арифметико-логический юнит

Базовый юнит центрального процессора для реализации операций над числами:

- Битовая логика
- Побитовые сдвиги
- Сложение и вычитание
- Умножение и деление

Управление ходом программы — тоже арифметические операции.

3.5 Представление чисел и переполнения

Было на семинарах...

3.6 Значения из нескольких байтов

- Минимальный адресуемый обьем данных — 8 бит.
- За один такт процессора он может адресовать одно **машинное слово** процессора - 32 или 64 бита.

3.7 Big-Endian и Little-Endian

- **Big-Endian:** Выглядит как человеческое представление байтов. сначала старшие байты, потом - младшие Пример: большинство RISC процессоров по умолчанию.
- **Little-Endian:** сначала младшие байты, потом старшие Пример: Процессоры Intel и ARM.

3.8 Ковертирование endian-ов

Беззнаковые типы можно конвертировать просто побитовыми сдвигами. Достаточно переставить байты в обратном порядке.

Со знаковыми типами есть проблемы, потому что там первый бит отвечает за знак.

3.9 Упаковка структур

Размер любой структуры должен быть кратен размеру машинного слова. Это связано с оптимизацией времени чтения/записи.

Поэтому если в структурке хранится два поля - на 16 бит и на 8 бит, то стуктура будет занимать 32 бита, а не $16 + 8 = 24$. Ведь размер машинного слова в x86 равен 32 битам.

Если же прописать у структуры специальный атрибут, компилятор уложит поля вплотную.

3.10 Где используется атрибут packed?

- Бинарные данные
- Сетевые пакеты
- Драйвера

В остальных случаях использование packed ненужно и даже вредно.

4 Команды процессора

4.1 Процессор

Процессор умеет только манипулировать с целыми числами. Значение адреса - тоже число.

4.2 Процессор с точки зрения пользователя

У процессора есть:

- Регистры - eax, ebx, ...
- Флаги - ZF, CF, SF
- Указатель на текущую команду (PC - pointer count)

4.3 Команды x86

nop - 0x00 halt - 0x10 call Dest - 0x80, Dest - 5 байтов ...

Команды кодируются переменным количеством байт. Количество байт может доходить до 6.

4.4 Представление структур программы в виде простых инструкций

Было на семинарах, на ассембелере писать уже должны уметь...

4.5 CISC

Определение 16. CISC — **C**omplex **I**nstruction **S**et **C**omputing

Процессоры: x86, Z80, PDP-11/VAX

Это тип архитектуры процессора, где:

- Команд много, на все случаи жизни
- Кодируются переменным количеством байт
- Разные режимы адресации
- Упрощают жизнь программисту на ассемблере

Например, инструкция **loop** Address позволяет реализовать цикл одной командой.

4.6 Конвейер

- Загрузка инструкции
- Декодирование команды
- Выполнение
- Доступ к памяти
- Записать результат в регистр

4.7 Проблемы ковейеризации

- Инструкции имеют разную длину в байтах
- Разные инструкции выполняются за разное число тактов
- Работа с памятью и с регистрами задействует разные блоки процессора

Одна из задач компилятора с точки зрения оптимизации - минимизировать время простоя при конвейеризации. Для этого компилятор может менять порядок команд, если это не изменит результата, но поможет с ковейеризацией.

4.8 RISC

Определение 17. RISC — **R**educed **I**nstruction **S**et **C**omputing

Процессоры: ARM, AVR, MIPS, PowerPC

- Только простейшие инструкции
- Инструкции фиксированной длины
- Почти у всех инструкций одинаковое время выполнения
- Адресация только по регистрам. Не можем читать напрямую из памяти

4.9 Современные RISC процессоры

- **PowerPC:** Playstation, XBox, суперкомпьютеры IBM
- **MIPS:** WiFi/Bluetooth-адаптеры, процессоры в телевизорах
- **ARM:** Смартфоны
- **AVR:** Arduino

4.10 Вещественная арифметика, IEEE 754

Было на семинарах. Очень очень лень это все дублировать...

5 Языки ассемблера и двоичный код

5.1 Стадии трансляции

- **Препроцессинг текста.** На выходе - текст.
- **Абстрактное дерево синтаксического разбора** и таблицы символов. Формат этого дерева внутренний, зависит от договоренностей.
- **Код на языке ассемблера или двоичный код.**

5.2 Языки ассемблера

- Линейная последовательность команд
- Нет вложенных конструкций
- Нет вычислимых выражений. Точнее бывает, но только с константами и это обрабатывается на стадии препроцессинга (syntax sugar).
- Текст распознается регулярным выражением, а не контекстно-свободной грамматикой.

Любую программу языка ассемблера можно скомпилировать, а потом выполнить обратную операции - дизасемблирование. При этом код программы будет почти в точности совпадать.

5.3 Кодирование команд

Есть две процессорные архитектуры:

- **RISC.** Каждая команда длинной в машинное слово. Поэтому зная начало кода можно извлечь любую инструкцию.
- **CISC.** Одна команда - произвольное количество байт.

5.4 Команды и регистры

- Почти всегда процессор умеет выполнять действия только над регистрами.
- Регистр - это не просто быстрая память. Это память, к которой процессор обращается напрямую.
- Доступ к памяти - достаточно сложная операция.
- Некоторый процессоры (например, x86) имеют команды для адресации памяти, но на самом деле они сводятся к загрузке этой памяти в регистры.

5.5 Кодирование команд

В команде кодируется:

- Какая именно команда
- С какими регистрами она работает
- Константы. При этом, не все константы можно закодировать в команде, потому что их слишком много.

5.6 Ключевая проблема

Нельзя впихнуть невпихимуемое. Длина команды - всего 32 бита.

Как закодировать адрес? Очевидно, нельзя адресовать все виртуальное адресное пространство, ведь это 4ГБ, а у нас есть всего 12 бит на кодирование адреса.

Поэтому код команды устроен так: 8 бит на значение и 4 бита на сдвиг с вращением. С помощью этого можно закодировать наиболее часто используемые адреса.

5.7 Procedure Linkage Table

Библиотечные функции находятся далько от нашего коде в адресном пространстве. Поэтому возникает проблема с тем, чтобы вызвать эту функцию, ведь как было показано выше, мы не можем закодировать в команде произвольный адрес перехода. Эту проблему решает Procedure Linkage Table.

Определение 18. Procedure Linkage Table — это таблица в памяти, рядом с исполняемым кодом, где записаны реальные адреса библиотечных функций. Это позволяет вызвать библиотечную функцию, которая находится далеко в адресном пространстве.

5.8 Про кодирование команд

У ARM есть расширения:

- **Thumb.** 16 битные инструкции
- **Jazelle.** Декодирование байткода Java. Раньше было полезно, чтобы запускать игрушки на слабых телефонах.

5.9 Уровни абстракции

- Высокоуровневый язык
- Промежуточный язык
  - байткод (Java, CLI или PyPy)
  - LLVM - абстракция от ассемблеров разных архитектур
- Язык ассемблера
- Двоичный код

5.10 Зачем понимать кодирование команд

- Just-In-Time компиляция
- Эмулятор компьютерной системы
- Трансляция команд

6 Способы ускорения выполнения кода

6.1 SRAM (Static RAM)

- Время чтения - 1 такт
- Время записи - 2 такта
- Тактовая частота зависит от размеров транзистора

6.2 DRAM (Dynamic RAM)

- 1 транзистор + 1 конденсатор
- Конденстатор требует перезарядки после каждого чтения
- Чтение/запись занимают много времени
- Периодическая регенерация (каждый 64 нс) из-за утечек

6.3 DRAM vs SRAM

Если вы делаете какую-то схему на базе микроконтроллеров и вам не требуется больших обьемов памяти, намного проще использовать SRAM. Размер памяти в SRAM измеряется порядками Кб, Мб на каждую схемку.

6.4 Локальность доступа

Иногда у вас есть нужен небольшой кусок данных, которые вы часто используете. Хочется иметь к нему быстрый доступ. Такая память называется кэш.

6.5 Информация о кэше в Linux

Определение 19. Кэш — это память, размер которой меньше размера всей памяти, который вы можете адресовать. Он работает намного быстрее и хранит нужный вам кусок.

Информацию о кэшах в вашем компьютере можно посмотреть в каталоге /sys/devices/system/cpu/cpu0/cache

6.6 Уровни кэша (в x86)

- **L1 (index0 + index1)** — самый близкий кэш микроинструкций и текущий данных, с которыми оперирую микроинструкции.
- **L2** — общий кэш, связанный с ограниченным количеством ядер (обычно одним).
- **L3** — общий кэш, связанный со всеми ядрами.

6.7 Ключевая проблема

Размер кэша очень маленький. Вся память в несколько Гб нельзя упихать в несколько Мб.

6.8 Причины кэш-промахов

Определение 20. Кэш-промах — это когда вы обращаетесь к какому-то участку памяти, который отсутствует в памяти.

- Первое обращение к определенной области памяти
- Данные были выгружены из-за ограниченного размера кэша.
- Данные были выгружены из-за ограниченной ассоциативности. Если обращаться к данным в рандомном порядке, процессор не догадается, какие из них надо закэшировать.

6.9 Как устроен кэш

Определение 21. Блок — минимальный адресуемый обьем данных в кэше. Это 64 байта для Intel.

Определение 22. Кэш-линия — блок + метаданные, определяющие адрес в памяти.

Определение 23. Набор — связан с некоторым адресом в оперативной памяти. Для L3 — 12 линий по 64 байта = 768 байт.

Кэш состоит из независимых наборов. Для Core i5/Gen10 размер кэша L3 6Мб = 64 байта в блоке * 12 линий в наборе * 8192 наборов.

6.10 Что значит Cache-Friendly

- По возможности использовать непрерывные блоки данных.
- Выравнивать данные по границе кэш-линии: Alignas(64) в Си.

6.11 Как использовать кэш?

- Нет команд, которые бы управляли кэшем. Нельзя програмно положить что-то в кэш.
- Компиляторы Си/C++ и пр. могут генерировать код для предзагрузки данных.
- Увеличить вероятность попадания в кэш можно размещая данные последовательно.

6.12 Профилирование

Исследовать программу на производительность можно с помощью команды valgrind –tool=cachegrind ПРОГ [АРГ0][... АРГn]

6.13 Стадии выполнения команд

Современные процессоры являются **суперскалярными**. Это означает, что они не обязательно выполняют команды строго последовательно.

Существуют следующие стадии выполнения команд:

- Instruction Fetch
- Instruction Decode
- Execute
- Memory Access
- Register Write Back

Каждая стадия задействует отдельные блоки процессора, которые слабо связаны между собой. Поэтому можно начинать выполнять следующую команду даже если не закончили выполнение предыдущей. Такой способ выполнения называет **конвейеризацией выполнения**.

6.14 Сверхдлинные конвейеры

С одной стороны это круто, потому что можно хорошо распараллеливать исполнение команд. Но в случае условных инструкций длинный ковейер придется долго откатывать.

У современных процессоров длина конвейера - 8...15.

6.15 Идея спекулятивного выполнения

- В случае условного выполнения вычисляем сразу обе ветки. Если не угадали ветку, результат просто отбрасываем.
- Компилятор может переставлять процессорные инструкции местами, если это не вляет на результат.

6.16 Векторные инструкции

Было на семах.

7 Прерывания и системные вызовы

Определение 24. Прерывание — это сигнал от программного или аппаратного обеспечения, сообщающий процессору о наступлении какого-либо события, требующего немедленного внимания.

7.1 Обработка прерывания в x86

- Каждое устройство имеет свой собственный номер прерывания (IRQ).
- Процессор получает сигнал INT и прекращает исполнение текущекого контекста.
- Каждое прерывание имеет свой адрес в векторе прерываний.
- Прерывания бывают не только аппаратные. Его можно сымитировать, вызвав прерывание программно.

7.2 Маска прерывания

Иногда хочется запретить прерывания в каком-то месте. Для этого существует маска прерываний.

7.3 Номера прерываний

- 0 — нажатие на кнопку включения или Reset
- 1...15 — Аппаратные прерывания. Тут задействованы реальные железяки.
- 16+ — программные прерывания, вызванные через int.

7.4 Базовый вектор прерываний

Кто прописывает нам функции, которые будут вызваны при прерывании?

- BIOS — содержит весь минимальный набор функций, нужный после включения.
- Обработка железяк (клавиатура, ...).
- Функции для обращения к данным на диске и загрузки операционной системы.

7.5 Обработка прерываний

- Сохранить EIF на стек
- Проставить 1 в флаг IF
- Перейти на инструкцию по адресу IDTR + offset

Перед вызовом обработчика прерывания:

- Сбросить флаги
- Переключить текущее адресное пространство
- Заменяется стек

Выполнение функции обработчика прерывания:

- Сохранить текущее значение регистров и флагов
- что-то поделать...
- бернуть состояние регистров и флагов обратно

7.6 Ядро

При запуске операционной системы есть одна программа - ядро. Она может делать все что угодно.

7.7 Режимы исполнения (x86)

Обычный режим:

- Процесс имеет доступ только к своей собственной памяти.
- Виртуальное адресное пространство каждого процесса начинается с адреса 0.
- Процесс не может взаимодействовать с устройствами.

Превилегированный режим:

- Полный доступ к физической (не виртуальной) памяти.
- Полный доступ к портам ввода/вывода.
- Некоторые дополнительные команды, не доступные в обычном режиме исполнения.

Определение 25. Системный вызов — механизм для взаимодействия обычных процессов с ядром. Он позволяет обратиться к функциональности ядра, чтобы сделать недоступные в обычном режиме операции.

7.8 Ядро

- Обычный ELF файл.
- Запускается в привилегированном режиме.
- Отвечает за взаимодействие с железом.

7.9 Запуск ядра

- Проинициализировать устройства.
- Проинициализировать вектор прерываний.
- Найти, загрузить и запустить все драйвера.
- Понизить привилегии процессора.
- Запустить первый пользовательский процесс — init.

7.10 Взаимодействие с ядром (x86)

- Все процессы непривилегированы.
- Единственный способ получить доступ к железу — переключиться в режим ядра.
- Все прерывания переключают процессор в привилегированный режим.
- Используйте int для того, чтобы получить доступ к системным вызовам.

7.11 Типы архитектуры ядер

- **Монолитные ядра.** Одна большая программа, которая исполняется в привилегированном режиме. Примеры: Linux.
- **Микроядерная.** Только небольшая часть ядра запускается в привилегированном режиме. Большая часть подсистем работает как пользовательские процессы. Примеры: Minix3.
- **Гибридная.** Модульная, но не одна большая программа, запускаемая в привилегированном режиме. Примеры: Windows, Mac, BSD или Linux.

7.12 Системные вызовы

Системный вызов можно осуществить с помощью:

- int 0x80 (Linux, BSD)
- sysenter/sysleave инструкции (Intel)
- syscall/sysret инструкции (AMD64/x86_64)

7.13 INT 0x80

- В eax сохраняется номер системного вызова
- Параметры сохраняются в ebx, ecx, ...
- Возвращаемое значение сохраняется в eax
- Конвенции вызовов отличаются от принятых в языке C!

8 Исполняемые файлы. Загрузчики

8.1 Стадии компиляции

- Получить бинарный код из исходного кода.
- Сделать из этого **исполняемый файл**.

8.2 x86 Форматы исполняемых файлов

- Простой бинарник. Никаких заголовков, только код с нулевым отступом.
- ELF (UNIXes)
- EXE (Windows)

8.3 ELF

- **Магические байты**: 0x7F, 'E', 'L', 'F'
- **Бинарный заголовок**:
  - архитектура процессора
  - разрядность процессора
  - точка входа
  - позиции сегментов

8.4 Интерпретатор ELF файлов

Специальная программа (/lib[64]/ld-linux.so) для загрузки программы в память. Она нужна, чтобы:

- Загружает сам файл память и все требуемые библиотеки.
- Аллоцирует память на стек.
- Прыгает на точку входа программы.

8.5 Динамическая библиотека vs Исполняемый файл

- Тот же самый ELF формат
- У исполняемого файла должна быть точка входа.
- У библиотеки должна быть таблица символов.

Команда ld линкует библиотеки.

Определение 26. Позиционно независимый код — код, который может быть загружен в произвольную область памяти. Его можно получиться с помощью опции -fPIC у gcc.

8.6 Flat-form файл

- Нет заголовков.
- Для запуска достаточно просто перейти на начало файла.
- Последняя инструкция должна быть ret.

8.7 Загрузчик ядра

Этапы работы загрузчика:

- Найти ядро на диске.
- Загрузить его и разместить в памяти.
- Запустить ядро.

8.8 Классический загрузчик

- BIOS определяет порядок дисков загрузки.
- На дисках проверяются первые 512 байт содержимого, **главная загрузочная запись** (master boot record). Она содержит:
  - Диск является загрузочным, если последние байты 0x55 или 0xAA.
  - 64 байта содержат основную таблицу разделов.
  - 446 байт содержат исполняемый код загрузчика.

8.9 Условия работы загрузчика

- Процессор находится в реальном режиме работы.
- Может общаться с видеопамятью напрямую.
- Могут использовать функциональность BIOS для работы с:
  - дисками
  - клавиатурой
  - COM-портами

8.10 Обзор загрузчиков

- Простые: ntldr (Windows), loadlin (Linux)
- Универсальный: GRUB
  - Умеет загружать все операционные системы
  - Выбор ОС и опций загрузки
  - Красивый графический фон

А как код GRUB умещается в эти 446 байт? Ответ — никак. На самом деле код разбит на модули, которые подгружаются после основного кода по необходимости.

8.11 Загрузка ELF снимка

- Переключение в 24-битный режим адресации памяти
- Загрузка файла ядра с диска и размещение его строго после 1Мб в адресном пространстве.
- Найти магические байты GRUB в снимке ядра. Точка входа располагается рядом с этими магическими байтами.
- Отключаем прерывания.
- Переключаемся в защищенный режим.
- Запуск!

В ноутбуках, особенно ультрабуках, чаще используется другой подход:

8.12 GUID Таблица разметки

Каждый диск получает уникальный 128-битный ключ.

Сейчас для разработки загрузчиков используют более высокоуровневую вещь — UEFI API (Unified Extensible Firmware Interface)

9 Операционные системы. Файловые системы.

9.1 Цель операционных систем

Глобальная цель — абстрагировать все оборудование, на котором работают пользовательские приложения через некоторую прослойку.

Разработчики не должны писать софт под каждый компьютер по отдельности, операционные системы помогают решить такие проблемы как:

- Вычислительные системы могут иметь разные архитектуры процессора.
- Разные модели памяти.

9.2 Основные компоненты

- Ядро.
- Минимальный набор библиотек и инструкментов.
- Высокоуровневые библиотеки и фреймворки.
- Окружение рабочего стола.

9.3 Абстракции уровня API

Стандартные библиотеки: C17, C++17.

Взаимодействие с ОС: Portable Operating System Interface — POSIX (в основном системные вызовы).

9.4 Реализации абстракций уровня API

Стандартные библиотеки: libstdc++ (Linux, *BSD), msvcp.dll (Windows) или libc++ (MacOS).

Реализации API POSIX стандарта: glibc, bionic, ...

9.5 Использование библиотек

- Внутри программы библиотеки загружаются с помощью ld.so.
- Содержимое библиотек живет внутри адресного пространства процесса.
- Функции должны вызываться с помощью Procedure Linkage Table.

9.6 Взаимодействие ядром

Системные вызовы можно использовать с помощью прерываний.

При этом не для всех системных вызовов нужно менять режим исполнения ядра. Системные вызовы __vdso_clock_gettime, ... (x86_64) функционируют как обычные функции. *Узнать больше: man 7 vdso.*

9.7 Подсистемы ядра

- Драйвера
- Управление памятью.
- Планировщик процессов и потоков.
- Межпроцессорное взаимодействие
- Поддержка файловых систем.

9.8 UNIX Виртуальная файловая система

В отличае от Windows и других стремных систем тут нет дисков. **Вся файловая система организована в виде дерева.** Совершенно неважно, как вы подключили устройство, вы всегда работаете единообразным образом.

9.9 Типы файлов в UNIX

- Обычный файл
- Каталог
- Устройство. Бывают блочные утройства и символьные устройства.
- Символические ссылки.
- Именованый канал (pipe).
- Сокет.

9.10 Обычные файлы

- Просто хранит данные.
- Произвольный доступ к данным с помощью lseek.

9.11 Каталоги

Определение 27. Каталог — просто файл определенного формата, чтобы хранить содержимое директории.

- Директория состоит из struct dirent.
- POSIX определяет содержимое struct dirent:
  - inode, т.е. уникальный идентификатор определенного файла.
  - размер записи.
  - имя файла.

9.12 Ссылки

Символическая ссылка ведет себя практически точно также, как и тот файл, на который она ссылается.

Определение 28. Жесткая ссылка — это другое имя того же файла. Это тоже файл, имеющий тот же inode (идентификатор файла). Жесткая ссылка увеличивает количество ссылок на файл. Когда ссылок станет 0, файл будет реально удален.

9.13 Устройства

Блочные устройства — диски. Символьное устройство отличается от блочного тем, что тут мы можем только писать и читать, но не перемещаться по содержимому.

9.14 FIFO и сокеты

FIFO файлы используются для простейшего межпроцессорного взаимодействия.

Сокет — более сложная концепция. Использовать ее надо через системные вызовы (например, connect).

9.15 Файловые системы

Каждая файловая система:

- подмонтируется к некоторому поддереву.
- самодостаточна

9.16 Типы физических файловых систем

- Дисковые: FAT, NTFS, ...
- Без структуры: SWAP
- Сетевые
- Виртуальные

9.17 Адресация файлов

С каждым файлом связано два числа: inode и номер устройства.

Определение 29. inode — ключ файла внутри одной файловой системы

9.18 Физическая память

- HDD — дешево, но есть накладные расходы по времени на перемещение головки.
- SSD — флэш память. Работают сильно быстрее, чем HDD. Но есть ограничение на количество циклов записи, поэтому их надо менять.

9.19 Концепции файловой системы

- Уровень пользователя: доступ к файлу по его имени (string).
- Уровень процесс: открыть файловый дескриптор (int).
- Уровень ядра: запись в виртуальной файловой системе (st_dev и inode).
- Уровень файловой системы: поиск данных по inode.

9.20 Linux файловые системы для дисков

- ext2/3/4 — самые надежные.
- XFS — подходит для хранения больших файлов.
- ReiserFS — наиболее подходящая для большого количества маленьких файлов.

10 Аллокация памяти

10.1 Физическая память

Основные параметры:

- частота шины (например, 1500 МГц).
- передача за одну секунду (например: 3000 MT/s).

Определение 30. Латентность — задержка между командой и передачей данных.

10.2 Физическое адресное пространство

Диапазон — 2^{bits_count}

10.3 Адресное пространство процесса

У каждого процесса есть свое адресное пространство. Оно никак не пересекается с адресным пространством других процессов.

Ограничения:

- 32-битные системы — 4 Гб.
- 64-битные — 256 Тб.

Пример содержимого:

- Пространство ядра — 0xc0000000
- Стек — 0xbf800000
- Разделяемые библиотеки
- Куча
- Программа — 0x004d9000

Для того, чтобы реализовать менеджмент памяти используется **страничная адресация**.

10.4 Страничная адресация

- Разбиение памяти на **страницы** фиксированного размера.
- У каждой страницы есть свои атрибуты.

10.5 Страничная адресация в x86-32

32-битный адрес в виртуальном адресном пространстве разбивается на 3 части — индекс в таблице первого уровня (10 бит), индекс в таблице второго уровня (10 бит) и смещение внутри страницы (12 бит).

- Специальный регистр CR2 указывает на таблицу первого уровня текущего процесса.
- Каждая запись в таблицу первого уровня указывает на таблицу второго уровня.
- Записи в таблице второго уровня содержат указатели на страницы.

- Размер страницы — 4 Кб
- В каждой таблице есть до 1024 записей

Советую посмотреть участок 30 сек для лучшего понимания.

При кодировании у нас остается 12 бит для флагов — это настройки страницы.

10.6 Нарушение доступа к странице

Флаг P (present — файл существует в физицеской памяти) равен 0. Тогда процессор бросает исключение. Ядро обрабатывает это исключение.

10.7 Аппаратная поддержка

- MMU (Memory Managment Unit) считает реальный указатель на используя указатель в виртуальном адресном пространстве.
- TLB (Translation Lookaside Buffer) предназначен для хранения текущих таблиц в памяти.
- Размер страниц прописан на уровне процессора, вы не можете его поменять программно.

10.8 Аллокаторы

- C++: стандартные new и new[] используют malloc/calloc. Это можно настраивать.
- C: malloc/calloc.

10.9 Выделение памяти на кучу

За счет чего реализованы библиотечные функции malloc и calloc?

Для этого используется один из системных вызовов:

- **brk** — устаревшая, но простая. Используется в glibc malloc до 128 Кб.
- **mmap** — более гибкий инструмент.

10.10 mmap

Было на семах.

Узнать больше: *man mmap*

10.11 Ограничения на память

Ограничения можно посмотреть с помощью *ulimit -a*.

10.12 Что если память кончилась?

Out-of-memory убийца процессов:

- Найти жертву для убийства
- Очистить память

С каждый процессом связан некоторый рейтинг (/proc/[pid]/oom_score), на основе которого выбирается жертва.

Рейтинг вычисляется ядром. У процессов с высоким потреблением памяти рейтинг более высокий, у привилегированных процесоов — более маленький.

10.13 Overcommit

Определение 31. Overcommit — это когда мы выделяем памяти больше, чем есть в системе. Это не вылетит пока мы не захотим ее использовать.

Настроить стратегию при overcommit можно командой sysctl.

Варианты стратегий:

- OVERCOMMIT_ALWAYS
- OVERCOMMIT_GUESS — стоит по умолчанию. Решение выносится на усмотрение ядра. Почти всегда это OVERCOMMIT_ALWAYS.
- OVERCOMMIT_NEVER

11 Процессы

11.1 Что такое процесс?

Определение 32. Процесс — это экземпляр программы в одном из состояний выполнения. Каждый из процессов выполняется в своем изолированном адресном пространстве.

11.2 Аттрибуты процесса

Память:

- Значения регистров процессора.
- Таблицы и каталоги страниц виртуального адресного пространства.
- Private и Shared страницы памяти.
- Отображение файлов в память.
- Отдельный стек в ядре для обработки системных вызовов.

Файловая система:

- Таблица файловых дескрипторов.
- Текущий каталог.
- Корневой каталог.
- Маска аттрибутов создания нового файла umask.

Другие аттрибуты:

- Переменные окружения
- Лимиты
- Счетчики ресурсов
- Идентификаторы пользователя и группы

11.3 Информация о процессах

Команда **ps** показывает список процессов, **top** — потребление ресурсов.

11.4 Жизненный цикл процессов

- Выполняется (Running)
- Остановлен (Stopped)
- Временно приостановлен
  - Suspended — может быть завершен
  - Disk Suspended — не может быть завершен
- Исследуется (Tracing)
- Зомби (Zombie)

Как устроен планировщик заданий?

11.5 Round Robin

Windows 9x, старые UNIX

Аппаратный таймер время от времени генерирует прерывание, после чего планировщик выбирает, какой процесс будет выполняться дальше.

11.6 Приоритет

Определение 33. Приоритет процесс — это численное значение от -20 до 19. Оно обозначает то, сколько раз пропустить планировщиком заданий.

- Значение от
- Численное значение —

11.7 Многоуровневая очередь

Linux, xBSD, Mac, Windows

Эта схема отталкивается от того, что есть два типа процессов:

- Не очень активно взаимодействующие с внешним миром (что-то вычисляют). Им надо просто не мешать выполняться.
- Много взаимодействующие с пользователем. Их надо переключать быстро.

Есть набор очередей, уровней, которые рассположены по увеличению времени переключения. При переключении очереди, если процесс освободил процессор до следующего события, он перемещается в следующую очередь (более редко переключаемую), иначе — в предыдущую.

11.8 Ничегонеделание

while (1) // Так делать очень плохо, потому что вы загружаете процессор

while (1) sched_yield(); // Передаем управление другому процессу

11.9 Создание процесса

Системный вызов fork(). Было на семах. *Узнать больше: man fork*

11.10 Копия процесса

При fork создается полная копия процесса. Память, регистры, ... — точная копия.

Не копируются:

- eax, rax
- Сигналы, ожидающие доставки
- Таймеры
- Блокировки файлов

В связи с копированием всего адресного пространства может возникать необычное поведение. Например, буфер ввода и вывода тоже копируется. Поэтому если не сделать fflush, один и тот же текст может продублироваться.

11.11 Ограничения

/proc/sys/kernel/pid_max — максимальное число одновременно запущенных процессов. /proc/sys/kernel/threads_max — максимальное число одновременно выполняющихся потоков (каждый процесс — уже один поток).

11.12 Дерево процессов

- Процесс с номером 1 — systemd
- У каждого процесса кроме systemd есть свой родитель
- Если родитель процесса умирает, то его родителем становится systemd.
- Если ребенок умирает, про это узнает его родитель.

11.13 Завершение работы процесса

- Системный вызов _exit(int)
- Функция exit(int)
- Оператор return в main

11.14 Ожидание завершения процесса

Системный вызов waitpid.

11.15 Zombie процессы

Удалением зомби из таблицы процессов занимается родитель — вызовом wait или waitpid.

11.16 exec

Системный вызом exec позволяет заместить тело процесса другой программой. *Было на семах.*

11.17 Аттрибуты процесса, сохраняемые exec

- Открытые файловые дескрипторы
- Текущий каталог
- Лимиты процесса
- UID, GID
- Корневой каталог — root.

11.18 SUID-флаг

Это довольнительный аттрибут выполняемого файла. Он означается, что файл запускается от имени того пользователя, который является владельцем файла.

11.19 setuid/getuid vs geteuid

Определение 34. Реальный user id — тот пользователь, который действительно запускает текущий процесс. Например, запуская с помощью sudo все равно можно узнать, кто именно запустил.

Определение 35. Эффективный user id — определяет то, с какими правами процесс исполняется.

11.20 Лимиты

Например, перед запуском процесса можно проставить лимиты.

Лимитировать можно:

- Обьемы памяти:
  - Адресное пространство
  - Стек
  - RSS
- Открытые файлы
- Количество процессов
- Процессорное время работы процесса