Справочная информация по системе Linux

Алексей Полухин

2023

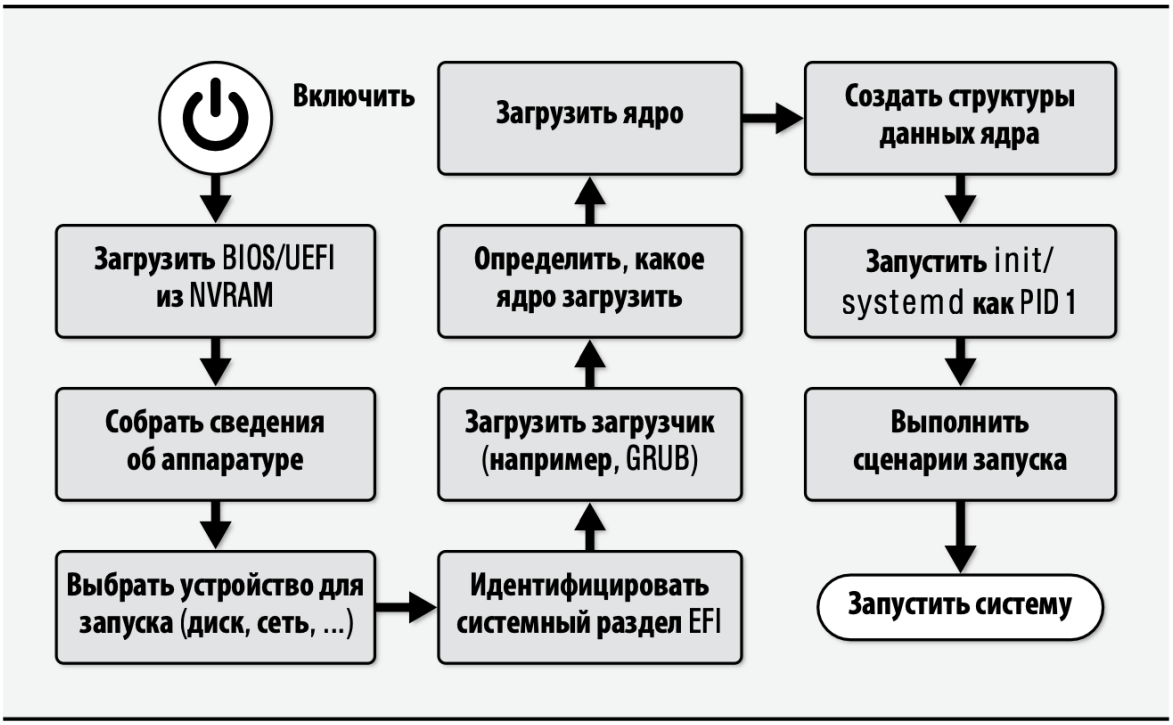

Рис. 1: Процессы загрузки Linux и Unix

1 Справочная информация. Полезные сведения

Демон в Linux – *фоновый процесс*. Программа на уровне пользователя

Киби-, меби-, гиби- байты

репозиторий — место, где хранятся и **поддерживаются** какие-то данные Примеры репозиториев: *App Store, Google Play*

2 Операционные системы

2.1 Ядро ОС

Ядро ОС – центральная часть ОС, обеспечивающая приложениям координированный доступ к ресурсам комьютера Рис. 2. В современных ОС приложение не может напрямую обратиться к ресурсам комьютера, поэтому приложения обращаются к **ядру**.

Рис. 2: Ядро ОС

Архитектура ядра операционной системы — это структура и дизайн основной части операционной системы, которая обеспечивает основные функции управления ресурсами компьютера, планирование выполнения задач, обработку прерываний и обеспечивает взаимодействие между аппаратными устройствами и пользовательскими процессами.

В ОС, основанных на ядре, приложения имеют собственные независимые окружения. То есть свои участки памяти, своё процессорное время, свой доступ к устройствам ввода и вывода.

Современные ОС имеют пространство ядра (где идёт работа с оборудованием) и пространство пользователя.

Архитектуры ядер

- Монолитное ядро (самое быстрое) — Linux
- Микроядро (самое отказоустойчивое)
- Гибридное ядро — Windows

Ближе к оборудованию — быстрее, ближе к пространству пользователя — стабильнее. (Интересная аналогия: Python, C++, assembler)

Создание ОС с ядром возможно, когда на аппаратном уровне появляются кольца защиты. (методы разграничения ресурсов компьютера)

Кольца защиты — аппаратная реализация механизма разграничения ресурсов компьютера

Интерпретатор команд — это обычное приложение. В Linux они бывают разные: *shell, bash, ksh, csh, psh ...* Он не является частью ядра ОС.

Для Linux не существует расширений файлов, они сделаны исключительно для удобства пользователя и для программ, которые взаимодействуют с этими файлами

Всё в Unix/Linux — это файл, просто поток байт

2.2 *UNIX*

Философия UNX

1. **Пишите программы, которые делают что-то одно, но хорошо**
2. Пишите программы, кототрые работают вместе
3. Пишите программы, кототрые поддерживали бы текстовые потоки, так как это универсальный интерфейс

POSIX — набор стандартов, описывающих инетерфейсы между операционной системой и прикладной прогаммой (в *UNIX*). (Средство общения с ядром ОС). Обеспечивает совместимость UNIX-подобных ОС.

Когда разработчики создают программы, используя стандарт POSIX, эти программы могут работать на разных операционных системах без необходимости изменять их код. Это облегчает перенос программного обеспечения с одной системы на другую, так как программы, сделанные в соответствии с POSIX, будут исполь-

зовать одни и те же команды и функции, доступные во всех системах, поддерживающих этот стандарт.

POSIX описывает работу в пространстве пользователя. (видимо, именно из-за этого команды в терминалах MacOS, Unix и Linux совпадают)

Команды вроде ls (список файлов и директорий) и pwd (текущая рабочая директория) являются стандартными командами, определенными в стандарте POSIX.

POSIX определяет интерфейс и поведение для командной оболочки и других системных команд в операционных системах, таких как UNIX, Linux, macOS и другие, чтобы обеспечить переносимость программ между различными системами.

Сейчас UNIX используется в серверах и мейнфреймах

Мейнфрейм (Mainframe) — это тип большого и мощного компьютера, который обычно используется в корпоративных средах для обработки больших объемов данных и критически важных бизнес-приложений. (Например, используется в центрах усправления (космическими) полётами, в банках, на биржах — в отраслях, где важна каждая *наносекунда*). (Используется в критически важных задачах, где важна скорость и бесперебойность)

На всех суперкопьютерах установлен Linux, потому что они решают сложные математические задачи, если они вдруг остановятся, ничего критического не произойдёт.

2.3 Структура каталогов в Linux

См. Рис. 3

Основные каталоги

- / — root
- /bin — Необходимые утилиты, необходимые при работе всем пользователям

(и в однопользовательском режиме)

- /boot — загрузочные файлы (файлы загрузчика, ядро, initrd, System.tap)
- /dev — основные файлы устройств
- **/etc** — общесистемные конфигурационные файлы (настройки) (настройки ОС и служб ОС)
- /home — домашние каталоги пользователей (их персональные настройки и данные)
- /lib — Основные библиотеки, необходимые для работы программ из /bin и /sbin
- /media — Каталог, где производится монтирование сменных носителей (USB, CD-ROM)
- /mnt — каталог содержит временно монтирование файловые системы
- /opt — Дополнительные программное обеспечение
- **/proc** — каталог, которые содержит информацию о всех процессах в нашей ОС
- /root — домашний каталог пользователя *root*
- /run — информация о системе с момента её загрузки (что запущено и чем это работает)
- /sbin — основные системные исполняемые файлы (основыне программы для настройки и администрирования сисетмы, *init, ifcondif, iptables*)
- /srv — данные для серисов, представляемых системой (*www* или *ftp*)

- **/sys** — содержит информацию об устройствах, драйверах и некоторых свойствах ядра
- /tmp — временные файлы
- /usr — Большинство пользовательских приложений и утилит (используемых в многопользовательском режиме)
- /var — изменяемые файлы. (файлы регистрации, временные почтовые файлы, файлы спулеров)
- /var/log — логи
- /home/username — домашний каталог пользователя

2.4 Установка ПО в Linux

Установка ПО в Linux/Unix — это просто переписывание бинарных файлов
пакет — скомпилированный бинарный файл и перечень зависимостей

1. Из исходных кодов (файлы на языке C) — **плохой способ**
    - Можно модифицировать и устанавливать без прав администртора
    - Поиск зависимостей очень долгий (как и сам процесс компиляции)
    - Нет контроля ПО. (что, какой версии, куда, когда и кто установил — нет возможности узнать)
    - Правильнее будет скопиллировать, собрать пакет и пакет установить с помощью пакетоного менеджера (будет вестись запись установленного ПО и версий)
2. Из пакетов
    - Сразу видны зависимости

- Не тратится время на компилляцию
- Просто распаковка архива и копирование файлов в нужное место ОС (если есть все зависимсти)
- Есть контроль версий ПО
- Нужны права администратора
- Пакеты создаются под определённый дистрибутив Linux

3. Из репозитория — **лучший способ**
    - Сразу виден перечень зависимостей
    - Есть контроль версий ПО
    - Как правило, все зависимости устанавливаются автоматически из репозитория
    - При установке пакетный менеджер сообщит, что нужно и сколько места это займёт
    - Репозиторий может быть локально или где-то на серверах

2.5 Создание Linux

Linus Torvalds в 1991 создал ***ядро*** операционной системы Linux. Ядро — часть ОС, которая отвечает за взаимодействие с оборудованием и предоставляет определённый интерфейс (в данном случае *POSIX*) Пользователи и администраторы не работают с самим ядром, они работают в пространстве пользователя

У Richard M. Stallman было готово окружение GNU, но не было ядра.

Проекты объединились и образовалась ОС GNU/Linx. ОС GNU/Linux — это ядро и набор программ.

Официальная версия ядра vanilla kernel *www.kernel.org/*

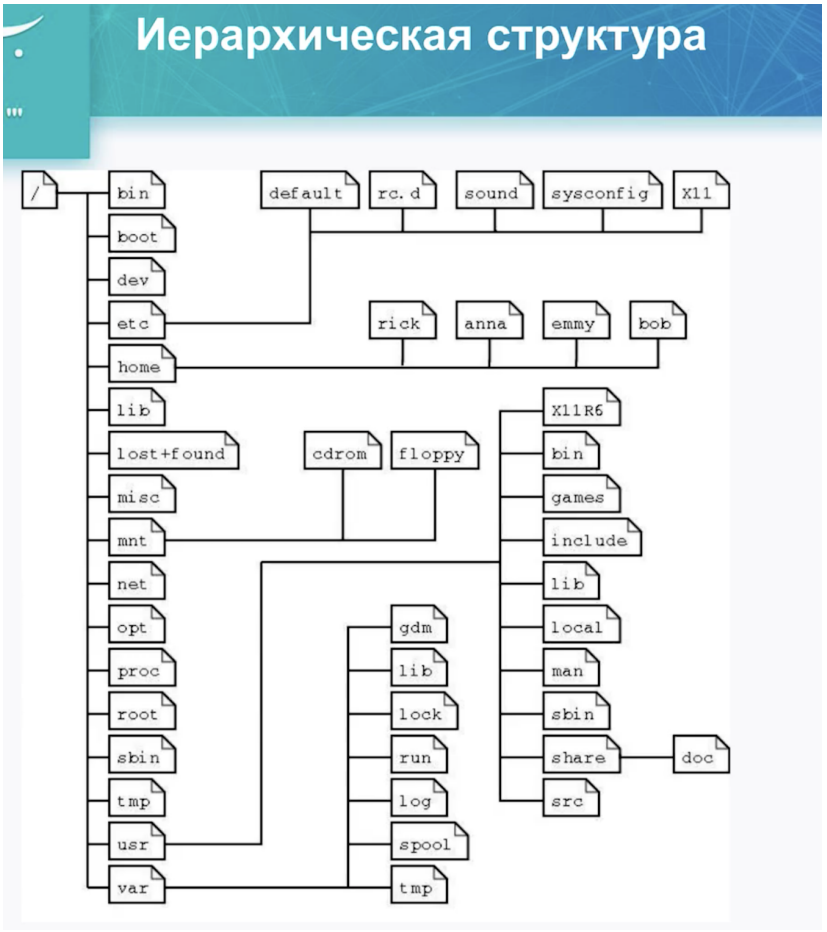

Рис. 3: Структура каталогов в Linux

Различия дистрибутив

- Разные версии ядра
- Разная структура каталогов
- Разные менеджеры пакетов

3 Полезные bash-команды

Формат:

команда *ключи* ***аргументы***

$ man -k word — *ищёт в документации ключевое слово word*

$ uname — выводит информацию о версии ядра

$ date — выводит текущую дату и время

$ ls -l — более подробный ls

(Если в первой колонке вывода $ *ls -l* стоит **-**, то это файл. А если **d** — то директория)

$ ls -la (или -l -a) — посмотреть скрытые файлы (имя начинаются с .)

$ ls -la .. — содержимое родительского каталога

$ *comand* --help — справочная информация

$ touch *existing_file* — изменить время создания файла

$ mkdir -p dir1/dir2/dir3 — рекурсивное создание директорий

Двойное нажатие TAB выдаст спсиок возможных дополнений. Кроме того, если дополнение единственное, будет дополено автоматически

$ cd — переводит в домашную директорию пользователя (аналог cd ~).

Конструкция $ cd alex эквивалентна $ cd ./alex

Маски для файлов («Регулярные выражения»)

* — любой набор любых символов

? — один любой символ

$ rm *2 — (всё, что оканчивается на 2)

$ rm file* — (всё, что начинается с file)

$ rm *.pdf — (все pdf файлы)

$ rm garbadge.*

3.1 Пример сборки кода из исходников

$ git clone https:/github.com/the-tcpdump-qroup/tcpdump.qit

$ cd tcpdump

$ ./confiqure

$ make

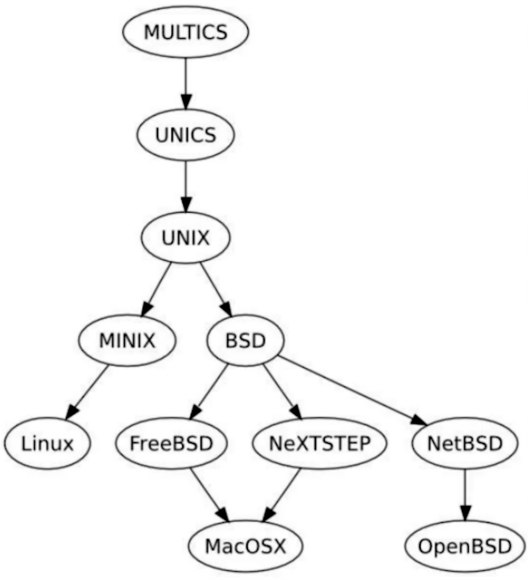

Рис. 4: Развитие операционных систем

```
$ sudo make install
```

3.2 Зависимости

Wants	Модули, которые должны быть активированы одновременно, если это возможно, но не обязательно
Requires	Строгие зависимости; отказ от каких-либо предварительных условий прекращает работу этой службы
Requisite	Аналогично *Requires*, но модуль должен быть активным
BindsTo	Аналогично *Requires*, но модуль должен быть связан еще более тесно
PartOf	Аналогично Requires, но влияет только на запуск и остановку
Conf1icts	Отрицательные зависимости; не может взаимодействовать с этими единицами

Таблица 1: Явные зависимости

4 Полезные слайды