Некоторые детали *nix экосистем

Александр Николаев

6 сентября 2017 г.

Содержание

1 Введение

1.1 О чем этот текст

Основная идея данного текста — дать широкую картину экосистемы *nix мира, показать как устроены некоторые вещи и дать возможность комфортно себя чувствовать нетехническим специалистам в относительно технических дискуссиях. Я не собираюсь углубляться в алгоритмы или детали реализации, цель, скорее — добиться достаточно понимания, чтобы не теряться.

1.2 1960-е

В 1960-х годах ЭВМ стали массово появляться в коммерческих организациях и университетах. На тот момент ЭВМ — это достаточно объемный агрегат, за котором работает много пользователей. Машинное время дорого, устройства ввода/вывода и связи медленны, поэтому отнюдь не все пользователи работают в интерактивном режиме — многие это делают пакетным образом, задавая наборы команд заранее и получая результаты с ощутимой задержкой. Таким образом, достаточно штатной ситуаций где-то до микрокомпьютерной революции 1980-х является следующая:

- В одной ЭВМ установлена одна операционная система (OS);
- В одной OS одновременно работает множество пользователей;
- Пользователи работают удаленно, не имея физического доступа к ЭВМ;
- Четко выражена роль пользователей имеющих административные привелегии и тех, у кого их нет;
- Использование пользователями ресурсов четко контролируется;
- Машинное время, память и диски дороги.

При этом, большинство OS на тот момент были написаны на ассемблере, что не позволяло их портировать на другие архитектуры или ЭВМ других производителей. OS обычно писались производителями самих ЭВМ и уровень совместимости их между собой был весьма низкий. Если техника от IBM в общем случае обладала достаточно похожими версиями OS, то сказать это о двух произвольных машинах от двух произвольных производителей в общем случае было нельзя.

1.3 Томпсон и Ричи

Кеннет Томпсон и Деннис Ричи в это время работают в Bell Labs — структурном подразделении AT&T, занимающимся разработками новых технологий. Бардак с операционными системами написанными на ассемблере их не устраивает, и они решают написать новую систему.

Параллельно с разработкой OS становится понятно, что существующих языков программирования для реализации замысла в желаемом виде не достаточно. Разработав ранее язык B и взяв его за основу, пара создает язык C и операционную систему UNIX.

Язык C оказывается весьма успешен и быстро портируется на многие архитектуры. Написанная в основном на нем операционная система UNIX требует небольших и заранее понятных изменений на ассемблере для портирования на новые архитектуры. Простота портирования UNIX позволила ему относительно быстро занять ведущую роль среди OS для коммерческого и академического применения.

1.4 Язык C

Язык C является компактным процедурным языком. Словарь его можно запомнить достаточно быстро; однако несмотря на небольшое количество зарезервированных слов, язык позволяет разработчику эффективно реализовывать любые алгоритмы. На данный момент C обычно используется в основном для системного программирования.

В эпоху своего создания, C позволил программистам писать не на ассемблере, а на гораздо более дружелюбном языке. При необходимости, код на C может вызывать инструкции ассемблера. C, таким образом, является гибким инструментом для специализированных задач. Гибкость эта, однако, требует высокого уровня понимания того, а что именно делает код и какая перед ним стоит задача — "выстрелить в свою ногу" в C неимоверно просто.

Концептуально похожие моменты присутствуют и в OS UNIX.

1.5 UNIX

К концу 1970-х годов OS UNIX становится достаточно похожей на текущие доступные нам версии. К этому моменту формируется несколько основных концепций, определяющие мировоззрение в среде UNIX:

- Каждая программа должна делать что-то одно и делать это хорошо;
- Вывод одной программы может быть вводом другой;
- Текстовые строки — базовый формат коммуникаций;
- Пользователь должен иметь возможность работать в неинтерактивном режиме;
- Ручной неквалифицированный труд нужно заменять автоматизацией.

На практике эти подходы означают, что присутствует ядро команд, позволяющих реализовать широкий спектр задач прямо в OS. Кроме того, изначальная нацеленность на неинтерактивные режимы работы пользователей создало почву для создания скриптов еще больше расширяющих возможности того, что можно сделать прямо в OS.

Распространение нового программного обеспечения в этой парадигме осуществляется путем распространения исходных кодов и дальнейшей сборки и компиляции бинарных файлов на конкретной системе. Наличие

исходных кодов предполагает, что разработчик может полностью контролировать и осознавать, что именно делает ПО.

Графические пользовательские интерфейсы во время разработки UNIX встречались достаточно редко. Операционная система позволяет работать с терминальной консоли и графический пользовательский интерфейс реализован отдельно.

1.6 Лицензирование

В 1980-е годы появляются концепции прав собственности на ПО, отличных от авторских прав. На практике это приводит к фрагментации версий OS UNIX на свободные (в основном на основе BSD), и проприетарные решения. Количество разных версий UNIX, восходящих корнями к разным общим веткам кода весьма велико и есть, как правило, либо следствия наличия отдельных производителей аппаратного обеспечения (Sun Solaris, MacOS), либо наличия конкретных задач (NetBSD, OpenBSD).

Существуют разные виды свободного ПО, в зависимости от способа лицензирования и философского подхода авторов.

1.6.1 GNU

"GNU — GNU is not UNIX" — инициатива GNU больше всего известна личностью Ричарда Столлмана, являющегося одним из наиболее известных апологетом идей свободного ПО. GNU занимает весьма последовательную позицию касательно невключения проприетарного кода и не использования бинарных файлов, исходных кодов которых не предоставлено (часто это происходит с драйверами видеокарт). В общем случае, GNU версии программного обеспечения точно являются свободными и соответствуют стандартам (или имеют режимы работы, соответствующие стандартам).

1.7 POSIX

POSIX является семейством стандартов описывающих то, что должна делать операционная система. В силу того, что на момент начала написания UNIX был основной OS, поддерживающей аппаратное обеспечение разных производителей, во многом POSIX стандарт опирается на UNIX.

Большинство *nix OS поддерживают POSIX в той или иной степени. Часто поддерживается более широкий набор параметров и свойств, чем требуется стандартом.

Набор ПО, поддерживающий POSIX может работать как OS и будет во многом совместим с существующими *nix системами.

1.8 Linux

В начале 1990-х Линус Торвальдс в Финляднии понял, что работа с существующими версиями UNIX неудобна для его целей и решил сделать набор утилит, которые позволили бы ему проще работать с той версией UNIX (Minix), что у него была под руками. Эта работа вылилась в деятельность по реализации большой части POSIX функционала и положила началу создания ядра Linux.

Linux не является какой-то одной операционной системой — присутствует некое общее ядро, на которое каждая группа разработчиков добавляет то, что считает нужным. Linux является свободным ПО, но при этом не всегда строго соответствует нормам GNU, при том, что может быть близок к ним.

1.9 MacOS, iOS

MacOS и iOS построены на Darwin, который содержит в себе элементы UNIX. Исходные коды Darwin являются открытыми. Исходные коды MacOS и iOS не доступны. Они включаются в себя как Darwin, так и проприетарные компоненты. MacOS и iOS не являются POSIX сертифициро-

ванными OS, но на практике POSIX совместимы.

1.10 OpenBSD

OpenBSD является версией UNIX нацеленной на безопасность. Весь код OS многократно проверяется. Как часть OpenBSD так же поддерживается набор инструментов openssh, который используется в других OS. OpenBSD часто используется в качестве OS, доступной из вне.

2 Работа с терминалом

2.1 Запуск команд

2.2 Ctlr последовательности

2.3 EOL, EOF

3 Базовые команды

3.1 man

3.2 Ls

3.3 pwd

3.4 cd

3.5 Ps -ax|grep

3.6 Tail

3.7 Head

3.8 Cat

3.9 Запуск с параметрами

3.10 touch

3.11 curl

3.12 wget

4 Пользователи

4.1 Кто такой пользователь

4.2 Как посмотреть пользователей

4.3 Группы

4.4 Как стать другим пользователем

4.5 SU и sudo

4.6 who, whoami

4.7 /etc/passwords/

5 Файловые системы

5.1 Что такое файловая система

5.1.1 Ext

5.1.2 ZFS

5.1.3 ReiserFS

5.2 Монтирование

5.3 df

5.4 du

5.5 NFS

5.6 Устройства

5.7 Ссылки

5.8 /dev/null

6 Файлы и папки

6.1 Файл

6.2 Папка

6.3 Права

6.4 Численная реализация прав

6.5 chmod, chown

6.6 su, sudo

6.7 cp, cp -R

6.8 rm, rmdir

7 Процессы

7.1 init

7.2 запуск процесса

7.3 ps -ax

7.4 background process

7.5 stdout, stdin, stderr

7.6 &1, &2

8 Системная информация

8.1 /etc

8.2 Стандартные папки

8.3 Переменные окружения

8.4 PATH

8.5 Демоны

9 shells

9.1 Что такое shell

9.2 Когда какой shell вызывается

9.3 bash

9.4 sh

9.5 csh, ksh, zsh &c

10 Pipes

10.1 |

10.2 compex pipes

11 Редакторы и работа с текстом

11.1 ed

11.2 sed

11.3 awk

11.4 pico

11.5 vi

11.6 emacs

12 Распространение ПО, разрешение зависимостей

12.1 Набор по умолчанию

12.2 Библиотеки

12.3 Исторический подход: исходные коды и локальная сборка

12.4 apt-get

12.5 yum

12.6 brew

13 cron

14 awk, sed, find

14.1 find

14.2 awk

14.3 sed

15 shell scripting

A Сеть

A.1 Что такое Интернет

A.2 TCP, UDP

A.3 IP, IPv4, IPv6

A.4 loopback

A.5 MTU

A.6 ifconfig

A.7 Proxy

A.8 NAT

B X server

B.1 graphic mode

B.2 server + client

C ssh

C.1 ssh

C.2 Как работает

C.3 параметры

C.4 public private key

C.5 прокидывание X сервера

D ftp

D.1 binary/text

D.2 Пример

D.3 sftp

E Hashes

E.1 Hash

E.2 md5

F Архитектурные соображения

G docker

G.1 Что есть docker

G.2 Плюсы и минусы

G.3 когда уместно

G.4 Примеры использования

H Виртуализация

H.1 Определение

H.2 Зачем используется на практике

H.3 Часто встречающиеся механизмы реализации

H.4 VirtualBox

I Git

I.1 git

I.2 git + ssh

I.3 git user

I.4 .gitignore

I.5 commit

I.6 push/pull

I.7 rebase

I.8 git console

J Часовые пояса

К TEX

К.1 Дональд Кнут

К.2 Metafont

К.3 TEX

К.4 LATEX