Operating System Overview (part 2)

Operating Systems

João Vicente Ferreira Lima

Universidade Federal de Santa Maria
jvlima@inf.ufsm.br
http://www.inf.ufsm.br/~jvlima

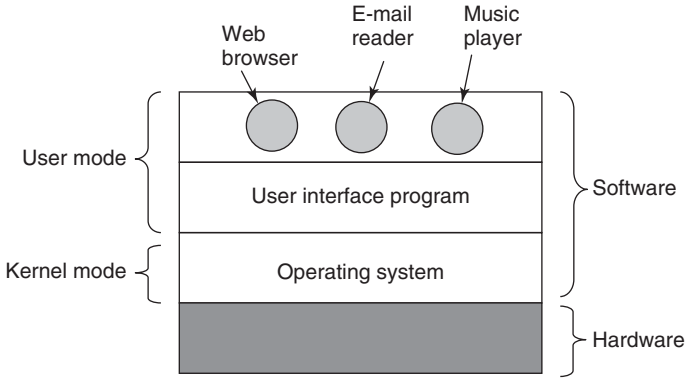
2020/2

Outline

Operating System

The term *operating system* is commonly used with two meanings:

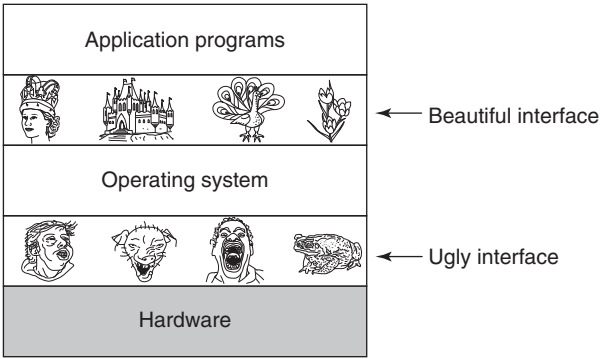
1. To denote the entire package: the central software such as command-line interpreters, GUI, file utilities, and editors.
2. To the central software that manages and allocates computer resources.



Operating System

The term *operating system* is commonly used with two meanings:

1. To denote the entire package: the central software such as command-line interpreters, GUI, file utilities, and editors.
2. To the central software that manages and allocates computer resources.

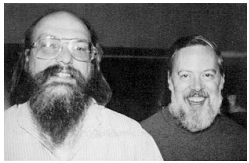


Outline

Unix history

UNIX was written in assembly by Ken Thompson at Bell Labs (AT&T) in 1969 for a Digital PDP-7 minicomputer. It took several ideias from MULTICS such as:

- tree-structured file system.
- separate program for interpreting commands (shell).
- notion of files as unstructured streams of bytes.



Denis Ritchie designed and implemented C language between 1969-1973 at Bell Labs. By 1973, UNIX was almost totally written in C.

C for system programming

In the 70s, widely used languages were designed to other purposes:

- FORTRAN for mathematical tasks
- COBOL for commercial systems

PDP-7 minicomputer

Unix popularity

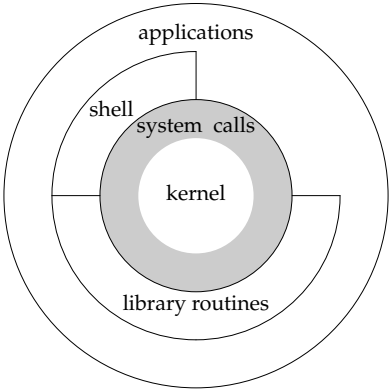
At this time, UNIX was popular in universities. The terms of AT&T' agreement with the US government prevented it from selling software. The UNIX university distributions included documentation and the kernel source code (10k lines).

BSD and System V

- BSD or *Berkeley Software Distribution* was a UNIX version created when Thompson spent the 1975/1976 academic year as a visiting professor at Berkeley. It included *C shell*, *vi*, *sendmail*, a Pascal compiler, etc.
- System V was a comercial version of UNIX when AT&T was permitted to market UNIX.

Other implementations: SunOS or Solaris, Ultrix and OSF/1, IBM's AIX, HP-UX, NeXT's NeXTEP, A/UX for the Apple Macintosh, XENIX.

Unix architecture



- The kernel is relatively small, and controls the hardware resources.
- The interface to the kernel is the system calls. Libraries are built on top of it, but applications are free to use both.
- The shell is a special application that provides an interface for running other applications.

Outline

The BSDs

It is worth noting that another free UNIX was already available during the early 1990s. Bill and Lynne Jolitz had developed a port of the already mature BSD system (Net/2 from AT&T) for the x86-32, known as 386/BSD. Alternatives from the 386/BSD were:

- NetBSD, FreeBSD, OpenBSD, DragonFly BSD (FreeBSD 4.x)

AT&T and Berkeley lawsuit

In April 1992, USL (AT&T) filed suit against BSDi (Berkeley) and the University of California claiming that the product was still encumbered by proprietary USL source code and trace secrets. A countersuit by the university ensued since USL had not given credit for the BSD code in System V.

The veredict: Berkeley was required to remove 3 of the 18k files from Net/2 release, make some minor changes, and add USL copyright, resulting in the 4.4BSD-Lite. The terms of legal settlement required BSDi, FreeBSD and NetBSD to replace their Net/2 base.

Outline

The GNU project

Richard Stallman started the GNU project to develop an entire, freely available, UNIX-like system, consisting of software+kernel. Initially they produced a wide range of other programs, but did not produce a working UNIX kernel.

GNU software

GNU C compiler, *bash* shell, GNU C library.

GNU/HURD

GNU/HURD project is a kernel based on the Mach microkernel, but never released.

Outline

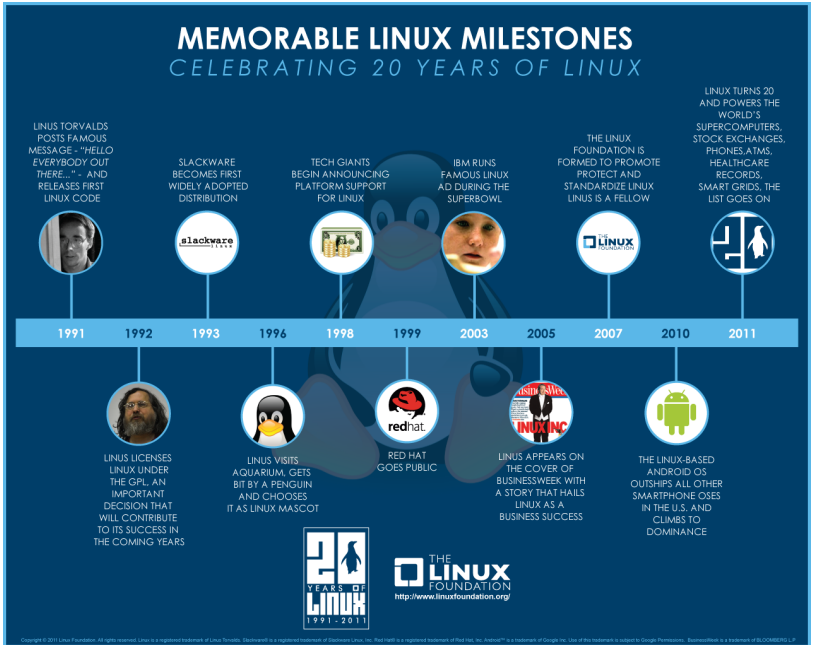
The Linux kernel

In 1991, Linus Torvalds was inspired to write an OS for his Intel 80396 PC. In the course of his studies, he had come into contact with Minix, a small UNIX-like operating system developed by Andrew Tanenbaum.
The first version was released by March 1994.

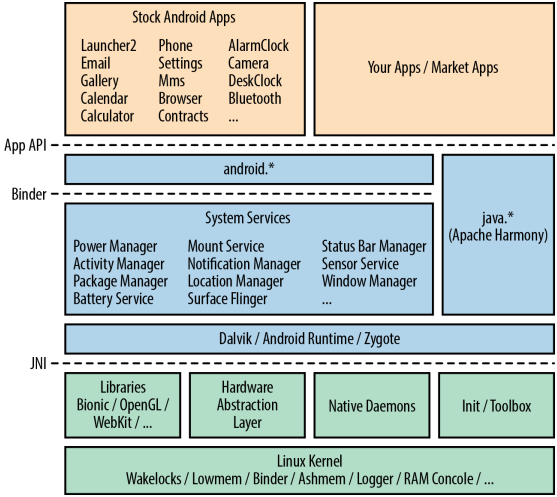
Linux distributions

- Slackware in 1993
- Debian soon after
- SUSE and Red Hat followed
- Ubuntu in 2004

IBM commercial: https://www.youtube.com/watch?v=x7ozaFbqg00


MEMORABLE LINUX MILESTONES
CELEBRATING 20 YEARS OF LINUX
LINUS TORVALDS POSTS FAMOUS MESSAGE - "HELLO EVERYBODY OUT THERE..." - AND RELEASES FIRST LINUX CODE
SLACKWARE BECOMES FIRST WIDELY ADOPTED DISTRIBUTION
TECH GIANTS BEGIN ANNOUNCING PLATFORM SUPPORT FOR LINUX
IBM RUNS FAMOUS LINUX AD DURING THE SUPERBOWL
THE LINUX FOUNDATION IS FORMED TO PROMOTE PROTECT AND STANDARDIZE LINUX LINUS IS A FELLOW
LINUX TURNS 20 AND POWERS THE WORLD'S SUPERCOMPUTERS, STOCK EXCHANGES, PHONES, ATMS, HEALTHCARE RECORDS, SMART GRIDS, THE LIST GOES ON
1991
1992
1993
1996
1998
1999
2003
2005
2007
2010
2011
LINUS LICENSES LINUX UNDER THE GPL, AN IMPORTANT DECISION THAT WILL CONTRIBUTE TO ITS SUCCESS IN THE COMING YEARS
LINUS VISITS AQUARIUM, GETS BIT BY A PENGUIN AND CHOOSES IT AS LINUX MASCOT
RED HAT GOES PUBLIC
LINUS APPEARS ON THE COVER OF BUSINESSWEEK WITH A STORY THAT HAILS LINUX AS A BUSINESS SUCCESS
THE LINUX-BASED ANDROID OS OUTSHIPS ALL OTHER SMARTPHONE OSES IN THE U.S. AND CLIMBS TO DOMINANCE
20 YEARS OF LINUX 1991-2011
THE LINUX FOUNDATION
http://www.linuxfoundation.org/

Android


Stock Android Apps
Launcher2
Email
Gallery
Calendar
Calculator
Phone
Settings
Mms
Browser
Contracts
AlarmClock
Camera
DeskClock
Bluetooth
...
Your Apps / Market Apps
App API
android.*
Binder
System Services
Power Manager
Activity Manager
Package Manager
Battery Service
Mount Service
Notification Manager
Location Manager
Surface Flinger
Status Bar Manager
Sensor Service
Window Manager
...
java.*
(Apache Harmony)
Dalvik / Android Runtime / Zygote
JNI
Libraries
Bionic / OpenGL /
WebKit / ...
Hardware
Abstraction
Layer
Native Daemons
Init / Toolbox
Linux Kernel
Wakelocks / Lowmem / Binder / Ashmem / Logger / RAM Concole / ...

Outline

The Kernel role

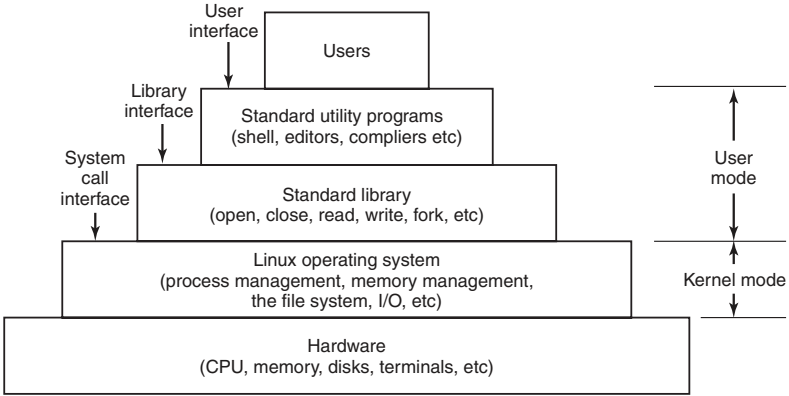
Tasks performed by the kernel:

- Process scheduling: Linux is a *preemptive multitasking* SO.
- Memory management: virtual memory management.
- File system: file I/O.
- Creation and termination of processes: allocating resources.
- Access to devices: interface between programs and devices.
- Networking: transmit and receive network messages on behalf of user processes.
- System call API: entry points to request various tasks.

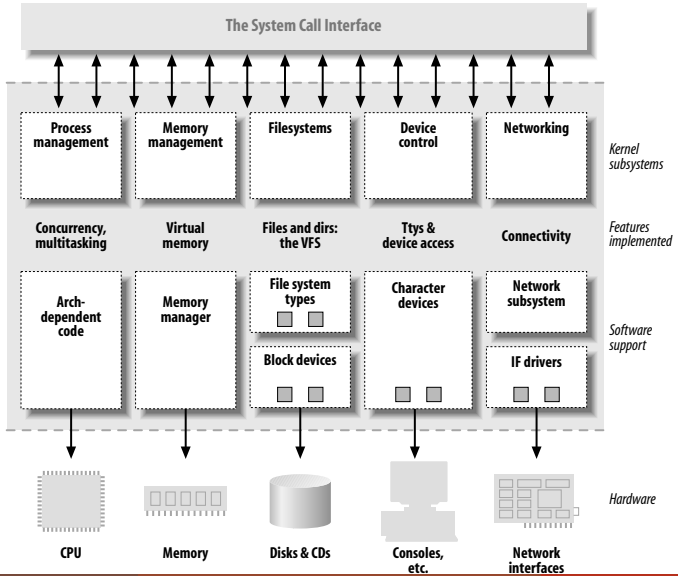
Kernel mode vs user mode

Tipically there are at least two differente modes: *user mode* and *kernel mode* (or *supervisor mode*). When running in user mode, the CPU can access only memory that is marked as begin in user space.
Certain operations can be performed only in kernel mode.

Linux system layers



Linux kernel layers



Outline

The Shell

The *shell* or *command interpreter* is a special-purpose program designed to read commands and execute then. On some OS the shell is an integral part of the kernel, on UNIX is a user process. Examples:

- Bourne shell (sh): the oldest of the widely used shells, written by Steve Bourne.
- C shell (csh): similar to C on control-flow constructs.
- Korn shell (ksh): successor of the Bourne shell.
- Bourne again shell (bash): GNU Bourne shell with features from C and Korn shells.

The shells are designed not only for interactive use, but also for the interpretation of *shell scripts* (text with commands). For this purpose, each has features such as: variables, loop and conditional statements, I/O commands, and functions.

Outline

Users and groups

Users: every user has a unique *login name* (username) and *user ID* (UID). There are defined in the system *password file* `/etc/passwd` with:

- Group ID: the GID of first group the user is a member.
- Home directory: initial user directory.
- Login shell: name of the program to interpret user commands.

The password is recorded in encrypted form in the *shadow password file*.

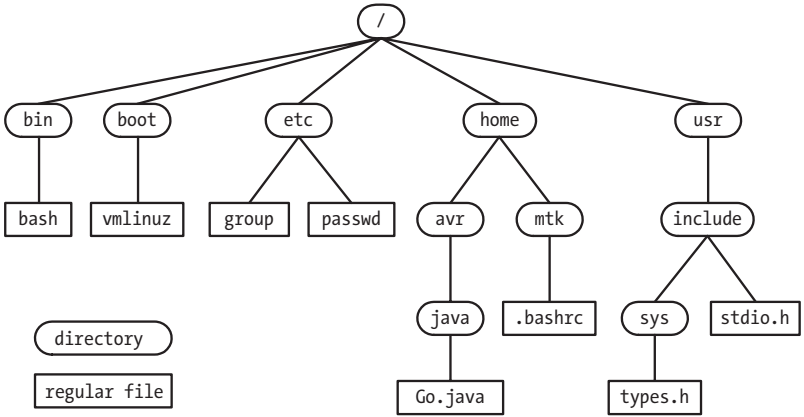
Groups: users are organized in *groups* The *group file* `/etc/group` has:

- Group name: unique group name.
- Group ID (GID): numeric ID.
- User list: list of login names of users who are members.

Superuser: or *root* has special privileges within the system.

Outline

Files and directories



Files and directories

File types: each file is marked with a *type*. Ordinary files are *regular* or *plain* files. Other types are devices, pipes, sockets, directories, and symbolic links.

Directories: a *directory* is a special file whose contents take the form of a table of filenames. Each directory entry is a *link* or *hard link*. Every directory has at least two entries:

- . (dot) which is a link to the directory itself
- .. (dot-dot) which is a link to its *parent directory*.

Symbolic links: while a hard link points to the data itself, a *soft link* only provides the text of an *actual* file name.

Files and directories

Filenames: On most Linux systems, filenames can be up to 255 characters long. We should avoid non-portable characters in filesnames (`[-._a-zA-Z0-9]`).

Pathname: is a string with an optional initial slash (`/`) followed by a series of filenames separated by slashes.

- Absolute pathname: begin with a slash (`/`). Ex: `/usr/include`.
- Relative pathname: relative to a process's current working directory. Ex: `../bin/exec`.

File ownership and permissions

Each file has an associated UID and GID that define its owner and group's owner. The system divides users into three categories of file access:

- owner (*user*) of the file
- group users who are members of the group
- other the rest of the world

Three permission bits may be set for each user:

- read
- write
- execute

In directories, these permissions have different meanings:

- read allows to list the contents (filenames)
- write add/delete/change contents
- execute (or *search*) access to files within the directory

File ownership and permissions

Binary	Symbolic	Allowed file accesses
111000000	rwx------	Owner can read, write, and execute
111111000	rwxrwx---	Owner and group can read, write, and execute
110100000	rw-r-----	Owner can read and write; group can read
110100100	rw-r--r--	Owner can read and write; all others can read
111101101	rwxr-xr-x	Owner can do everything, rest can read and execute
000000000	---------	Nobody has any access
000000111	------rwx	Only outsiders have access (strange, but legal)

File I/O Model

One of the features of UNIX systems is the concept of *universality of I/O*, i.e, system calls perform I/O on all types of files, including devices.
File descriptors - the I/O system calls refer to open files by a *file descriptor*, a non-negative integer. Normally, a process inherits three open file descriptiors when it is started by the shell:

- 0 (*standard input*) from it takes the input.
- 1 (*standard output*) to which the process writes its output.
- 2 (*standard error*) to which the process writes error messages.

File descriptions

```
char c = fgetc( 0 );
fprintf( 1 , "%c\n", c);
```

The *stdio* library

```
char c = fgetc(stdin);
fprintf(stdout, "%c\n", c);
```

Outline

System Calls

The interface between OS and programs. Most calls are performed by system libraries (`libc`).

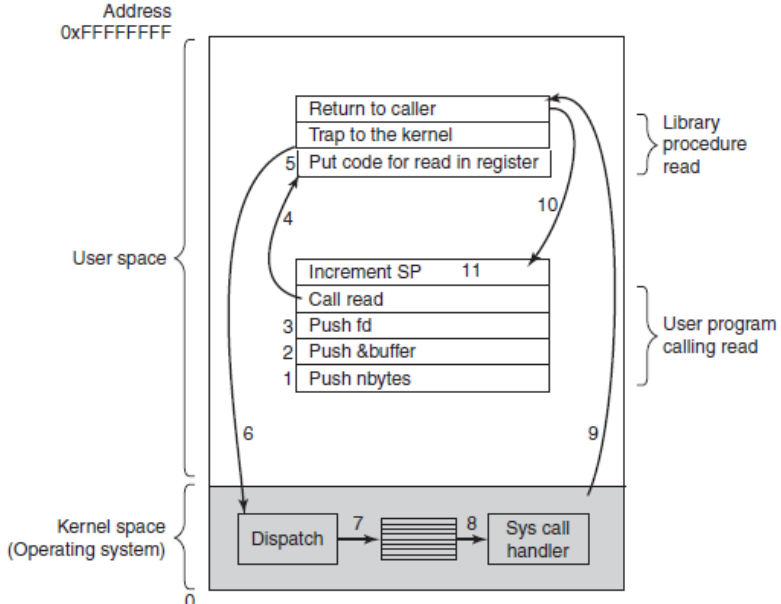
File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

Call to read()


Address
0xFFFFFFFF
User space
Kernel space
(Operating system)
0
Return to caller
Trap to the kernel
5 Put code for read in register
4
10
Increment SP 11
Call read
3 Push fd
2 Push &buffer
1 Push nbytes
6
9
Dispatch
7
8
Sys call
handler
Library
procedure
read
User program
calling read

Outline

Process definition

A process is a instance of an executing program. The kernel:

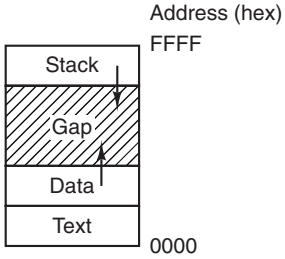
1. loads the code into virtual memory
2. allocates space for program variables
3. sets up kernel bookkeeping data structures to record various information)

Process memory layout

A process is divided into the following parts, known as *segments*:

- text - the intructions of the program.
- data - the static variables.
- heap - an area from which programs dynamically allocate memory.
- stack - piece of memory that grows and shrinks as functions are called and return, used to allocate storage for local variables and function call linkage information.

Process memory layout



Process creation

The `fork()` system call creates a new process. The process that calls `fork()` is the *parent process*, and the new *child process*. Additional steps:

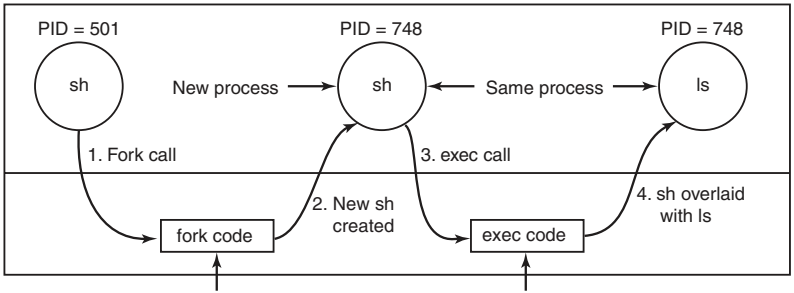
- the kernel creates the child by making a duplicate of the parent
- the child inherits copies of the parent's data, stack, and heap segments.

The child process goes on either to execute a different set of functions, or to use the `execve()` system call to load and execute a entirely new program.

Process identifier

Each process has a unique *process identifier* (PID), and a *parent process identifier* (PPID).

A ls from the shell



Process tree

```
init-+-acpid
     |-atd
     |-cron
     |-cups-browsed
     |-cupsd
     |-exim4
     |-6*[getty]
     |-irqbalance
     |-portmap
     |-redis-server---2*[{redis-server}]
     |-rsyslogd-+-{in:imklog}
     |          |-{in:imuxsock}
     |          `-{rs:main Q:Reg}
     |-screen---bash
     |-sshd-+-sshd---sshd---bash---pstree
     |      `-2*[sshd---sshd---bash]
```

Special-purpose processes

The init process - when booting the system, the kernel creates a special process called *init*, the parent of all processes, which is derived from the program `/sbin/init`. All programs from the system are created either by *init* or by one of its descendants.
The *init* process can't be killed, and terminates only when the system is shut down.

Daemon processes - a *daemon* is a special-purpose process that is created and handled by the system, but:

- It is long-lived, in general started at system boot and remais in existence until the system is shut down.
- It runs in background, and has no controlling terminal from which it can read input or to which it can write output.

Outline

FreeBSD overview

FreeBSD is a 4.4BSD-Lite based operating system for Intel and AMD64 computers. Software can be installed using:

- pkg is the binary package management for FreeBSD.
- Ports Collection which is a set of `Makefiles`, patches, and description files stored in `/usr/ports`.

Systems using FreeBSD

- Playstation 4 - Sony's PlayStation 4 Is Running Modified FreeBSD 9
- Apple OS X and some of iOS
- Apache, Whatsapp, Netflix, etc