Programming in Shell 1

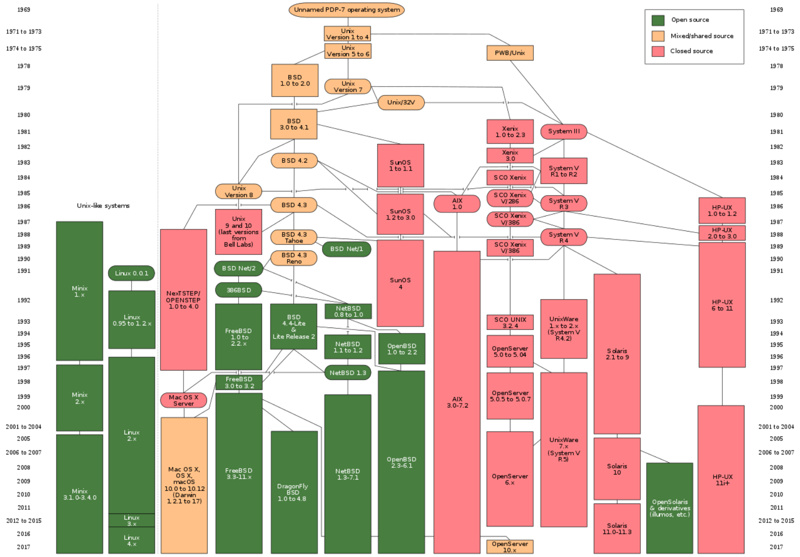
Introduction to Unix like operating systems

Jan Trdlička

Czech Technical University in Prague, Faculty of Information Technology
Department of Computer Systems

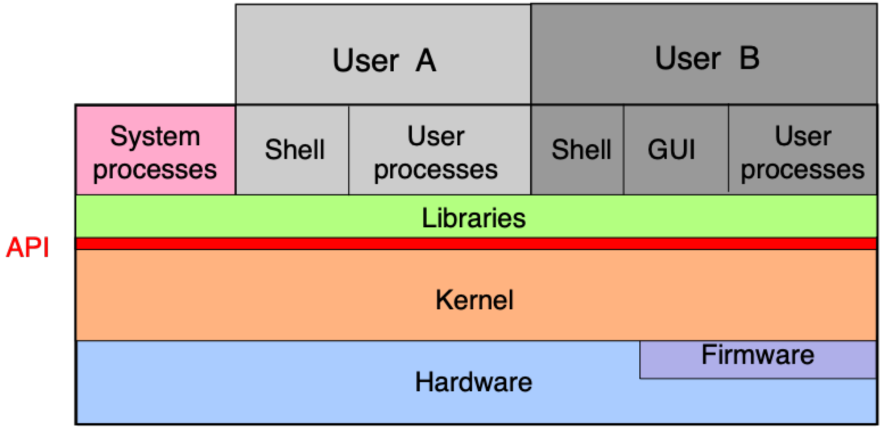
Content

Unix – history



- Source: Wikipedia

Unix – architecture



Unix – architecture

- **Hardware (HW)**
  - Physical resources: CPU, RAM, bus, disk, network card, ...
  - Firmware (software for hardware testing, kernel loading,...): BIOS, ...
- **Operating system (OS)**
  - Kernel and drivers (basic part of OS)
    - Logical resources: user, process, file, permissions, ...
    - Resource management
    - Application program interface
- **Processes**
  - Abstraction of the running program/application
  - OS processes
    - Graphical user interface (GUI): GNOME, KDE,...
    - Command-line interface (CLI): shell
    - Command line tools (commands) and other applications
  - Other suppliers' processes: web browser, graphic editor, design tools, ...
- **Users/user account**
  - Abstraction of physical user for which some attributes are defined
    - User name + password
    - User ID, member of some groups, home directory, login shell, ...

Unix like operating systems

- **GNU/Linux**
  - Linux kernel
  - GNU tools and other tools
  - Distributions: OpenSUSE, Red Hat, Debian, ...
- **Oracle Solaris**
  - SunOS kernel + tools (UNIX System V Release 4 + BSD)
  - GNU tools can be added (see /usr/gnu/bin on fray1.fit.cvut.cz)
- **macOS**
  - macOS kernel + tools (UNIX 3 + BSD)
  - GNU tools can be added by Homebrew

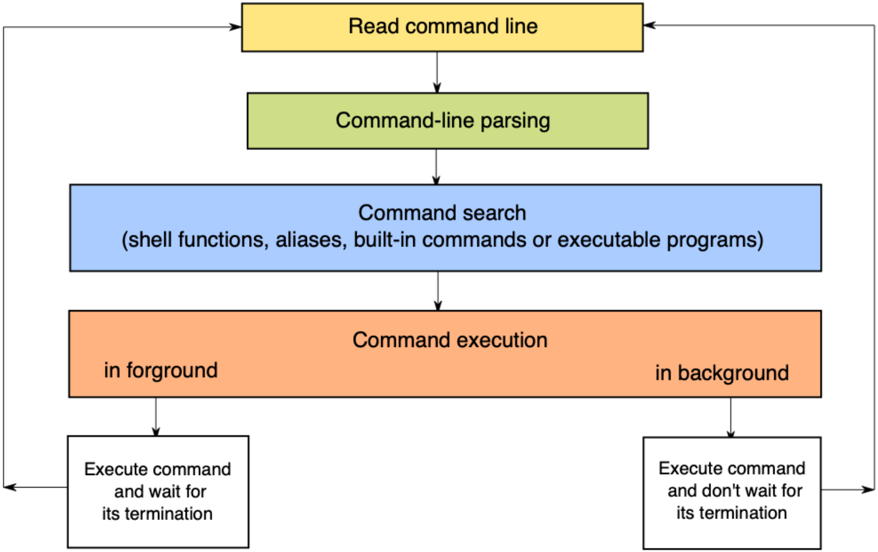
UNIX - properties

- Portable
  - 90% of kernel is written in C.
- Multi-user
- Multitasking, time-sharing
- Multithreading
- Symmetric Multi Processing (SMP)
- CLI
- IO redirection
- Hierarchical FS
- TCP/IP networking, NFS,. . .
- GUI
  - X-Windows
  - Window managers - CDE, GNOME, KDE,. . .

Shell – command interpreter

- Interface between user and kernel.
- Environment setting
  - Shell variables can define application behaviour.
- CLI
  - Command-line parsing (e.g. find and replace special symbols)
  - Command execution.
- Shell script
  - Shell executes commands from file (scripts).
  - Script = Unix commands + control structures (e.g. loops, if/else. . .)

Command-line parsing



Read command line
Command-line parsing
Command search
(shell functions, aliases, built-in commands or executable programs)
Command execution
in forground
in background
Execute command and wait for its termination
Execute command and don't wait for its termination

Command line syntax

- **Variables**
  - <prompt> <variable_name>=<value>
  - <prompt>
    - Prompt is printed by shell.
    - Value of prompt is defined by the shell variable PS1.
  - <variable_name>
    - Variable name is identifier.
    - No spaces around symbol =.
    - Shell assigns the value to the variable.
  - <value>
    - By default it is string.

Command line syntax

- **Simple commands**
  - <prompt> <command_name> <options> <arguments>
  - <command_name>
    - It defines which program will be executed (which).
    - It can be only name or path to the file (relative/absolute).
  - <options>
    - They can modify the behavior of command (how).
  - <arguments>
    - They specify the input date (what).
- Command name, options and arguments are available
  - in script by variables $#, $0, $1, $2, ...
  - in C program by variables argc , argv[0], arg[1], ...

Examples

- Little bit more complicated?

```
$> echo PID FD EXEC FILENAME; PID=$(pgrep ''); pfiles $PID | awk
'BEGIN { fd=-1; }  /^[0-9]/ { if (fd>=0) { print pid, fd, exec;
fd=-1; }; pid=substr($1,0,length($1)-1); exec=$2; } /^ *[0-9]*:
/ { if (fd>=0) { print pid, fd, exec; fd=-1; }; if
($2=="S_IFREG") { fd=substr($1,0,length($1)-1); } } /^ *\//
{ fd=-1; }' | while read pid fd exec; do echo $pid $fd $exec
$(echo 0t$pid ::pid2proc \| ::fd $fd \| ::print file_t f_vnode
\| ::vnode2path | mdb -k 2>/dev/null); done
```