Introduction to Linux

Prof. Jinkyu Jeong(jinkyu@skku.edu)
TA – Seokha Shin(seokha.shin@csl.skku.edu)
TA – Jinhong Kim(jinhong.kim@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
What is OS? (1)

**Application side**

- Provides the program execution environment
  - Hides the messy details which must be performed
  - Make machine easy to use

- Provides an abstract view of the underlying system
  - Processors -> Processes, Threads
  - Memory -> Virtual memory
  - I/O devices -> Files
What is OS? (2)

- **System side**
  - OS is a *resource manager*
    - Sharing
    - Protection
    - Fairness
    - Performance
  - What is *resource*?
    - Hardware
      » CPU, Memory, I/O devices
    - Software
      » Queues, ...
    - Miscellaneous
      » Energy, Power, ...
Unix (1)

- **History and motivation**
  - Originally developed at AT&T Bell Labs for internal use in the early 1970s
  - Borrowed best ideas from other OS’s
  - Unix is designed so that users can extend the functionality – to build new tools easily and efficiently

- **Why Unix?**
  - Used in many scientific and industrial settings
  - Huge number of free and well-written software programs
  - Many important OS concepts are developed on Unix.
Unix (2)

Unix is
- Interactive
- Time-sharing
- Multi-tasking
- Multi-user

Flavors of Unix
- System V (AT&T -> USL -> Novell -> SCO -> Caldera -> SCO)
- BSD (UC Berkeley)
- SunOS, Solaris (Sun)
- IRIX (SGI), AIX (IBM), HP-UX (HP), Mac OS X (Apple)
- Linux, FreeBSD, NetBSD, and etc.
Linux

- Open-source development began in 1991
- First released by Linus Torvalds

**Linux kernel**
- The core of Linux system
- Thousands of contributors
- Supervised by Linus and other maintainers

**Distribution**
- A collection of software based around Linux kernel
- Red Hat, Fedora, Debian, Ubuntu, Android, ...
OS Internals (1)

User Application

C Library (libc)

System Call Interface

Kernel

Arch-dependent kernel code

Hardware Platform

User space

Kernel space
OS Internals (2)

- A software between applications and hardware
  - Highly-concurrent
  - Event-driven

- What kind of events?
  - System calls
  - Interrupts
User Interfaces

- The space where we interact with machines

- **Command-line interface (CLI)**
  - Command interpreter
  - Difficult to learn
  - Called as “shell”

- **Graphical user interface (GUI)**
  - KDE, Gnome, Unity, Xfce, ...

- **Touch user interface**
  - Smartphones, tablets
Shell (1)

- Executing programs on a shell
  
  `$ command [options] [arguments]

  • [$ 1s] and [$ 1s –a1] show different results
  • All commands, options, arguments are case-sensitive

- Shells execute commands by means of *processes*
  
  • An instance of a program in execution
Shell (2)

- **A shell allows three types of commands**
  - An internal shell command (built-in command)
  - An executable file that contains object code
  - An executable file that contains a sequence of shell command lines (shell script)

- **There are two families of shells**
  - One based on "Bourne shell" (sh)
    - We will use "Bourne again shell" (bash) for the course
  - The other based on "C shell" (csh)
File System Overview (1)

- A Unix file is a sequence of bytes
  - Collection of related information defined by its creator
  - Unstructured sequence of bytes

- File system
  - Consist of two distinct parts:
    - A collection of files
    - A directory structure
  - It provides the mechanism for on-line storage and access to file contents
File System Overview (2)

- **Features of Unix file system**
  - A hierarchical structure
  - It allows dynamic growth of files
  - The ability to create and delete files
  - The protection of the file data
  - Unix treats the peripheral devices as files

- “Everything is a file” in Unix
  - Documents, directories, hard-drives, network sockets, keyboards, printers are stream of bytes exposed through the file system namespace
File System Overview (3)

- Hierarchical, tree-like structure
  - Root
  - Non-leaf nodes
    - Directories
  - Leaf nodes
    - Directories
    - Regular files or special device files
File System Overview (4)

*http://www.linuxplanet.com/linuxplanet/tutorials/6666/1*
File System Overview (5)

- **Root directory ["/"]**
  - The top-most directory in a hierarchy

- **Home directory ["~"]**
  - A special directory for a user
  - It contain the user’s files; including texts, musics, videos, or configuration files

- **(Current) Working directory**
  - Each process has associated with it a directory
  - The directory where a user currently located
File System Overview (6)

- **/bin**
  - Contains certain fundamental utilities
- **/dev**
  - Essential devices
- **/etc**
  - Host-specific system-wide configuration files
- **/tmp**
  - A place for temporary files
- **/var**
  - A place for files that may change often
Path

- The general form of the name of a file or a directory

- Delimiting characters ["/"]
  - Represent each directory in path expressed in string

- Absolute path (full path)
  - A path points a location regardless of present working directory
    - $ cat /home/sanghoon/textfile
    - $ cat ~/textfile

- Relative path
  - A path relative to the working directory of the user
    - $ cat textfile [if cwd is "/home/sanghoon"]
File Permission

- Every files have a set of permissions

Ownership
  - User/owner
    - The person who owns/created the file.
  - Group
    - Unix allows for the creation of groups
  - Others
    - Everyone else in the world that has access to that computer

Permission for Access

- Read (4)
- Write (2)
- eXecute (1)
Contents

- Basic commands
- Basic C coding
- Basic File I/O
Basic commands (1)

- **man**
  - Display the manual page
  - Display a manual of a program or a function

$ man qsort

$ man man (manual for manual page)
Basic commands (2)

- **ls**
  - List files
  
  ```
  $ ls
  $ ls
  $ ls -al /etc
  $ ll
  ```

- **ps**
  - List process
  
  ```
  $ ps
  $ ps -ef
  $ man ps
  ```
Basic commands (3)

- **pwd**
  - Print working directory

- **cd**
  - Change working directory
    - $ cd ..
    - $ cd /proc
    - $ cd ~
Basic commands (4)

- **echo**
  - Display a line of text
  
  $ echo "Hello?"

- **printf**
  - Print a formatted line of text
  
  $ printf "%s\n" Hello?

- **cat**
  - Displaying files
  
  $ cat /etc/issue

- **more / less**
Basic commands (5)

- **mkdir / rmdir**
  - Make / remove a directory
  - $ mkdir dir1

- **mv**
  - Move or rename files
  - $ mv dir1/ dir2/

- **cp**
  - Copy files

- **rm**
  - Remove files
Basic commands (6)

- **date**
  - Print or set the system date and time

- **grep**
  - Searching files for a specified expression
  
  $ grep [expression] [files]
  $ grep root /etc/passwd
Basic commands (7)

- **chmod**
  - Change the permissions on a file or directory

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>user + to add a permission</td>
</tr>
<tr>
<td>g</td>
<td>group - to remove a permission</td>
</tr>
<tr>
<td>o</td>
<td>other = to assign a permission explicitly</td>
</tr>
<tr>
<td>r</td>
<td>read (4)</td>
</tr>
<tr>
<td>w</td>
<td>write (2)</td>
</tr>
<tr>
<td>x</td>
<td>execute (1) access (for directories)</td>
</tr>
</tbody>
</table>

```
$ chmod u=rw file1
$ chmod u+x,g+w,o-r file2

$ ls -l swex2/
$ chmod 750 swex2/
$ ls -l swex2/
```
Basic commands (8)

- **diff [file1] [file2]**
  - Reports line-by-line differences between file1 and file2
Development tools

- **vi[m]**
  - A text editor for programmers
  - $ vim [file_name]
  - Create (if not exist) or open a file 'file_name'
  - [http://csl.skku.edu/SSE2033F18/Resources](http://csl.skku.edu/SSE2033F18/Resources)
  - $ vim hello.c

- **gcc**
  - GNU compiler collection
  - $ gcc -o hello hello.c
  - $ ./hello
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}

Exercise (1)

**Lab exercise #1:**
- Make "sse" directory on your home directory
- Create hello.c on the directory
- Compile it
- Run the program
- Remove "sse" directory

**Lab exercise #2:**
- With hello.c file, make shell script that run exercise #1 automatically.
Basic File I/O (1)

- **Opening a file**
  - int fd = open("path", flags)

- **Read a character from a file**
  - read(fd, &c, 1)

- **Write a character to a file**
  - write(fd, &c, 1)

- **Closing a file**
  - close(fd)
Lab exercise #3:
- Let’s make xcp utilities
- xcp copies contents of a file into a new file
- Basically, executing xcp will be same as executing cp without any options, respectively.

Your job is to make xcp by using system calls provided by Linux.