Log Book Week 1

Jose Devian Hibono
1706039603
System Programming - C

System Programming and Application Programming

System Programming is a process to produce software that interacts with the machine's hardware and operating system to provide services to the other hardware and software. This uses low-level language. E.g.: USB Device Driver, Printer Driver

Application Programming is a process to produce software that provides services to the user. This process uses high-level language. e.g.: Microsoft Word, Google Chrome.

Scripting vs Programming

Scripting is used for the backend system to access hardware. Whereas programming usually used for building a user interface programs.

Kernel and User Mode

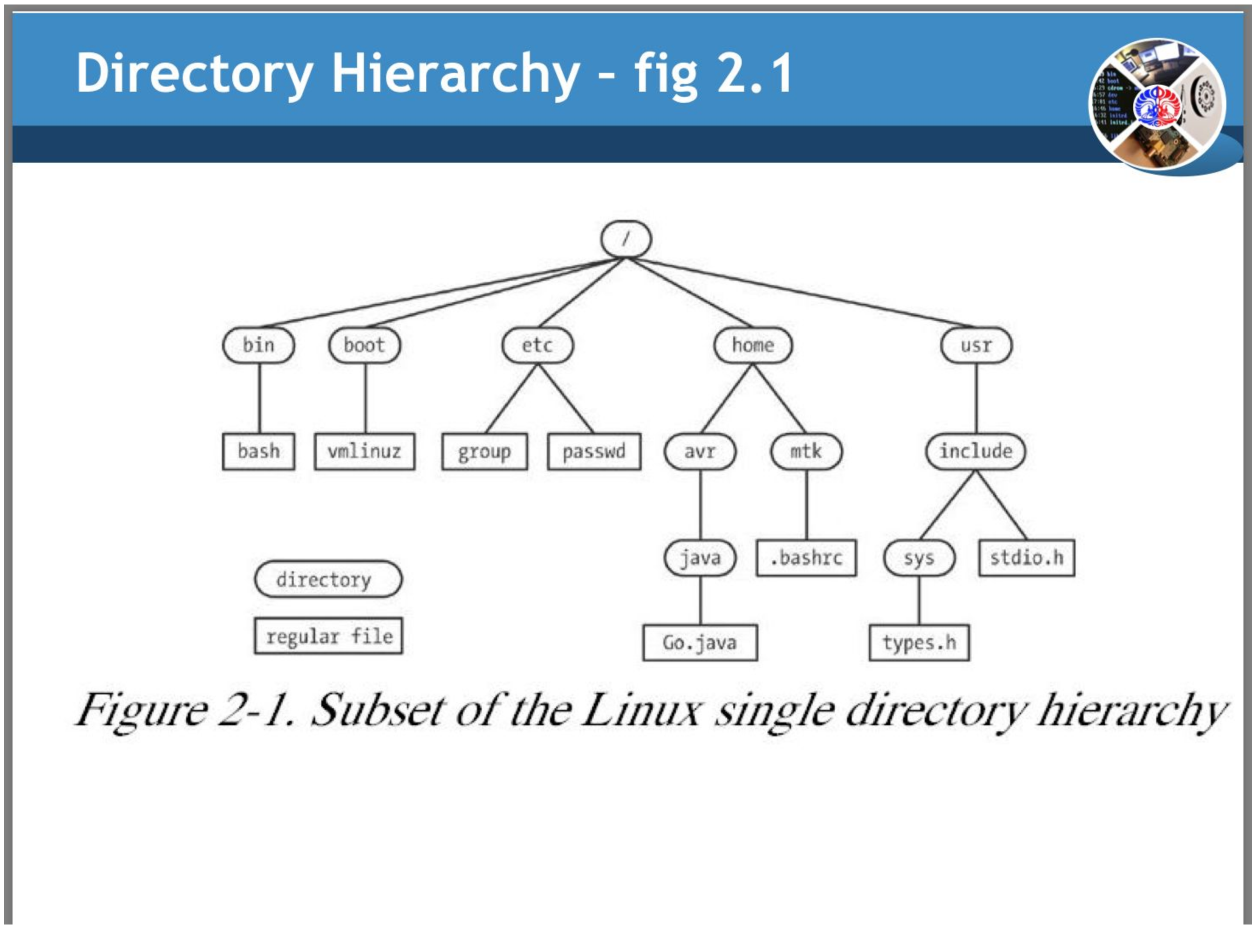
The kernel is the core of an operating system. Kernel is a bridge between hardware and software of a system. Kernel is a program that has a complete control over everything inside the system. Kernel is a part of an operating system. Kernel mode has unlimited access to the system without any restrictions to every part of the system. User mode only has access to the outer layer of a system. We need both modes because common users just need User mode to prevent accidental error. When needed, more advanced users can use the kernel mode to access the system.

There are several kernel main tasks such as:

- Process scheduling
- Memory management
- File System
- Access to devices
- /dev – device drivers
- Networking

- System call API

File Hierarchy System (FHS)



Directory Hierarchy - fig 2.1

Figure 2-1. Subset of the Linux single directory hierarchy

Taken from course’s slides.

Filetypes

- Filenames
- Regular files
- Special files
  - Directory:
    - `./` = this directory
    - `../` = parent directory

Path names

Unix basic commands:

- cd = change directory. e.g.: `cd documents`, change directory to documents folder
- pwd = print working directory, e.g.: `pwd`, the output will be: `/etc/documents/`

Relative Paths:
The start of the path starts from the working directory. e.g.: We're at `/usr/jose/` want to access `file1.txt`. The relative path is `./file1.txt`

Absolute Path:
Using the same example above, the absolute path will be `/usr/jose/file1.txt`.

The Shell

Command interpreter in Linux:

- Read command
- Perhaps pre-process command
- Fork/execute
- Return exit status of command

Users, groups, permissions

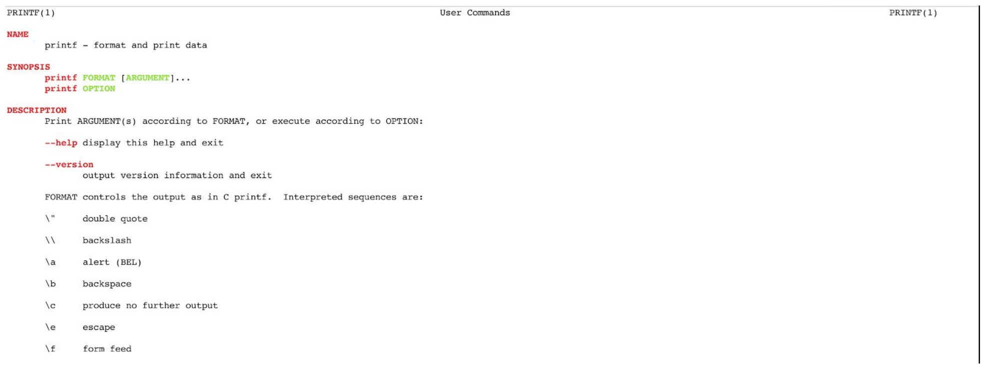
User ID (UID): `/etc/passwd`
Group ID (GID): `/etc/groups`
Superuser: `root; sudo; su`

Man – the online manual

used to display the user manual of the Linux operating system.

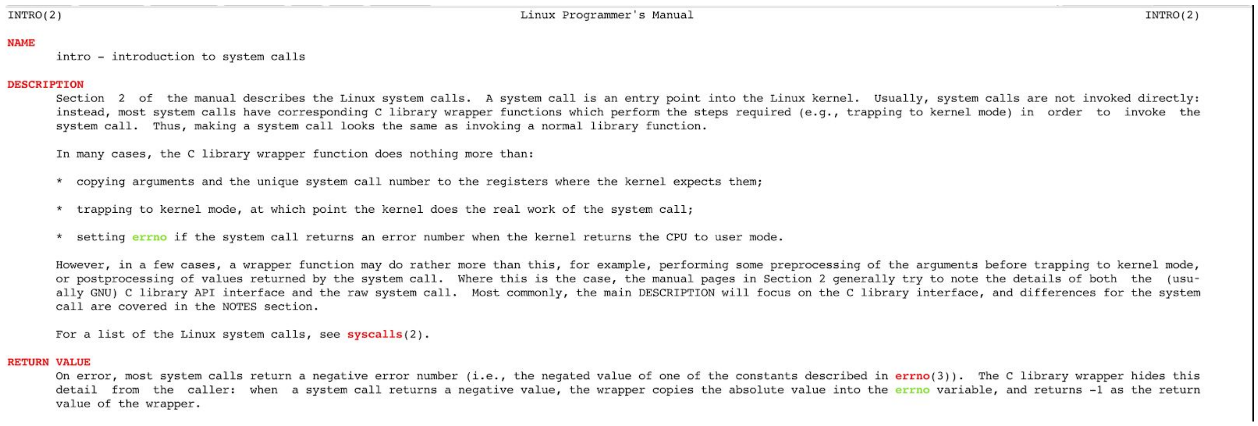
example:

- `man printf` – format and print data



```
PRINTF(1)                          User Commands                          PRINTF(1)

NAME
       printf - format and print data

SYNOPSIS
       printf FORMAT [ARGUMENT]...
       printf OPTION

DESCRIPTION
       Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

       --help display this help and exit

       --version
              output version information and exit

       FORMAT controls the output as in C printf.  Interpreted sequences are:

       \"     double quote

       \\     backslash

       \a     alert (BEL)

       \b     backspace

       \c     produce no further output

       \e     escape

       \f     form feed
```

- `man 2 intro` – introduction to system calls



```
INTRO(2)                          Linux Programmer's Manual                          INTRO(2)

NAME
       intro - introduction to system calls

DESCRIPTION
       Section  2  of  the manual describes the Linux system calls.  A system call is an entry point into the Linux kernel.  Usually, system calls are not invoked directly:
       instead, most system calls have corresponding C library wrapper functions which perform the steps required (e.g., trapping to kernel mode) in  order  to  invoke  the
       system call.  Thus, making a system call looks the same as invoking a normal library function.

       In many cases, the C library wrapper function does nothing more than:

       *  copying arguments and the unique system call number to the registers where the kernel expects them;

       *  trapping to kernel mode, at which point the kernel does the real work of the system call;

       *  setting errno if the system call returns an error number when the kernel returns the CPU to user mode.

       However, in a few cases, a wrapper function may do rather more than this, for example, performing some preprocessing of the arguments before trapping to kernel mode,
       or postprocessing of values returned by the system call.  Where this is the case, the manual pages in Section 2 generally try to note the details of both  the  (usu-
       ally GNU) C library API interface and the raw system call.  Most commonly, the main DESCRIPTION will focus on the C library interface, and differences for the system
       call are covered in the NOTES section.

       For a list of the Linux system calls, see syscalls(2).

RETURN VALUE
       On error, most system calls return a negative error number (i.e., the negated value of one of the constants described in errno(3)).  The C library wrapper hides this
       detail  from  the  caller:  when  a system call returns a negative value, the wrapper copies the absolute value into the errno variable, and returns -1 as the return
       value of the wrapper.
```