CS 153 Lab1

Kishore Kumar Pusukuri

Outline

Overview of Linux/Unix
UNIX/Linux Goals
Interfaces to Linux
Shell
Linux Utility Programs
Booting Linux/UNIX
After booting process
Next lab ? System Calls...

- Linux is a variant of UNIX (with only small deviations).
- Linus Torvalds. 1991.
- Monolithic kernel rather than Micro-kernel.
- Linux is free, this license, the GPL (GNU Public License), is no longer than Microsoft's Windows license and specifies what you can and cannot do with the code.
- Users may use, copy, modify, and redistribute the source and binary code freely. The main restriction is that all works derived from the Linux kernel may not be sold or redistributed in binary form only ; the source code must either be shipped with the product or be made available on request.

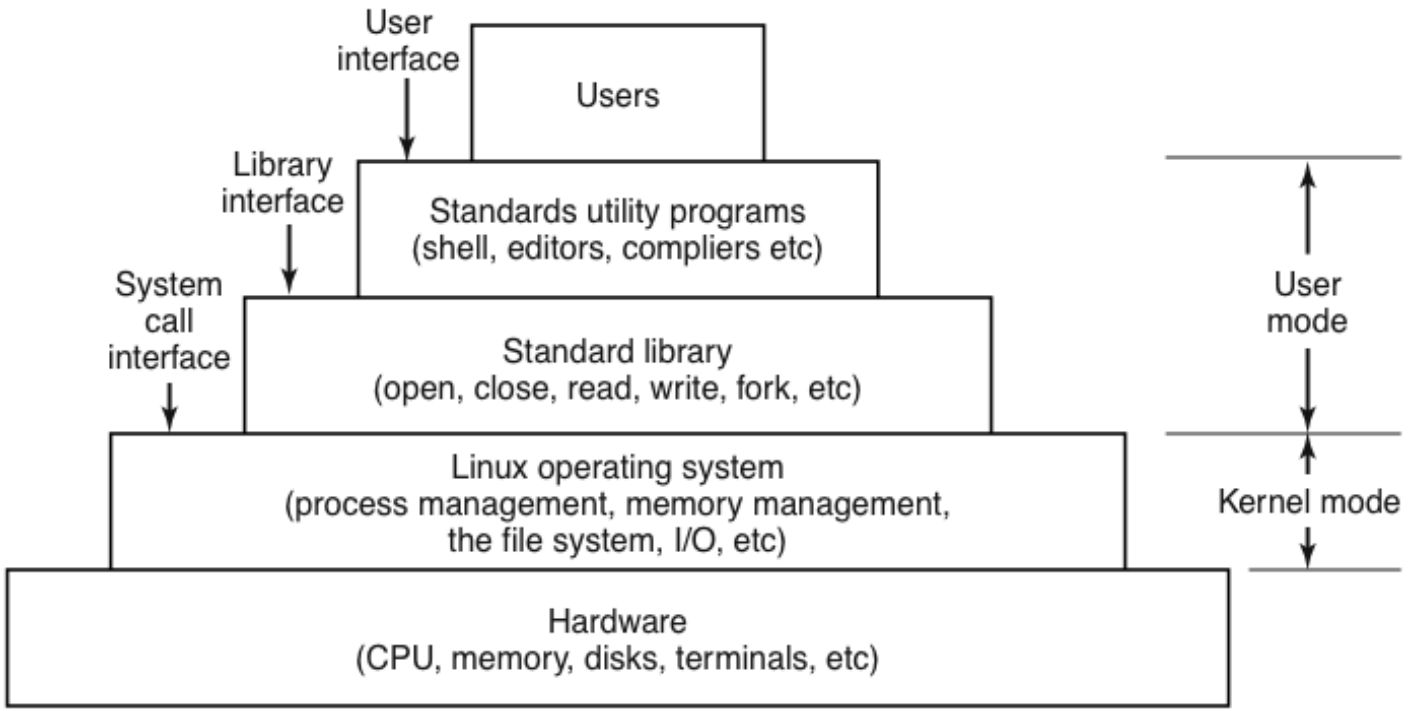
UNIX/Linux Goals

- Designed to handles multiple processes and multiple users at the same time.
- Simple. elegant, and consistent.
- For example, at the lowest level, a file should just to be a collection of bytes. Having different classes of files should just be a collection of bytes.
- Similarly, if the command ls A* means list all the files beginning with "A", then the command rm A* means remove all the files beginning with "A" and not remove the one file whose name consists an "A" and an asterisk.
- This characteristic is sometimes called the "principle of least surprise".

cont...

- Provide power and flexibility. A system should have a small number of basic elements that can be combined in an infinite variety of ways to suit the application.
- No redundancy. Example : Why type copy when cp is enough ?
- These are the things programmers (we) want ...

Interfaces to Linux



cont..

- Control the hardware and provide a system call interface to all the programs.
- These system calls allow user programs to create and manage processes, files, and other resources.
- What is POSIX ? Please read...
- Standard utility programs : shell (command processor / command interpreter), compilers, editors etc..

Shell

- Graphical user interface, but sophisticated users prefer command line interface (Shell).
- There are types of shells. Bash shell, ksh, csh etc...
- Bash is the default shell in most Linux systems.
- Terminal emulator program (xterm) provides standard input , standard output, and standard errro along with the Shell.
- Shell reads from the keyboard, and writes to the monitor, and the power to execute other programs (commands).
- command may take arguments. Ex : cp src dest. Invokes cp program with the arguments.

cont...

- flags : Arguments that control the operation of command are called flags. Note : Not all arguments are file names.
- head -20 file (default ten lines)
- head 20 file
- To make it easy to specify multiple file names, the shell accepts magic characters, sometimes called wild cards. For example : An asterisk matches all possible strings, so ls *.c ?
- ls [ape]* ?

cont...

- sort <inputfile > outputfile
- A program that reads its input from standard input (keyboard), does some processing on it, and writes its output to standard output (monitor) is called filter.
- sort <inputfile > temp ; head -30 < temp ; rm temp
- sort < inputfile | head -30
- we can execute multiple programs, ex : wc -l < inputfile &
- Files containing shell commands are called shell scripts. Shell process them in the order.

Linux Utility Programs

- File and directory manipulation commands. Ex : cp a b etc.
- Filters : head, tail, grep, sort etc.
- Program development tools, such as editors, compilers : gccetc.
- System administration : chmod etc

Common Linux utility programs

Program	Typical use
cat	Concatenate multiple files to standard output
chmod	Change file protection mode
cp	Copy one or more files
cut	Cut columns of text from a file
grep	Search a file for some pattern
head	Extract the first lines of a file
ls	List directory
make	Compile files to build a binary
mkdir	Make a directory
od	Octal dump a file
paste	Paste columns of text into a file
pr	Format a file for printing
ps	List running processes
rm	Remove one or more files
rmdir	Remove a directory
sort	Sort a file of lines alphabetically
tail	Extract the last lines of a file
tr	Translate between character sets

What is a bootloader

"A bootloader is a program that is run when a computer is started that is responsible for loading an operating system."

- Bootloaders usually contain several ways to boot the OS kernel.
- On a PC, the boot loader is read from disk and generally loads the kernel from the disk, while on other computer architectures it may be built into the boot firmware.
- Since it is usually the first software to run after powerup or reset, it is highly processor and board specific.

What a bootloader should do

- Configuring the system's memory
- Loading the kernel image
- Loading an initial RAM disk
- Initialising a console
- Kernel parameters
- Obtaining the Linux machine type
- Starting the kernel

Configuring the system's memory

- The bootloader is expected to find and initialise all RAM that the kernel will use for volatile data storage in the system.
- It performs this in a machine dependent manner.
- It may use internal algorithms to automatically locate and size all RAM, or it may use knowledge of the RAM in the machine, or any other method the bootloader designer sees fit.
- The kernel should have no knowledge of the setup or configuration of the RAM within a system other than that provided by the bootloader.
- The physical memory layout is passed to the kernel using some parameter.

Loading the kernel image

- Kernel images generated by the kernel build process are either uncompressed "Image" files or compressed zImage files.
- The uncompressed Image files are generally not used, as they do not contain a readily identifiable magic number.
- The zImage has several benefits in addition to the magic number.
- Typically, the decompression of the image is faster than reading from some external media.
- The integrity of the image can be assured, as any errors will result in a failed decompress.
- The kernel has knowledge of its internal structure and state, which allows for better results than a generic external

Kernel parameters

- The bootloader must pass parameters to the kernel to describe the setup it has performed.
- The size and shape of memory in the system and, optionally, numerous other values.
- The list must be stored in RAM and placed in a region of memory where neither the kernel decompresser nor initrd manipulation will overwrite it.

Starting the kernel

- Once the bootloader has performed all the other steps it must start execution of the kernel with the correct values in the CPU registers.
- The bootloader is expected to call the kernel image by jumping directly to the first instruction of the kernel image.

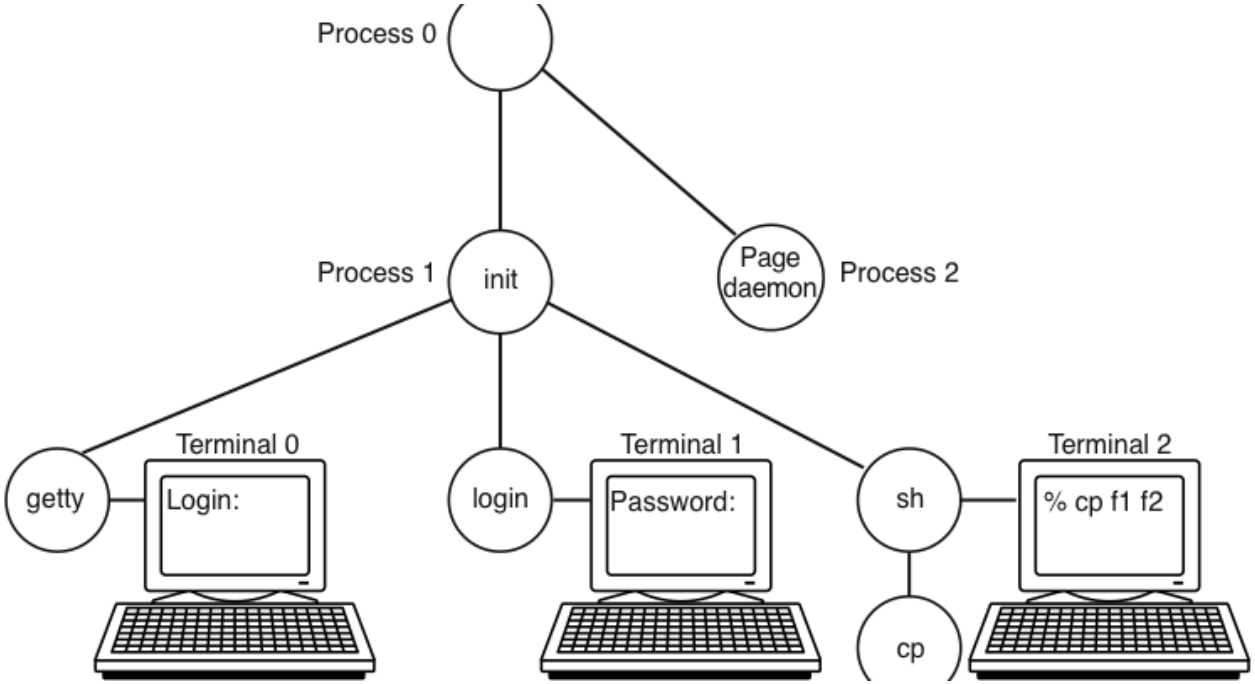
Linux booting process

- When the computer starts, the first sector of the boot disk (the master boot record) is read into memory and executed.
- This sector contains a small program (512-byte) that loads a standalone program called boot/boot-loader from the boot device, usually a disk. Why 521-byte ?
- The boot program copies itself to the main memory and reads the root directory of the boot device.
- To do this, it must understand the file system and directory format, which is the case with GRUB (GRand Unified Bootloader). Other like LILO, do not rely on specific file system, instead it needs a block map and low-level addresses that describes physical sectors...

- Then boot reads in the operating system kernel and jumps to it. At this point, it has finished its job and the kernel is running.

Initial processes

- Configures hardware. Creats process 0. It programs the real-time clock, mounting root file system, and creating init (process 1) and page daemon (process 2).
- init checks its flags to see if it is supposed to come up single user or multiuser.
- In the former case, it forks off a process that executes the shell and waits for this process to exit. In the latter case according to /etc/rc system initialization script, it checks file consistency checks...etc.
- Next, it executes a program called Getty.
- Getty sets the line speed and then types login : on the terminals screen and tries to read the user's name from the


Process 0
Process 1
init
Page daemon
Process 2
Terminal 0
getty
Login:
login
Terminal 1
Password:
sh
Terminal 2
% cp f1 f2
cp

- After entering a login name, getty terminates by executing /bin/login, the login program.
- Next login asks a password, encrypts it, and verifies it against the encrypted password stored in the password file, /etc/passwd.
- If it is correct, then login replaces itself with the user's shell, which then waits for the command.

System Calls

- fork()
- exec()
- open()
- read()
- write() ..etc

Thaaaank Yoooouuu!!!