COEN 146: Computer Networks
Lab 1: Basic Linux, network commands, and C programming skills

Objectives

1. To review Linux/Unix commands.
2. To write sample C programs that demonstrate the use of system calls and files.

Guidelines

For all COEN 146L, you need to develop a good knowledge of Linux/Unix commands and write C in a Linux development environment. You are highly encouraged to use command line tools in developing, compiling, and running your programs. This Lab is designed to familiarize you with the basic Linux commands and programming skills in C that you will need for the upcoming labs.

Please pay attention to your coding style and good programming practices; if your program is not worth documenting, it probably isn't worth running.[1] Please follow the GNU coding standards available on: https://www.gnu.org/prep/standards/html_node/Writing-C.html.

Unix/ Linux

Linux is a Unix-like operating system, following the design principles of the Unix monolithic kernel in process control, CPU scheduling, memory management, file systems, networking, and access to the peripherals. Please read the details on https://en.wikipedia.org/wiki/Linux.

Unix/ Linux has had a culture of distinctive art and powerful design philosophy since its inception in 1969. Software technologies come and go, but Unix/ Linux remained dominant and continued to evolve on various machines ranging from supercomputers and PCs to handheld devices and embedded networking hardware. C language is ubiquitous and a central technology of Unix/ Linux. It is tough to imagine developing applications at the core system level without C. The POSIX, Portable Operating System Standard, and the Unix API (Application Programming Interface) are used to write genuinely portable software that can run across a heterogeneous mix of computers. TCP/IP (which you will learn in this class) and Unix represent the core technologies of the Internet. For more details, see [2].

It is recommended that you install Linux on your, either as a bare operating system or as a virtual machine. For details, please check https://www.linux.com. If you are using Mac OS, you can use Linux commands in a terminal window, including the vi and gcc compiler.

Command-line

The use of command-line programs is a tradition in Unix\Linux. Commands are executable programs that can run with various arguments (options). Command-line options are single letters preceded by a single hyphen, including:

-a: all, -b: buffer, -c: command, -d: debug, -e: execute, -f: file, -l: list, -o: output, -u: user

Some of the basic commands are:

- ls: lists all files and directories (try with options: -a, -al)
- cat: displays file content (try cat file1 file2 > file3)
- mv: moves a file to a new location (try mv file1 file2)
- rm: deletes a file

[1] Jonathan Nagler, "Coding Style and Good Computing Practices", *PS: Political Science and Politics*, Volume 28, Issue 3, 1995, pp. 488-492.

[2] Eric S. Raymond, *The Art of Unix Programming*, Pearson, 2004

- cp: copy file
- man: gives helpful information on a command
- history: gives a list of past commands
- clear: clear the terminal
- mkdir: creates a new directory
- rmdir: deletes a directory
- echo: writes arguments to the standard output (try echo 'Hello World' > myfile)
- df: shows disk usage
- apt -get: install and update packages
- mail -s 'subject' -c 'cc-address' -b 'bcc-address' 'to-address' < filename: sends email with attachment
- chown/ chmod: change ownership/ permission of file or directory
- date: show the current date and time
- ps: displays active processes
- kill: kills the process
- sh: bourne shell – command interpreter (good to learn about shell programming)
- grep: searches for a pattern in files
- Ctrl+c: halts current command
- Ctrl+z: stops current command and resumes with foreground
- Ctrl+d (exit): logout of current session

In general, you will probably use the commands - ls, more, mv, rm, mkdir, rmdir, cd, cp, chmod, who, ps, kill, ctrl+c, cmp, grep, cat, and man – more often. The man helps you to learn about a specific command and its use in Linux. For example, $man cat displays the meaning and usage of cat.

Network commands and tools

Step 1. [20%] Run each of the following basic networking commands in Linux, write down your observation and explain the usage of the command

a. netstat: displays the contents of network interfaces; with – a option, the command displays the state of all active sockets (more to come on sockets as endpoints of communication). With the – r option, the routing table is displayed. Please type man netstat to learn about all options.

b. ifconfig: configures network interface parameters. With – a option, the state of all interfaces is displayed. Other options are also used to assign a new IP address to an interface, assign a new network mask for an interface, disable an interface, and more. Please type man ifconfig to learn about all options.

c. hostname: displays and sets the hostname of the system. Please type man hostname to learn about all options.

d. ping: sends ECHO_REQUEST datagram to a network host using ICMP protocol to elicit an ECHO_RESPONSE from the host or gateway. Please type man ping to learn about all options.

e. traceroute: displays the route packets trace to a network host using IP protocol "time to live". Please type man traceroute to learn about all options.

f. telnet: remote connection to the server at a specific port (mainly 80 – http port)

g. host/dig: performs DNS lookups.

h. route: manipulates network routing tables.

i. arp: displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol.

Note: Some commands may require you to log in with admin privileges.

Step 2. [20%] Select three hosts on the internet (one in North America, one in Asia, and one in Europe), and experiment with pinging each host with different packet sizes ranging from 32 to 1048 bytes. For each host,

a. identify the packet loss.

b. identify the RTT "Round-Trip-Time".

Explain the correlation between these measurements to the geographical location of the hosts.

System calls

System calls are often called kernel calls. They are c libraries that execute at the kernel level to allow users to interact with the operating system for services that include:

- Process creation and management (e.g. fork(), exec(), wait(), exit())
- File management (e.g. open(), read(), write(), close())
- Communication (e.g. pipe(), shmget(), mmap())
- Networking (e.g. socket(), bind(), connect(), listen(), sendto(), recvfrom())

Copying files

Problem: Write a C program to copy binary/ text files simultaneously.

Analysis:

- Input: pass the file as arguments to main(), so your main function needs to be defined as follows:
  ```
  int main(int argc, char * argv[])
  ```
  Your files: src.dat and dest.dat files.

- File reading can be accomplished by using either:
  - Functions: fopen, fwrite, and fread for binary files or fprintf and fscanf for text files
    - FILE *fopen(const char *filename, const char *mode)
    - fwrite(ptr, int size, int n, FILE *fp); or fprintf() (for text files)
    - fread(ptr, int size, int n, FILE *fp); or fscanf() (for text files)
    - fclose(ptr);

    e.g.
    ```
    FILE *fp;
    fp = fopen("src.dat","r");
    fp = fopen("dest.dat","w");
    fwrite(&buf,1, sizeof(buf),fp);
    fread(&buf,1,sizeof(buf),fp);
    fclose(fp);
    ```

    OR

  - System calls: open, read, write
    - int open (const char* Path, int flags [, int mode]);
    - size_t read (int fd, void* buf, size_t cnt);
    - size_t write (int fd, void* buf, size_t cnt);

e.g.:

```
int fd = open("foo.txt", O_RDWR);
int nw = write(fd, buf, strlen(buf));
int nr = read(fd, buf, 40);
close (fd);
```

You need to include the following libraries:

- #include<sys/types.h>
- #include<sys/stat.h>
- #include <fcntl.h>

You may create files of random data with different sizes/ bytes. You may use “cat” and “head” commands (i.e. $cat /dev/random | head -c <bytecount>). /dev/random are special files that serve as pseudorandom number generator. “cat” is used to display the content, and “head” is used to display the specified number of lines. So the result of “cat” is sent to the upstream end of PIPE, and “head” receives these results and redirects the content of the specified bytes to a file.

$cat /dev/random | head -c 100000 > src1.dat creates a file with 100KB
$cat /dev/random | head -c 1000000 > src2.dat creates a file with 1MB

Check the size of the files with the command “ls -la”

Step 3. [20%] Write your C program to copy files (binary and text) using functions, compile, debug, run, and test

Step 4. [20%] Write your C program to copy files (binary and text) using system calls, compile, debug, run, and test

Step 5. [20%] Calculate the time taken to copy files for steps 3 and 4. You may use `clock()` function (in time.h library). Call the clock function at the beginning and end of the code to measure the time, subtract the values, and then divide by CLOCKS_PER_SEC (the number of clock ticks per second) to get processor time. You may use the following code snippet:

```
#include <time.h>
clock_t start, end;
double cpu_time_used;

start = clock();
... /* Time-consuming process. */
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

Requirements to complete the lab

1. Demo to the TA correct execution of your programs [recall: a successful demo is 25% of the grade]
2. Submit the source code for all your programs as .c file(s) and upload it to Camino.

Please start each program/ text with a descriptive block that includes the following information minimally:

```
# Name: <your name>
# Date: <date> (the day you have lab)
# Title: Lab1 – task
# Description: This program computes … <you should
# complete an appropriate description here.>
```