#### COEN 146L: Computer Networks Lab

**Lab 1: Unix/Linux Commands and Overview of C Programming**

**Objectives**

##### To learn Unix/ Linux

##### To use command – line programs

##### To develop sample programs in C

##### 

##### **Guidelines**

##### Good knowledge in Unix/Linux based C programming is required for all labs. You are highly encouraged to use command line tools in developing, compiling, and running your programs. You may use editors like vi, emacs, nano or gedit and gcc compiler. You may also choose to use VS Code, NetBeans or CodeBlock IDE. This lab is designed to teach you basics and advances of the Linux technical environment, including tools, commands, and system calls.

##### 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]](#footnote-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 Unix monolithic kernel in process control, cpu scheduling, memory management, file systems, networking, and access to the peripherals. Please read details on <https://en.wikipedia.org/wiki/Linux>. Unix/ Linux has 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 a wide variety of 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 very hard to imagine developing applications at the core system level without C. The POSIX, Portable Operating System Standard, the Unix API (Application Programming Interface) is used to write truly 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]](#footnote-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 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 variety of 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
* cp: copy file
* man: gives help 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 process
* sh: bourne shell – command interpreter (good to learn about shell programming)
* grep: searches for pattern in files
* Ctrl+c: halts current command
* Ctrl+z: stops current command and resumes with foreground
* Ctrl+d (exit): logout of current session

**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())

**C Program with two processes**

Demonstrate each of the following steps to the TA to get a grade on this part of the lab assignment

1. [5%] Write the following C program in a Linux environment using vi, nano, emacs, or an editor of your choice

/\*Sample C program for Lab assignment 1\*/

#include <stdio.h> /\* printf, stderr \*/

#include <sys/types.h> /\* pid\_t \*/

#include <unistd.h> /\* fork \*/

#include <stdlib.h> /\* atoi \*/

#include <errno.h> /\* errno \*/

/\* main function with command-line arguments to pass \*/

int main(int argc, char \*argv[]) {

pid\_t pid;

int i, n = atoi(argv[1]); // n microseconds to input from keyboard for delay

printf("\n Before forking.\n");

pid = fork();

if (pid == -1) {

fprintf(stderr, "can't fork, error %d\n", errno);

}

if (pid){

// Parent process

for (i=0;i<100;i++) {

printf("\t \t \t Parent Process %d \n",i);

usleep(n);

}

}

else{

// Child process

for (i=0;i<100;i++) {

printf("Child process %d\n",i);

usleep(n);

}

}

return 0;

}

1. [5%] Compile the program using gcc compiler by typing gcc YourProgram.c – o ExecutableName. When it compiles without errors or warnings, make a copy of the source file then go to step 3.
2. [5%] Run the program by typing ./ExecutableName and take a note of your observation.
3. [10%] Re-run the program by typing ./ExecutableName 3000. Note that the delay in the loop depends on the command line argument you give, here the delay is 3000 microseconds.
   1. Enter delays of 500 and 5000, what happens?
4. [30%] Rewrite the program in Step 1. with two threads instead of two processes, then demonstrate steps 1 – 3 to the TA. Include the pthread.h library and use the function pthread\_create() instead of fork().

#include <pthread.h>

int pthread\_create(pthread\_t \*thread, pthread\_attr\_t \*attr,

void \*(\*start\_routine) (void \*arg), void \*arg);

When your program compiles without errors or warnings, make a copy of the source file

Note: you will most probably use - ls, more, mv, rm, mkdir, rmdir, cd, cp, chmod, who, ps, kill, ctrl+c, cmp, grep, cat, and man – commands in linux. Type man cat in the command line to learn about cat command, as an example.

##### **Circuit switching and packet switching**

Write a C program that implements quantitative comparisons between circuit switching and packet switching according to the following description

Variables

* The bandwidth of a network link is denoted by int linkBandwidth;
* The bandwidth required for a given user is denoted by int userBandwidth;
* The number of circuit switching users is denoted by int nCSusers;
* The number of packet switching users is denoted by int nPSusers;
* The percentage of time a packet switching user needs to transmit is denoted by double tPSuser;
* The probability that a given (*specific*) packet switching user is busy transmitting is denoted by double pPSusersBusy;
* The probability that one (*specific*) packet switching user is not busy transmitting is denoted by double pPSusersNotBusy;

Scenarios: Consider the following two scenarios:

1. A circuit-switching scenario in which nCSusers users, each requiring a bandwidth of userBandwidth Mbps, must share a link of capacity linkBandwidth Mbps.
2. A packet-switching scenario with nPSusers users sharing a linkBandwidth Mbps link, where each user again requires userBandwidth Mbps when transmitting, but only needs to transmit at a percentage of tPSuser.

Computations:

1. [15%] Circuit switching scenario
   1. The number of circuit-switched users that can be supported is computed as follows:

nCSusers = linkBandwidth/ userBandwidth;

1. [30%] Packet switching scenario
   1. The probability that a given (*specific*) user is busy transmitting is computed as:

pPSusersBusy = tPSusers;

* 1. The probability that one specific other user is not busy is computed as:

pPSusersNotBusy = 1 - pPSusers;

* 1. The probability that *all* of the other specific other users are not busy is computed as:

(1 – pPSusers) *nPSusers -1*;

* 1. The probability that one specific user is transmitting and the remaining users are not transmitting is computed as:

*pPSusers1* \* pPSusersNotBusy *nPSusers -1*

* 1. The probability that exactly one (any one) of the nPSusers users is busy is  pPSusers times the probability that a given specific user is transmitting and the remaining users are not transmitting, i.e.:

nPSusers \*( *pPSusers1* \* pPSusersNotBusy *nPSusers -1*)

* 1. The probability that 10 specific users of nPSusers are transmitting and the others are idle is computed as:

*pPSusers10* \* pPSusersNotBusy *nPSusers -10*

* 1. The probability that any 10 users of nPSusers are transmitting and the others are idle is computed as:

(nPSusers, 10) \* *pPSusers10* \* pPSusersNotBusy *nPSusers -10*, where

(nPSusers, 10) = nPSusers!/(10!\*(nPSusers – 10)!) co-efficient of binomial distribution

* 1. The probability that more than 10 users of nPSusers are transmitting and the others are idle is computed as:

Σ *i=11..nPSusers* (nPSusers, i) \* *pPSusersi* \* pPSusersNotBusy *nPSusers -i*

Demonstrate steps 6 and 7 for your program to the TA with the following inputs:

linkBandwidth = 200 Mbps

userBandwidth = 20 Mbps

tPSuser = 0.10

nPSusers = 19

**Requirements to complete the lab**

1. Show the TA correct execution of the C programs.
2. Submit your answers to questions, observations, and notes as .txt file and upload to Camino
3. Submit the source code for all your programs as .c file(s) and upload to Camino.

Be sure to retain copies of your files. You will want these for study purposes and to resolve any grading questions (should they arise)

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

# Name: <your name>

# Date: <date> (the day you have lab)

# Title: Lab1 – task

# Description: This program computes … <you should

# complete an appropriate description here.>

1. Jonathan Nagler, "Coding Style and Good Computing Practices”, *PS: Political Science and Politics*, Volume 28, Issue 3, 1995, pp. 488-492. [↑](#footnote-ref-1)
2. Eric S. Raymond, *The Art of Unix Programming*, Pearson, 2004 [↑](#footnote-ref-2)