ECE 373 Assignment #1
Spring 2022

Introduction:

The purpose of this exercise is to bring everyone to a common point with basic Linux, C programming, and the Linux-based tools for working with C programs. If you have an Ubuntu Linux on a system at home, or any other current Linux distribution, you can do all of this exercise on it and much of the exercise steps will likely be very familiar to you. If you do not have a Linux system at home, you can work through the exercises on one of the available resources provided by the CAT, or on your eventually-needed VirtualBox Virtual Machine (VM). The last bit with the kernel module cannot be done on CAT resources.

A. Information Gathering:

1. Look at the man page for the "open()" system call, found in section two of the manpages.
    A. List the BUGS when using the system call.
    B. What files need to be included to use this function?
    C. List the first three related system calls to open().
    D. Choose one of the system calls from above and list its bugs (also list what system call you chose) and files needing to be included to use the system call.
2. Use https://elixir.bootlin.com/ to search for the following:
    A. Search for "usb_device". In what file is the structure defined, and what are the first five members of the struct?
    B. In what header file is the type declared for the 5th member of the struct? (hint: don't look in test tools or staging directories)
    C. Include the entire enumeration declaration from above.

B. Basic Linux Use:

1. Log onto an Ubuntu Linux or other system (presumably your VirtualBox virtual machine)
2. Under each heading in the toolbar at the top, pop up and note the menu and sub menu tools/topics. Note where you find Terminal and the gedit Text Editor, or any other text editor of your choice.
3. Click on the Terminal entry to bring up a command shell line. This should be familiar to you from working with Cygwin in ECE 371/372. You will use Terminal to enter Linux commands.
4. To get an overview of Linux structure and commands, go to http://linuxcommand.org/lc3_learning_the_shell.php . Study the "Learning the shell" section carefully and make some notes on basic commands to make a directory, determine current directory, change to a different directory, do a long or short form listing of the files in a directory, copy a file, etc.
5. The "script" command is a useful way to make a record of all the commands you enter at the command shell and the results of those commands. Script records the commands, etc. in a file called typescript in your current directory. You terminate "script" with CTRL-D or 'exit'. You can then edit/print the typescript file. You will be turning in a copy of the typescript file for a shell practice session.
6. For a start with the basic shell commands, enter script at the command prompt and then:
    A. Enter the command to determine the current directory

   B. Enter the command to command to show the files in that directory with file permissions.
   C. Enter a command to make a directory called ECE373.
   D. Change the working directory to that directory.
   E. Exit script with CTRL D.
7. Find the gedit program under Application|Accessories and open a gedit window. Find the File|Open menu command and find the typescript file you created. Click on Open and you should see the typescript file in the window. It likely contains some control characters along with the commands but you can easily edit these out, if you want.

C. Basic C Programming in Linux:

The tools you will use for developing C programs for this class are the GNU tools, gcc, ln, and gdb. For a start with these, work through the following steps.

1. Type in a simple C program such as a Fahrenheit to Celsius program or one of the examples from the GNU C reference (https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html#A-Sample-Program) that takes in data and outputs a result to the display.
2. To compile your C program, enter the command `gcc -g -o filename filename.c` where filename is the name you gave your C file. The –g switch tells the compiler to include debug information that will be used by the gdb debugger.
3. Edit and recompile the .c file until you get no compile errors.
4. To run the program from the command line, just enter `./filename`.
5. If the program functions correctly, rejoice, turn off script, print the typescript file, and print the C source file.
6. If the program does not function correctly or even if it does, it is time to learn about gdb.
7. At the command prompt, enter `man gdb`. This should bring up the manual pages for gdb.
8. Read through the pages using the arrow keys to find out how to use gdb. Write some notes about the main gdb commands such as break, run, next, step, print, and quit. Hit the q key to exit the man pages.
9. Try running your program under gdb with a `gdb filename` command. Enter a breakpoint at main and then run the program. Execution then stops at main(). You can step through and execute one line of the program with step or next (note the difference). You can determine the values of variables at any point with the print command.
10. Experiment with gdb until you are comfortable working through a program with it.

D. Hello, Kernel:

Now that you have the basics, we get to the real point of the whole assignment: load a simple "hello" module into the kernel.

1. Following the notes from the lecture, create a simple `hello.c` file with the basic `__init` and `__exit` functions, where the init function prints "Hello, Kernel" and the exit function prints "Goodbye, Kernel".
2. Create a simple makefile for compiling the hello.c file into a kernel module
3. Compile the `hello.c` file into `hello.ko`, re-editing as necessary to clean up any and all

compiler warnings and errors.

4. Load the module into the kernel with the `insmod` utility and look for the “Hello, Kernel” message in the log file `/var/log/messages`, or in `dmesg` output
5. Remove the module from the kernel with the `rmmod` utility and look for the “Goodbye, Kernel” message in the log file.

What to turn in:

In a text file, or Word document, or Open Office document, include:

1) For the manpage exercise, all of the information from #1 in "The Work" above.
2) For the LXR exercise, copy and paste the lines of code the LXR hits on PLUS the kernel you searched against. For example, searching for `sys_gettimeofday()` on kernel 3.14, then the line of code would be lines 203 and 204, and the code would be:

   ```
   asmlinkage long sys_gettimeofday(struct timeval __user *tv,
                                    struct timezone __user *tz);
   ```

   Also include the file names requested in each of the parts of the exercise above.
3) Copy of the example C program from section C.
4) Copy of the typescript showing execution of program from section C.
5) Copy of the `hello.c` module code from section D.
6) Copy of the 'hello' and 'goodbye' logfile lines from section D.

Add all of these files to the Git repository for this assignment, check them in, and push them to your repo. All of these items must be pushed to turn them in by **11pm, Pacific time, Wednesday, 13-April-2022**.