COMP 3500 Introduction to Operating Systems
Project 1 – Using the Linux Terminal

Version 4.0
Revised: July 11, 2023

Points Possible: 100

There should be no collaboration among students. A student shouldn't share any project code with any other students. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.

Objectives:

- Prepare a Linux programming environment for the future COMP 3500 projects
- Install VMWare
- Install CentOS 7 on a virtual machine in VMWare
- Get to know the Linux operating system and use basic commands
- Install VS Code on CentOS 7
- Compile and debug your first C program in Linux

Requirements:

- Each student must independently accomplish this project assignment. You may discuss with the other COMP3500 students, but you must individually solve the coding problems.
- You should use CentOS 7 as a Linux distribution.
- Your CentOS 7 must be running on a virtual machine in VMWare

1. Setup Linux Programming Environment (10 Points)

1.1 Step 1: Install VMWare Workstation 17.0.2 Player (Difficulty Level: ★)
CAUTION! VirtualBox is not allowed in COMP 3500 projects.
Please follow the instructions specified in "*How to install CentOS 7 on VMWare.pdf*" to learn a way of installing VMWare in your own laptop or desktop computers.

1.2 Step 2: Install CenterOS 7 on a virtual machine in VMWare (Difficulty Level: ★★)
CAUTION! Ubuntu is not allowed in COMP 3500 projects because the COMP projects 2-4 are incompatible with Ubuntu.

- Please follow the instructions specified in "*How to install CentOS 7 on VMWare.pdf*" to learn how to install CentOS 7 on a virtual machine in VMWare Workstation player.
- After you install CentOS, please follow the steps to finish the OS configuration.

- Open a terminal and do "`yum -y update`" to update all the packages. More specifically, you enter "`su`" to become the root before updating your CentOS in the two commands below.

```
su
yum -y update
```

- Please make sure the following packages are installed: **gcc, g++, gedit, vim, nano, gdb, ethtool,** and **pciutils.** They most likely are already installed. You can run the following commands to confirm their availability and install as needed. If you are not a root user, you may enter "`su`" to become one before running the commands below.

```
yum -y install gcc
yum -y install g++
yum -y install gedit
yum -y install vim
yum -y install nano
yum -y install gdb
yum -y install ethtool
yum -y install pciutils
```

- The "`yum -y install ncurses-devel`" command installs the ncurses-devel package, which contains header files and libraries for developing applications that use the ncurses terminal handling library. You must have the ncurses-devel package for your future COMP 3500 projects 2-4.

```
yum -y install ncurses-devel
```

2. The `script` Command

The script command line tool allows you to save a session of your terminal. In addition to saving each command per line in a text file, the script command makes a typescript of everything that happens on your Linux terminal. Screencasting tools to a desktop session (GUI) is what script is to a terminal. Let us demonstrate the usage of `script` through the following example:

```
script
Script started, file is typescript
cd ~
ls
exit
Script done, file is typescript
```

Then, you may use the `mv` command to change the file name from `typescript` to any name you like. Alternatively, you may specify the name of your log file upfront as below:

```
script sample.script
Script started, file is typescript
```

```
cd ~
ls
exit
Script done, file is sample.script
```

3. Tasks (90 Points)

Script the following session using the `script` command. You may save each session (i.e., each task below) in one script file. Using the *`tar`* command to submit a tarred and compressed file named `project1.tgz` (see Section 4.2 for details).

3.1 (Task 1: 25 points. Difficulty Level: ★) Please use the `script` command to create a file named "`commands.script`" demonstrating that you understand how to use the following basic Linux commands.

- `man [command]` – displays the help information for the specified command.
- `cd` – changes a directory.
- `pwd` – displays the pathname for the current directory.
- `ls` – lists the files in the current directory.
- `mkdir` – makes a new directory.
- `cp` – copies a file from one location to another.
- `mv` – moves a file from one location to another.
- `rm` – removes a file.
- `rmdir [options] directory` – deletes empty directories.
- `chmod [options] mode filename` – changes a file's permissions.
- `clear` – clears a command line screen/window for a fresh start.
- `top` – displays the resources being used on your system. Press `q` to exit.
- `who [options]` – displays who is logged on.
- `nproc` – displays the number of cores.

Reference: How to Start Using the Linux Terminal
https://www.howtogeek.com/140679/beginner-geek-how-to-start-using-the-linux-terminal/

3.2 (Task 2: 10 points. Difficulty Level: ★) When you log in a Linux system, you should get to know the system in more detail. You are asked to find out your computer system's attributes, including **CPU frequency, cache size, memory size, the list of PCI devices, hard drive, network MAC address and link speed,** and **the devices generating interrupts**. The following system commands can help you. Many of these commands are for system administration, so you should run them as root if needed.

```
more /proc/cpuinfo
more /proc/meminfo
more /proc/interrupts
lspci
```

Tip: You may pipe the output of any Linux command to a file on Linux. For example:

```
ls > test.txt
```

The above command writes all files in the current directory to a file called "`test.txt`" instead of displaying the files on a monitor. If you choose this option, you should include the pipeline output file(s) in your submitted tarball.

Please store the output from the above four commands into the following four files using the pipe (see the above tip).

- `cpuinfo.txt`
- `meminfo.txt`
- `interrupts.txt`
- `lspci.txt`

Tip: You should use the following commands to create the above four text files.

```
more /proc/cpuinfo > cpuinfo.txt
more /proc/meminfo > meminfo.txt
more /proc/interrupts > interrupts.txt
lspci > lspci.txt
```

3.3 (Task 3: 30 points. Difficulty Level: ★★) With the computer up and running, you should give it a try to see if you can use the utilities on the system. For a system programmer, these include at least the editor, the compiler, the libraries, and the debugger. You are asked to do the following

Note: You must make use of the script command (see Section 2 page 3) to save a session for task 3. The script file name should be "`gdb.script`".

3.3.1. Using your favorite editor, you should write a program (`simple.c`) that processes an array of 10 numbers, calculates the average of their square roots, and prints it out.

Tip: You may use either VS Code, nano, gedit, or vim (vi) as an editor for this project. You should be able to type the `simple.c` program without too much effort in VS Code, nano, and gedit. In your future projects, you should make use of VS Code to write your source code.

3.3.2. The GNU compile is the default open source compiler on Linux. You should check a little on what `gcc` you have, and then compile your program as follows. Please do not forget the flag '`-g`' for using debugger in Step 4 described in Section 3.3.4 on page 5.

```
gcc -v
gcc -g -lm -o simple simple.c
```

3.3.3 Practice the command **ldd**, and understand the libraries on which your program is dependent upon for execution.

```
man ldd
ldd simple
```

3.3.4 The debugger is a friend you must get acquainted with to be a good programmer. Here is a little trick in using the GNU `gdb` debugger.
(1) First run your program, **simple**, alone

(2) Using VS Code, nano, or gedit to create a file name as .gdbinit in your home directory. Your .gdbinit file should contains the following lines:

```
file simple
break main
break sqrt
info registers
```

Note: Please ensure that your .gdbinit file is located in your home directory under the following path:

```
~/.gdbinit
```

You may create or edit `.gdbinit` file by adding the following line:

```
set auto-load safe-path /
```

This line allows our programs to use a current directory initialization file, meaning that you can create a `.gdbinit` for each future project you will be debugging.

Reference on gdbinit: [GDB - Init File — Debugging documentation (unsw.edu.au)]

(3) You are advised to go through the gdb tutorial documented below:
https://www.geeksforgeeks.org/gdb-step-by-step-introduction/
Now, you should demonstrate that you are capable of running the gdb debugger with these commands (`'r'`, `'s'`, `'n'`, and `'c'`), one command at a time complete the program.

Note: You don't have to follow the above command sequence in gdb: you may run the gdb commands in any order of your choice. You will receive full credit if your gdb.script file shows that you are familiar with the basic gdb commands such as run, next, step, continue.

3.3.5 Your project report should include the source program, and the output from Steps (1)-(4).

3.4 (Task 4: 25 points. Difficulty Level: ★★) Now you should learn how to use `git` – a version control system - to manage your software development conducted by your future groups. An online book focusing on git can be found here: https://git-scm.com/book/en/v2.

Note: You must make use of the script command (see Section 2 page 3) to save a session for task 4. The script file name should be "`git.script`".

3.4.1 Install git

```
sudo yum install git
```

Reference: https://www.linode.com/docs/development/version-control/how-to-install-git-on-linux-mac-and-windows/

CAUTION! Please refer to the following webpage for detailed instructions on Steps 3.4.2-3.4.9.
https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository

3.4.2 Initializing a Repository in an Existing Directory. The first command below creates a directory "comp3500/project1" in your home directory.

```
mkdir -p ~/comp3500/project1
cd ~/comp3500/project1
```

If you want to start version-controlling existing files (as opposed to an empty directory), you should probably begin tracking those files and do an initial commit. You can accomplish that with a few `git add` commands that specify the files you want to track, followed by a `git commit`. You should run a list of 26 git commands below, where the meanings of the commands are presented prior to the commands.

(1) The following four commands create an empty git repository, followed by three empty source code files – `cpu.c, driver.c`, and `mem.c`.

```
git init
touch cpu.c
touch driver.c
touch mem.c
```

(2) The `git status` command displays information about the current state of your working directory and your Git repository. The `git add *.c`

command is used to stage -- preparing for commit -- all files in the current directory that have the .c file extension.

```
git status
git add *.c
git status
```

(3) The `git commit -m 'Initial project 1'` command creates a new commit in a Git repository with a brief commit message called `'Initial project 1'` describing the changes being committed. A commit is a snapshot of the changes you've staged above using the `git add` command.

```
git commit -m 'Initial project 1'
git status
```

(4) If you accidentally delete a source code file, you can use the `git checkout` command to recover the removed file. The following example shows that `cpu.c` is removed, but it can be easily restored using the `checkout` command.

```
rm cpu.c
ls
git checkout cpu.c
ls
```

(5) The `git checkout -b main2` command is used to create a new branch named `main2` and switch to that branch in a Git repository. Then, in this new branch, cpu2.c is created, added an committed.

```
git checkout -b main2
touch cpu2.c
git add cpu2.c
git commit -m ‘cpu2 is added into main2’
```

(6) The `git branch` command is used in Git to list and manage branches within a repository. The command shows that there are two branches, namely `master` and `main2`. , and the current branch marked with ‘*’ is `main2`. The git checkout master command switches the current branch from `main2` to `master`. You may use the `ls` command to verify the source code files in each branch.

```
git branch
ls
git checkout master
git branch
ls
```

(7) The `git clone -b master . ../new_project1` command clones a Git repository while specifying a specific `master` branch and copying the repository contents into a new directory called `new_project1` located one level above the current directory. The `cd ..`, `ls`, and `cd new_project1` commands allow you to verify the content of the newly cloned directory.

```
git clone -b master . ../new_project1
cd ..
ls
cd new_project1
```

3.4.3 Cloning an Existing Repository

You clone a repository with git clone <url>. For example, if you want to clone the Git linkable library called libgit2, you can do so like this:

```
git clone https://github.com/libgit2/libgit2
```

That creates a directory named libgit2, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new libgit2 directory that was just created, you'll see the project files in there, ready to be worked on or used.
If you want to clone the repository into a directory named something other than libgit2, you can specify the new directory name as an additional argument:

```
git clone https://github.com/libgit2/libgit2 mylibgit
```

3.4.4 Checking the status of your files:

```
git status
```

3.4.5 Tracking New Files

In order to begin tracking a new file, you use the command git add. To begin tracking the README file, you can run this:

```
git add README
```

3.4.6 Staging Modified Files

Now we change a file that was already tracked. If you change a previously tracked file called `CONTRIBUTING.md` and then run your `git status` command again.

3.4.7 Viewing Your Staged and Unstaged Changes

If the `git status` command is too vague for you — you want to know exactly what you changed, not just which files were changed — you can use the `git diff` command.

```
git diff
```

3.4.8 Committing Your Changes
Now that your staging area is set up the way you want it, you can commit your changes. Remember that anything that is still unstaged — any files you have created or modified that you haven't run git add on since you edited them — won't go into this commit. They will stay as modified files on your disk. In this case, let's say that the last time you ran git status, you saw that everything was staged, so you're ready to commit your changes. The simplest way to commit is to type `git commit`:

```
git commit
```

3.4.9. The most basic and powerful tool to view the commit history is the `git log` command.

```
git log
```

4. Deliverables

One of the following deliverables is acceptable. You may choose to submit a single PDF file (see Section 4.1) or a tarred and compressed file (see Section 4.2).

4.1 Project Report

Please submit your project report with needed contents as specified in the questions (see Section 3). You must submit your report through Canvas (no e-mail submission is accepted. Please report any problems you have solved when you learn Linux commands and `git`.

The file name should be formatted as:

```
"project1.pdf" or  "project1.txt"
```

4.2 Multiple Script Files

If you have generated multiple script files, please save all the script files in one directory say (project1). Then, you should archive all the script files (see Sections 3.1, 3.3, 3.4) and text files (see Section 3.2) along with your report (See Section 4.1) and the source code (`simple.c`) into a single tarred and compressed file. Assume that the script files, text files, and your report are located in `~/comp3500/project1`, then you can follow the instructions below to prepare a single compressed file. After you successfully execute the tar command, project1.tgz will exist in the `~/comp3500` directory.

```
cd ~/comp3500
tar vfzc project1.tgz project1
```

Your tarball should embrace the following files:

- project1.pdf or project1.txt (see Section 4.1)

- commands.script (see Section 3.1)
- cpuinfo.txt (see Section 3.2)
- meminfo.txt (see Section 3.2)
- lspci.txt (see Section 3.2)
- interrupts.txt (see Section 3.2)
- gdb.script (see Section 3.3)
- simple.c (see Section 3.3)
- git.script (see Section 3.4)

5. Grading Criteria

1) Setting up your Linux programming environment: 10% (see Section 1)
2) Using Linux commands: 25% (see Section 3.1)
3) Getting to know your system: 10% (see Section 3.2)
4) Using `gcc` and `gdb`: 30% (see Section 3.3)
5) Using `git`: 25% (see Section 3.4)

6. Late Submission Penalty

- Ten percent (10%) penalty per day for late submission. For example, an assignment submitted after the deadline but up to 1 day (24 hours) late can achieve a maximum of 90% of points allocated for the assignment. An assignment submitted after the deadline but up to 2 days (48 hours) late can achieve a maximum of 80% of points allocated for the assignment.
- Assignment submitted more than 3 days (72 hours) after the deadline will not be graded.

7. Rebuttal Period

- You will be given a period of **one week** to read and respond to the comments and grades of your homework or project assignment. The TA may use this opportunity to address any concern and question you have. The TA also may ask for additional information from you regarding your homework or project.