CPE-2600 FALL 2024

Systems Programming

Lab 1 WSL Setup

Introduction

The purpose of this assignment is to set up the Windows Subsystem for Linux (WSL). WSL acts as a virtual machine that will allow you to develop code on a Linux based operating system while working in your Windows environment. By completing this exercise, you will have the ability to install Linux within WSL, create a program, and start learning about operating systems based on Linux.

For more information on WSL see the documentation on Microsoft's web page: https://docs.microsoft.com/en-us/windows/wsl/

You may work on this assignment with one or two other students if you choose, but everyone must complete each step (and deliverable step) so as to have his/her own working development environment. Submissions are to be made individually.

The deliverable will be a series of screen-caps depicting various steps of the setup procedure with occasional commentary. Collect the screen caps and commentary in a Word doc as you go and you will submit the completed assignment in pdf format to Canvas. The submission is expected prior to departing the lab period in Week 1.

Objectives

By the end of this assignment you will be able to:

- Create a fresh installation of Linux
- Understand the concepts of the Windows Subsystem for Linux (WSL)
- Demonstrate an ability to use a Linux shell.
- Use the 'man' command to obtain documentation about Linux commands
- Explain how to list the contents of a directory in multiple forms.
- Navigate the Linux file system by changing directories.
- Manage the creation and deletion of new files and directories from within the command shell.

Background and References

Unix is a multitasking and multiuser operating system originally developed in the 1960's. Since its original creation it has received many updates and today many variants of the operating system exist all of which follow the principles of the UNIX Philosophy of minimalist OS design and modular development.

The interfaces provided by the Unix operating system have been standardized by the IEEE as the Portable Operating System Interface (POSIX). The idea is by following certain rules a software application can be executed on any POSIX compliant operating system without having to be modified. Of course, different CPU architectures would require the software to be rebuilt to execute a different instruction set architecture, but the source for the application could remain the same.

In 1991, as a personal project Linux Torvalds began working his own variant of Unix to be published as free and open software. This operating system eventually became Linux, a portmanteau of Linus and Unix. While not as popular as a desktop operating system, Linux is utilized on many server platforms, embedded systems, and is the basis for the Android operating system.

While often you hear of Linux described as an operating system, Linux itself is not an operating system, but a kernel. Many operating systems have been built using the Linux kernel. These operating systems (e.g. Ubuntu, Mint, Arch, Raspberry Pi OS) are often called 'distributions'. These operating systems bundle the Linux kernel with additional applications and services.

In this class, we will be learning and using POSIX services provided by the Linux kernel. In order to use a Linux distribution, there are several options available:

- Find a computer and install a Linux distribution on it
- Install a Linux distribution in a virtual machine such as VMWare or VirtualBox
- Use the Microsoft Windows Subsystem for Linux (WSL) which allows a Linux distribution to run within the environment of Windows.

For this class, we will be using the WSL since your MSOE issued laptops already have Windows installed and working with a virtual machine can be a bit cumbersome. But, as mentioned earlier, any application written to the POSIX interfaces will work on any variant of Unix.

Install Windows Subsystem for Linux (WSL)

The following procedure is documented in Microsoft's WSL install web page: https://docs.microsoft.com/en-us/windows/wsl/install

To install a Linux distribution in WSL, you need to perform 2 things:

1. Install the WSL variant of the Linux kernel
2. Install the distribution for the Linux operating system

Microsoft makes this easy by doing both in a single step. By default, the WSL installer installs the Ubuntu Linux distribution, which is sufficient for the work we'll be doing. If you want to look into installing other distributions, you can find information on the Microsoft WSL install web site: https://docs.microsoft.com/en-us/windows/wsl/install#ways-to-run-multiple-linux-distributions-with-wsl

To install WSL and Ubuntu:

1. Open a command window as Administrator
    1. Open the start menu and type `cmd.exe`
    2. In the menu on the right, click: Run as Administrator
2. When the command prompt opens run `wsl --install`

This might take a while as WSL installs the kernel and the Linux distribution. It will default to the Ubuntu distribution, although others are available. It will also default to WSL 2. You'll see status messages printed to the command prompt as it goes.

Once it has completed installation, you will most likely be required to reboot your PC. Once your PC restarts, you will be asked to create a username and password for your Linux distribution. This does not need to be the same as your Windows login, but it might be easier to remember if you do set it to the same. The choice is yours.

File System Setup

Once the installation completes, and you create a username and password you will see a slightly different command prompt. You are now running in the Ubuntu Linux environment. Any command you type will default to a Linux command.

In addition, the Linux subsystem has its own filesystem that is separate and distinct from the Windows filesystem. The Windows filesystem can be accessed from WSL and vice versa.

Accessing Windows Files from Linux

On Windows, the file system is typically NTFS and is organized by drive (e.g. C: or D: etc.). The file system works slightly different on Linux. Directories (i.e Folders) are 'mounted' inside each other. You can think of it like a book on a bookshelf. Each disk drive on your system is like a book and the file system as a whole is like the shelf. In Windows, each book is given a drive letter, while in Linux each book is given a directory name.

For example, to access your Windows file system from Linux you have to go where it is mounted (e.g. what shelf it is on). In WSL, the mount point for each drive is in `/mnt`.

NOTE: that Linux uses forward slash '/' to delimit a directory while Windows uses the backslash ''.

The `cd` command allows you to change directories while the `ls` command allows you to list the contents of a directory from the command line.

```
user@machine:~$ ls /mnt
c  wsl
user@machine:~$
```

This lists the contents of the `/mnt` directory. In this example, there are two things `c` which represents the C: 'drive' on Windows and another directory `wsl` which WSL uses for device mapping.

You can access any file in your Windows C: drive from Linux though `/mnt/c/`.

For example, to list the contents of your Windows desktop you'd type:

```
user@machine:~$ ls /mnt/c/Users/WINDOWSUSERID/
```

Replacing `WINDOWSUSERID` with your actual Windows username.

Accessing Linux Files from Windows

On Windows, files are typically accessed through the File Explorer. WSL makes it easy to get to the Linux file system through the File Explorer. You can actually open the Windows File Explorer directly from your Linux command line.

```
user@machine:~$ explorer.exe .
```

NOTE: it is important add the dot (.) after the command. The dot tells the Windows File Explorer to open the current directory (e.g. your Linux directory) instead of the default directory on Windows.

When the File Explorer opens, you'll notice that your Linux file system is actually mapped to a network device in Windows, most likely: `\\wsl$\Ubuntu`. If you installed a different Linux distribution, the name might be different.

On Windows, the folder for all of your user files (desktop, document, etc.) is located in `C:\Users\WINDOWSUSERID`. On most Linux distributions, your Linux user files are located in `/home/LINUXUSERID`. Using the File Explorer that is equivalent to `\\wsl$\Ubuntu\home\LINUXUSERID`.

Find this folder on your Windows File Explorer and add it to your quick access bar. It will be very helpful to do this for accessing your Linux files later.

Installing Software

We'll need to use several Linux applications in this class. Installing software in Linux is pretty easy. Ubuntu provides a command called `apt` which stands for Application Package Tool. It utilizes a central repository for commands that can be installed directly from the command line.

To install the software we'll need for this course run the following. You'll have to enter your password to grant `apt` administrator authority on your Linux installation.

```
sudo apt update
sudo apt install gcc make git
```

You'll be asked to confirm before `apt` starts downloading and installing the software.

In order to install software in a Linux distribution you need to run as the system administrator. Windows has a similar restriction for most software. On Linux, the system administrator user is called `root`. The `sudo` command stands for "switch user and do an operation". There have been several comics and memes around `root` and `sudo`.

In this case, that operation will be to run `apt` to install software. The first command updates the local database on your Ubuntu installation so that installed software will be at the most recent versions. The second command installs two (2) applications: gcc and make.

- `gcc`: is the GNU C compiler - we'll be using that to build programs
- `make` is a build tool that can be used to automate builds for projects.
- `git` is a version control tool, specifically designed for distributed and group development.

We will likely add a few more packages as we go.

Editing a Text File

Using the Windows File Explorer, create a text file in your Linux home directory. Add some text to the file, use your favorite text editor.

From the Linux command window, change to your home directory by typing `cd` without any arguments. You'll know you're in your home directory when you see a tilde (~) symbol at your command prompt.

Now, display the contents of the file with the `cat` command on Linux. For example, from the command prompt try this:

```
user@machine:~$ ls
text.txt
user@machine:~$ cat text.txt
this is some text
```

The `cat` command is short for "concatenate". It will print out the contents of a file to the command prompt.

SUBMISSION REQUIREMENT: Capture a 'screenshot' of the command(s) to print out the file.

Exploring Linux

Now that you have WSL and Ubuntu Linux set up and installed, you will explore your Linux environment. This exploration will involve experimenting with some other commands. There are some interesting and powerful things you can accomplish with relative ease.

From your Linux command window and experiment with the commands below. You can get help on commands with the `man` command (which is short for "manual"). For example, try running `man ls`.

You can search for appropriate commands with the `apropos` command. For example, try running `apropos zip`.

Some commands you should experiment with (these are all typed via the command line, or terminal):

1. view a man page: man < command > e.g. man ls
2. list a directory: ls
3. list a directory, long: ls -l
4. list a directory, long, all: ls -al
5. search for a file in the file system: find -name .profile
6. Search through a file or a through a program output: grep
7. change directory: cd < dir >
8. change directory up one level: cd ..
9. change to last directory: cd –
10. make directory: mkdir < dir >
11. remove file: rm < file >
12. remove directory: rmdir < dir >
13. copy files: cp < source > < dest >
14. display contents of a file: cat output.txt
15. zip some files into a .zip file: zip lab1 hello.cpp output.txt
16. Create an empty file: touch empty.txt

    ⓘ The last command in the list may seem silly. Why would you want to create an empty file? It is often more convenient to create the empty file, then open that file in an editor and start editing than it is to open the editor, create a new text file, add content, then fumble around with Save-As and locating the directory in which you wish the file to reside.

Some of these commands may not be installed yet, if you need to install them use `apt` as we did before. For example, to install `zip` you'd use: `sudo apt install zip`.

Creating a Development Environment

Create a Development Directory

Next you will want to create a development directory to hold the projects you will complete. To do this, create a directory (folder) in your Linux home directory for this class. It is recommended to do with within WSL, but Windows Explorer could be used for this purpose. **Avoid spaces in all directory and file names.**

Setting up your Development Environment

Since we have a bit of a "hybrid" environment, we have some interesting options available for development. One given is that we will compiling and executing our programs from within WSL. However, our WSL environment will not necessarily have a full graphical environment available. So, we will likely want to edit our source files via a Windows program. So, we may need some setup on both the Windows and WSL sides.

WINDOWS:

I highly recommend that you use VSCode https://code.visualstudio.com/. VSCode is an basic IDE created by Microsoft with lots of configuration and plugin options. VSCode is very

WSL aware and can be invoked from the WSL command line via the command `code <filename to edit>`. Through various plugins, VSCode can be configured into a fully-featured IDE, but we will not necessarily be doing that in this course.

However, you can really use any Windows-based editor that you like. A couple of other options:

- CLion - This is a C/C++ editor created by JetBrains and is similar in look and feel to IntelliJ.
  - You can download CLion from JetBrains web site: https://www.jetbrains.com/clion/
  - There is no community version, but you can get a free license if you register with JetBrains as a student: https://www.jetbrains.com/community/education/#students
- Notepad++ https://notepad-plus-plus.org/ is a simple text editor with text highlighting and has project file management as well.

LINUX:

While for this class, it will be easiest to edit files from Windows and then build and execute them from your Linux command line, if you want to experiment with editors on Linux there are many out there. Unfortunately, running graphical Linux applications in WSL doesn't work (at least not with additional setup). The following examples are editors that can be used via the command line. They have no mouse support, but can be useful for editing text files in a hurry:

- nano is a non-graphical text editor. It uses CTRL shortcuts for many operations. It should already be installed on your Ubuntu system. More information on nano can be found here: https://www.nano-editor.org/
- vim is a non-graphical text editor that you might have used if you use Git Bash on Windows. It is installed by default on your WSL Ubuntu installation. It should already be installed on your Ubuntu system. Even if it does not become your primary coding editor, basic vim skills are useful as it is quite universal. More information on vim can be found here: https://www.vim.org/
- Emacs is another non-graphical text editor. It can be installed by running: `sudo apt install emacs`. More information on Emacs can be found here: https://www.gnu.org/software/emacs/. > What ChatGPT had to say about Emacs: "Ah, Emacs users—the digital equivalent of artisanal coffee brewers who insist on hand-grinding their beans. These are the folks who revel in the arcane, who find joy in a labyrinth of keybindings that would make a concert pianist sweat. They're not just writing code; they're crafting an experience, one Ctrl-Alt-Shift command at a time. Sure, they could opt for a sleeker, more modern text editor that doesn't require a Ph.D. in Emacs Lisp to customize. But where's the fun in that? Why simply get the job done when you can also get a workout for your fingers and a test for your memory? Emacs users are the type who believe that efficiency is overrated. After all, if you finish your tasks too quickly, what will you do with all that extra time?

Probably just waste it learning another obscure Emacs command. Ah, the circle of life."

NOTE: If you wish to enter the long-running debate of 'emacs' vs 'vim' (https://en.wikipedia.org/wiki/Editor_war feel free to try out one (or both) of those editors.

Building and Running a Program

In this section you will be compiling, running, and modifying a program.

Create a source file (hello.c) and add the following to that file.

```c
/*********************************
 * hello.c
 *
 * A hello world program in c
 *********************************/
 #include <stdio.h>
 int main (int argc, char* argv[])
 {
     // Print a message to the user
     printf("Hello world!\n");

     // Return to exit the program
     return 0;
}
```

In order to execute this source code, it must be compiled. Like 'Java', compiling a 'C' program translates the source code into a format that is executable. One difference between 'Java' and 'C' code is that 'C' is translated directly into an executable and can be run without the need for a virtual machine interpreter.

SUBMISSION REQUIREMENT: As you run the commands to compile and run your program, make sure you answer the questions associated with each step.

At the command shell, issue the command:

```
gcc hello.c
```

Run the command to perform a long listing (listing with details) of the files that now exist in the directory with `hello.c`.

- What files do you see? Include a 'screenshot' of the listing output.
- How big is each file? How did you determine the size(s)?
- Research the compiler `gcc`, what file contains the executable generated from compiling `hello.c`?

Now run the generated compiler output. Include a 'screenshot' of the output in your submission.

Invoking gcc as above will "build" the application from source. Building actually comprises several steps including compilation and linking. Intermediate files are created but removed by default when gcc finishes, leaving only the executable. We can supply options to gcc by adding arguments to the command line.

Some options you should try: `-S -c -Wall -Wextra -std=c89`

Try each of these options and by observation and research determine how each option changes the behavior of gcc. Describe briefly in your submission.

By now you should know that the name of the executable gcc builds is `a.out` by default. Probably not terribly useful. How can you change this behavior?

To run the program you will have to type at the prompt, $ ./a.out

Deliverables

As noted at the beginning of this document, collect various requested screenshots and commentary [in a Word document]. Submit a pdf of that document to the Week 1 Lab Assignment in Canvas. It is intended that you be able to submit this by the end of the lab period.