Lab 1
Introduction to Linux and Makefiles

Prof. Kredo

Due: Start of lab Friday, January 30

Name:	
Name:	

Introduction

In this lab you will accomplish several goals:

- Gain a basic working knowledge of Linux systems
- Learn how to develop programs in a Linux environment

Work individually or in pairs for this lab using the equipment at your desk. Distribute the work evenly to make sure both group members know the material, as you will be required to know the material for evaluation.

1 Prelab [10 Points]

Perform the prelab reading and answer the questions before coming to lab.

1.1 Reading

If you are unfamiliar with UNIX or Linux systems, read the Introduction[1], Tutorial One[2], and Tutorial Two[3] of the UNIX Tutorial for Beginners[4]. You may want to take notes and bring them to lab to help you complete your assignment.

[1]http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html
[2]http://www.ee.surrey.ac.uk/Teaching/Unix/unix1.html
[3]http://www.ee.surrey.ac.uk/Teaching/Unix/unix2.html
[4]http://www.ee.surrey.ac.uk/Teaching/Unix/

Questions

Answer these questions before lab.

1. Write the command to change directory to `/home/student`, independent of the current directory.

2. Write the sequence of commands you would use to accomplish the following tasks: 1) create a directory named `temp`, 2) copy the file `/etc/hosts` into your new directory, 3) examine the contents of the file you copied, 4) delete the file you copied, and 5) delete the directory you created.

2 Introduction to Linux [40 Points]

You will now begin working with a Linux system, which you will use for all labs and programs. This lab is intended to get you started using Linux, but there is much more to learn. Turn on your host and boot into Fedora, our Linux distribution. If your system is on, but currently running Windows, reboot your host.

2.1 Login

Begin by logging in to your host using the following information.

Username	student
Password	chico

Open a command line terminal (henceforth simply called terminal) by clicking on the monitor icon on the bottom panel.

2.2 Networking Configuration

Configure the networking interface for your host by typing in the following commands. Don't worry if you don't know what they mean as this will be explained later in the semester. Replace `<your ip>` with the address listed in the table at the end of the lab handout. The `ip` command configures and manages the IP networking system on Linux.

```
student@host ~ $ sudo ip address add <your ip>/16 dev eth0 broadcast +
student@host ~ $ sudo ip link set dev eth0 up
student@host ~ $ sudo ip route add default via 10.11.0.1
```

Check your settings by running the following commands. You should see output resembling, but *not identical* to:

```
student@host ~ $ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether <your mac> brd ff:ff:ff:ff:ff:ff
    inet <your ip>/16 brd 10.11.255.255 scope global eth0
    inet6 <ipv6>/64 scope link
       valid_lft forever preferred_lft forever
student@host ~ $ ip route
default via 10.11.0.1 dev eth0
10.11.0.0/16 dev eth0  proto kernel  scope link  src <your ip>  metric 2
```

Use a cable to connect your host to the external (EXT) network. Verify you have joined the network by trying to access the Internet with a web browser. If you are unable to access the Internet, check that you performed all the previous steps correctly. Once connected, you can access the UNIX Tutorial for help completing this lab.

2.3 Commands

Try out the commands you learned from the UNIX tutorial by performing performing the following steps and answering the questions below.

1. Enter the commands from Prelab Question 2. Any problems? Write any corrected commands below.

2. Copy the `/etc/services` file to your home directory. How many lines does the file contain? How many lines are in the `/etc/hosts` file?

3. Examine the `services` file you copied. What do you think is the purpose of this file?

2.4 Manual (man) Pages

Manual pages (man pages) are an important source of information in Linux/UNIX systems. They contain documentation about programming APIs, command syntax, system details, and other useful information. You access manual pages with the `man` command. Enter `man man` in the terminal to learn about man pages.

man pages are divided into sections to help organize this vast wealth of information. Sometimes a particular name belongs to multiple objects, such as a command and an API description. `rmdir` is one example, with a man page in section 1 for the command line tool and a man page in section 2 for the system call. To differentiate these pages, you'll often see the appropriate section number provided with a name. For example, `rmdir(1)` refers to user command man page in section 1 and `rmdir(2)` refers to the system call man page in section 2. Man pages often have examples at the end, so be sure to scan them throughly.

Type `man ip` in the terminal. Briefly describe the page that you see. To which manual section does it belong?

If you want to see all the man pages that belong to a name, use the `-a` option. Type `man -a ip` in the terminal. How many sections have a man page for the name `ip`? To which manual sections do they belong?

If you want to see the man page for a name from a particular section, you can include the section number before the name. For example, `man 0 ip` would display only the man page for `ip` from section 0 (there isn't really a section 0). What command would you type to only see man page describing the IPv4 protocol implementation on your Linux system?

`apropos(1)` can be a helpful command when you are looking for a man page, but don't know the exact name to use. Investigate `apropos` if you desire.

3 Linux Development [50 Points]

Much of this class involves developing C or C++ programs in the Linux environment. This section will help you get started doing this.

3.1 Editors

There are several text editors available to create and modify programs. Command line editors include `vim`, `emacs`, and `nano`. Graphical editors include Leafpad and scite. Try out the available text editors and pick the one that works best for you. If you have a favorite editor that isn't installed, ask to see if it can be added.

Use a text editor to create a simple Hello World program in C or C++.

3.2 `gcc` and `g++`

Linux systems often come with a free compiler suite called the GNU Compiler Collection (gcc). (gcc is both the name of the C compiler command and the compiler suite.) The two compilers of interest in this class are `gcc` for C programs and `g++` for C++ programs.

Compile your Hello World program by entering the appropriate command `gcc hello.c` or `g++ hello.cc`. This command produces an executable file (program) named `a.out`. Run your program now.

Some useful arguments to `gcc` and `g++`:

`-o`	Define the executable filename to use instead of `a.out`.
`-Wall`	Have the compiler warn about all code issues. All your programs must compile cleanly with this option set, so get in the practice of using it.

What command would: compile your Hello World program while providing warnings about all code issues and name the executable `hello`?

Enter this command in the terminal. Fix any errors or warnings you encounter until your program compiles and runs cleanly (no warnings or errors produced when you compile and run the program).

3.3 Makefiles

Typing the commands to compile your program every time you make a change becomes tedious quickly. Thankfully, Makefiles simplify the process. Makefiles are regular text files that describe how to convert input files to output files. They are typically used to control the compilation of software, but can be used any time you need to process files. Below is a makefile for a sample Hello World program.

```
#### Makefile start ####
a.out: hello.c
    g++ hello.c
#### Makefile end ####
```

The first line creates a rule that tells how an input file or collection of input files relate to an output file. The line states that `hello.c` needs to be processed to create the output file `a.out`. The following lines detail how the input files are processed to produce the output files. On each of the following lines you provide the exact commands and arguments you would type in the terminal to produce the output. In this case, we produce the executable file `a.out` by compiling the source file `hello.c` with the command `g++ hello.c`. Note that the commands following a rule must be tab indented, not space indented.

Create a makefile for your Hello World program that uses the `-o` and `-Wall` arguments to `gcc` or `g++`. Provide your Makefile below.

You use Makefiles by running the command `make`, which looks for a file named `Makefile` or `makefile` and performs the necessary steps outlined by the rules inside. Delete any executable files in your directory and then run `make` in the terminal. What happens when you run `make` several times?

Edit your source file (you could just add or change a comment). Run `make` again. What happens?

Delete your executable file (NOT your source file!). Run `make` again. What happens?

Based on the last few questions, can you explain how `make` decides when to run the rule commands?

Submit your completed lab handout before the next lab.

4 Lab 1 Addresses

	Host Addresses	
Desk	**PC 1**	PC 2
A	10.11.50.101	10.11.50.201
B	10.11.50.102	10.11.50.202
C	10.11.50.103	10.11.50.203
D	10.11.50.104	10.11.50.204
E	10.11.50.105	10.11.50.205
F	10.11.50.106	10.11.50.206
G	10.11.50.107	10.11.50.207
H	10.11.50.108	10.11.50.208
I	10.11.50.109	10.11.50.209
J	10.11.50.110	10.11.50.210
K	10.11.50.111	10.11.50.211
L	10.11.50.112	10.11.50.212
M	10.11.50.113	10.11.50.213
N	10.11.50.114	10.11.50.214
O	10.11.50.115	10.11.50.215