CS3841 Lab 1
Getting used to Linux

Dr. Walter Schilling

Due: September 13, 2012 23:00 CDT

1 Introduction

The purpose of this lab is to set up the Linux Virtual machine on your laptop as well as investigate the development environment.

2 Lab Objectives

1. Demonstrate an ability to use a Linux shell.
2. Use the man command to obtain documentation about Linux commands.
3. Explain how to list the contents of a directory in multiple forms.
4. Navigate the Linux file system by changing directories.
5. Manage the creation and deletion of new files and directories from within the command shell.
6. Capture the output of a Linux program executing to a file.
7. Manage the creation and extraction of zip files and tarballs using the command shell.
8. Construct a makefile which will automatically generate the project as well as allow for the clean building of source code.
9. Debug code in Linux using GDB from a command line shell.

3 Laboratory Preparation

This section explains the general steps required to install and configure your virtual machine. In all cases, it is recommended that you complete these steps with your machine connected to a wired connection. These steps may involve large downloads that are significantly faster using a wired connection over a wireless connection.

3.1 Install VirtualBox

Prior to lab, you will need to install the VirtualBox environment as well as the Linux image onto your laptop. To start, install the VirtualBox software from the course website. This simply involves downloading and installing exe file from the course website. The program can be installed to any location you desire, but be consistent.

Next, you will need to download the Linux image from the course website onto your laptop. The files you will need are all zipped into a single file which you will need to extract. You can copy them to any location on your machine. However, it is recommended that you place them on the D drive of your machine.

At this point, you will need to create a directory on your D drive called "cs3841Code". This is a directory which will be accessible both from within the virtual machine as well as on your local machine, enabling you to transfer data and files between Windows and Linux.

Once you have copied your files, run Virtual Box. Select "Machine" and "add" and browse to the location of the files you copied. You should be able to select a file called "UBUNTU Virtual Machine for OS 2011", and it should be an instance of UBUNTU which is powered off. Clicking on the "Start" arrow should start the virtual machine running. Clicking on Play virtual machine will start the program. This will open the virtual machine which will begin to boot just like an actual PC would. The Windows firewall may pop a security alert to which you should go ahead and unblock. Eventually you will be presented with a login screen to which you may login with username: student, password: cs3841.

4 Explore Linux

I suggest you explore the UNIX shell a bit. There are some interesting and powerful things you can accomplish with relative ease. You can get help on commands with the man command. For example, try man ls. You can search for appropriate commands with the apropos command. For example, try apropos zip. You may wish to take a look at "Unix is a Four Letter Word", a unix manual that Dr. Taylor wrote some years ago. He presents a brief tutorial on how to use the most common commands.

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

1. list a directory: ls
2. list a directory, long: ls -l
3. list a directory, long, all: ls -al
4. view a man page: man < *command* > e.g. man ls
5. Search through a file or a through a program output: grep
6. change directory: cd < *dir* >
7. change directory up one level: cd ..
8. change to last directory: cd -
9. make directory: mkdir < *dir* >
10. remove file: rm < $file$ >
11. remove directory: rmdir < *dir* >
12. copy files: cp < *source* > < *dest* >
13. capture output of a program: ./hello > output.txt
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 a tarball: tar -cf new_tar_file.tar *
17. Extract a tarball: tar -xvf new_tar_file.tar

5 Making a development directory

Next you will want to create a development directory to hold the lab projects you complete. To do this, change into the directory "/media/sf_cs3841Code". This is a shared directory which you will be able to access from both windows and Linux. Create a directory called "dev" and underneath that directory a directory "lab1".

6 Build a Program

As a start, try compiling the following program (copy/paste to a file named hello.c):

```
/*********************************
 * hello.c
 * Written by: H. Welch - 11/26/2006
 * Modified W. Schilling - 8/15/2009
 *
 *
 * Demonstrate basic C-program along with
 * system call requiring struct and pointer
 * manipulation.
 *********************************/

#include <stdio.h>
#include <sys/utsname.h>
#include <stdlib.h>

int main (int argc, char* argv[])
{
  /* Get system information using the uts system interface.*/
  /* Declare a buffer to store information about the system. */
  struct utsname buf;

  /* Declare a pointer to the user information. */
  char *usr;

  /* Populate the buffer with data from the system. */
  uname(&buf);

  /* Get information about the user from the system. */
  usr=getenv("USER");

  /* Print out system information to the console. */
  printf("Hello %s:%s:%s:%s:z%s\n", buf.sysname, buf.nodename,
         buf.release,buf.version,buf.machine);
  printf("The size of the UTS structure is %d.\n", sizeof(buf));

  /* Print out the user information if the pointer is not NULL. */
  if (usr != NULL)
  {
    printf("%s\n",usr);
  }
  else
  {
    printf("User information not returned by the operating system.");
  }

  /* Return to O/S */
  return 0;
}
```

At the command shell, issue the command

```
gcc -E hello.c | more
```

This command will preprocess the source code and pipe the results to the console. What is the length of the sysname element of the utsname structure?

Now issue the commands

```
gcc -S hello.c
```

```
cat hello.s | more
```

This will compile the given code, resulting in the creation of a .s file containing the assembly code for the compiler. Note the align command. This aligns the start of the definition on a 4 byte boundary. Why does this happen? Of the assembly code, which seems to be the most common instructions?

Now issue the commands

```
gcc -c hello.c
ls -al
```

The first command creates an object file for the hello.c source code. This is code which contains assembled instructions that can later be linked by the linker. What extension is given to this file?

Lastly, issue the command

```
gcc -o hello hello.c
```

This will perform all stages of compilation, writing the given file to the program hello. Issue the command

```
./hello.
```

Looking at the total structure size by adding all of the element array lengths together, does the total size reported to the screen make sense?

7 Piping the output

Now that you have reached this stage, run the program and pipe the output to a file. When we pipe the output to a file, any text which is written to the console will be stored in a file. We do this by using the > and | characters. The details of doing this have been left up to you. However, a good explanation is available at `http://www.linuxjournal.com/article/2156`.

8 Creating a makefile

Now that you have defined your program, issuing each of these commands becomes a bit tedious. We want to define a makefile which will do the following:

1. Determine the dependencies for the given files
2. Generate the required preprocessed source code
3. Compile the code
4. Link the files
5. Generate an executable
6. Allow the user to perform a clean build in which all generated artifacts are regenerated.

The following provides a template makefile which will do all of these things. Use it to compile your hello.c program. To do so, list the names of the object files that you are going to create (i.e. hello.o) in the OBJS. In the executable segment, name the executable you wish to create (i.e. hello). **Note that lines which are indented must use tabs, not spaces.**

```
SHELL = /bin/sh
SRCDIR = .
CC = gcc
YACC = bison -y
CDEBUG = -g
COMPLIANCE_FLAGS =
CFLAGS = $(COMPLIANCE_FLAGS) $(CDEBUG) -I. -I$(SRCDIR)
LDFLAGS = -g

#############################################################################################################
# List your sources here.
SOURCES =
#############################################################################################################

#############################################################################################################
# list the name of your output program here.
EXECUTABLE =
#############################################################################################################
# Create the names of the object files (each .c file becomes a .o file)
OBJS = $(patsubst %.c, %.o, $(SOURCES))

include $(SOURCES:.c=.d)

all : $(OBJS) $(EXECUTABLE)

$(EXECUTABLE) : $(OBJS)
        $(CC) -o $(EXECUTABLE)  $(OBJS)

%.o : %.c #Defines how to translate a single c file into an object file.
        echo compiling $<
        echo $(CC) $(CFLAGS) -c $<
        $(CC) $(CFLAGS) -E $< > $<.preout
        $(CC) $(CFLAGS) -S $<
        $(CC) $(CFLAGS) -c $<
        echo done compiling $<

%.d : %.c #Defines how to generate the dependencies for the given files.  -M gcc option generates dep
        @set -e; rm -f $@; \
        $(CC) $(COMPLIANCE_FLAGS ) -M $< > $@.$$$$; \
        sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
         rm -f $@.$$$$

clean :   # Delete any and all artifacts from the build.  The only thing which is kept is the source
        rm -f *.o
        rm -f *.preout
        rm -f *.s
    rm -f *.S
        rm -f *d
        rm -f $(EXECUTABLE)
```

Note that lines must be indented with tab characters, not spaces. Now that you have created this makefile, enter "make clean" and "make all" to compile your code. The clean option will do what is called a clean build. In essence, everything except for the source code will be deleted. This prevents artifacts from previous builds from

interfering with your project. The make all will build all of the pieces of this project, namely all of the object files as well as the program.

Further details on makefiles are available on the internet. A good site is `http://www.up.ac.za/organizations/societ` and the ultimate source is the FSF documentation on make, available at `http://www.gnu.org/software/make/manual/`.

9 Using the GDB Debugger

Before proceeding, watch the two quick tutorials online on using the gdb debugger. The tutorial will explain to you how to add breakpoints, step your code, and do other things which are necessary to debug in a unix environment.

Once you have watched this tutorial, download the tar file file lab1BrokenCode.tar.gz. This project is a simple implementation of a calculator that allows you to type basic expressions on the keyboard and evaluate them. For example, typing "$2.0 + 1 =$" should result in the value of 4 being printed. However, the program is broken in a few subtle ways. Using gdb, correct the program so that everything works properly. You'll most likely need to set breakpoints, watch variables, and do other things in order to ensure correct operation.

Once you have the program working properly, create a tar file of your corrected code for submission.

10 Deliverables

1. Tar file including all of your source code, makefile, etc.
2. Lab report, submitted in pdf format, with the following
   (a) Name, date, and lab title
   (b) Answers to questions posed within the report.
   (c) Problems encountered with the lab and how you solved them
   (d) Screen capture of your program running
   (e) Program output piped to a file.
   (f) Screen capture showing your debugging session in gdb.
   (g) Description of what you learned
   (h) Conclusions

Deliverables are to be uploaded to the course website.