Linux VM Components and Installation

Your final assignment for COP4600 is going to be writing an actual, if simple, device driver for the Linux operating system. To do so, you will need to complete several preliminary tasks. You need:

- A virtual machine (VM) system so that we can all work on the same platform
- A copy of Linux to run In a VM
- The necessary tools for Linux to perform kernel development

If at any point during this you get lost or have too much trouble, don't be afraid to restart it! Once you've got the big pieces downloaded, none of these steps take more than a few minutes.

Step 1: Getting the Pieces

VirtualBox is a program from Oracle Corporation that allows you to easily run other operating systems on your PC, Mac or Linux desktop. It is not the best-performing VM solution available, but it is by far the easiest to set up and configure.

Ubuntu is a Debian-derived, GNU-based Linux distribution that is one of the easiest to set up and configure.

You are *required* to use *this software* for this assignment. If you use a different VM system or version of Linux, the teaching assistants will be instructed not to assist you until you obtain and install this software.

1. Download VirtualBox for your preferred platform from: https://www.virtualbox.org/wiki/Downloads.
2. Download the ISO file for the desktop **32-bit version** of Ubuntu 14.04.4 from http://www.ubuntu.com/download/desktop.
    - It is important that you download the **32-bit version.**
    - The filename before extension should end in **i386.**

Do not obtain this software from anywhere else. There are innumerable sites waiting to "help" you install Linux; many of them bundle advertising software, or worse.

Step 2: Create the Virtual Machine

1. Install VirtualBox on your computer by running the file you downloaded for it and following the prompts.
2. Start VirtualBox.
3. Follow the prompts to create a virtual machine for 32-bit Ubuntu.
    a. Give the VM at least 1GB of RAM and 20GB of hard disk space.
4. Change the VM's settings, and point its virtual optical drive at the Ubuntu ISO file you downloaded.
5. Start the VM and allow the Ubuntu installer image to run.
    a. Choose to download updates while installing.
    b. Choose to erase the disk and install Ubuntu.
    c. Do not bother with encryption options (today).
    d. Otherwise, follow all prompts normally.
6. At the end of the installation, restart the VM.

Step 3: Configure the Virtual Machine

This step is unavoidably messy. We need to get Ubuntu to realize that it's in a virtual machine and properly integrate with it. Otherwise, you won't be able to resize the VM window, copy and paste won't work from the VM to outside it, and several other annoyances.

1. If you don't still have the VM running from Step 2, start it.
2. Log in to the VM.
3. Dismount any virtual CD currently mounted by scrolling the taskbar on the left down to the picture of a disc, right-clicking it, and selecting Eject.
4. Open a command-line terminal window by doing the following:
    a. Click on the Ubuntu icon in the top-left.
    b. Click on the Applications icon at the bottom (next to the Home icon).
    c. Type "terminal".
    d. Click on the **Terminal** application that is already installed.
5. From the terminal, run the following commands. You will need to enter your password at least once.
    ```
    sudo apt-get install build-essential module-assistant
    sudo m-a prepare
    ```
6. From the Devices menu, select **Insert Guest Additions CD Image…**
7. Allow the VirtualBox Guest Additions installation to automatically run.
8. Restart the VM.

Congratulations; you have a working Linux VM. Log in to it and make sure it works: move some files around and browse the Web a little (Firefox is right there on the taskbar!)

Step 4: Compile Something

This will be familiar if you've ever programmed in a Linux environment before. We're going to make sure **gcc** is working.

1. Load the text editor by doing the following:
    a. Click on the Ubuntu icon in the top-left.
    b. Click on the Applications icon at the bottom (next to the Home icon).
    c. Type "editor".
    d. Click on the **Text Editor** application that's already installed.
2. Write a tiny C program. Hello World works just fine.
3. Save it in your home directory with a **.c** extension.
4. Open a terminal window.
5. From the terminal, compile and run the program. If you named it **hello.c**, do this:
    ```
    gcc -o hello hello.c
    ./hello
    ```

Congratulations; you've compiled and run a Linux program on your Linux VM.

Step 5: Create a Makefile

You can skip this step if you're an experienced programmer in UNIX-like operating systems.

Makefiles are the UNIX-like approach to programming project files. A makefile is simply a script that contains the structure of a project, and the files that need to be created.

Make a Makefile

Create a text file called *Makefile* in your home directory, with the following lines:

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

*It's important that the second line be indented with a tab character, not with spaces - otherwise, make won't work. The **g** in **gcc** may be under the **e** in **hello.c** instead of under the space as it appears here - that's fine, as long as you're using a tab character.*

Try the Makefile

From the command line, run `make`. If you haven't changed anything else since Step 3, you'll get:

```
make: `hello' is up to date.
```

Now, edit **hello.c**, change something, and run `make` again. This time, **hello** will be recreated, and make will tell you how it's doing the job:

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

This is normal, because of what the makefile is really doing.

Understanding Makefiles

Makefiles are entirely about *dependencies* and *targets*, and this simplest possible makefile translates to:

*"The target **hello** is dependent on **hello.c**. If **hello** doesn't exist, or **hello.c** has been updated since **hello** was updated, you can recreate **hello** by running* `gcc -o hello hello.c`*."*

Think about this for a few minutes and you'll see that makefiles are as powerful as any other project mechanism - if not, sadly, as intuitive as some.

More Makefiles

If we want to split the compilation and linking steps, that's simple too:

```
hello: hello.o
	gcc -o hello hello.o

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

This sets up two targets: **hello**, which is dependent on **hello.o**, and **hello.o**, which is in turn dependent on **hello.c**. It gives **make** instructions for creating both of them. Modify **hello.c**, and **make** will solve the dependency graph and recreate both targets.

Step 6: Compile a Kernel Module

The bad news is that building a kernel module is *really complicated*. The good news is that the kernel developers have already done most of the work for you. Open up a terminal and let's get to work.

Preliminaries

Create a new directory in your home directory:

```
mkdir modtest
```

Find out which version of the Linux kernel you're running:

```
uname -r
```

The Simplest Kernel Module

Now create the following files in your **modtest** directory:

tinymod.c:

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void) {
	printk(KERN_INFO "Installing module.\n");
	return 0;
}

void cleanup_module(void) {
	printk(KERN_INFO "Removing module.\n");
}
```

makefile:

```
obj-m += tinymod.o

all:
	make -C /lib/modules/4.2.0-27-generic/build M=/home/mgerber/modtest modules
```

*Change the kernel version in the **/lib/modules** path if you need to, and change **mgerber** to your username.*

This makefile is *calling another makefile*. That other makefile is part of the Linux kernel development tool chain, and is there to make your life a lot easier when building simple kernel modules like this. Actually doing all the work to configure a build environment for a kernel module is far, *far* beyond the scope of a course like this one.

Testing the Module

Build the module by running `make`, and install it by running:

```
sudo insmod tinymod.ko
```

This doesn't tell you anything interesting, because `printk` isn't going to your terminal. Run `dmesg` to see what's going on inside the kernel. Sure enough, the last few lines will be complaints about your module not having a license, followed by the "Installing module" message!

Now remove the module by typing…

```
sudo rmmod tinymod
```

…and run `dmesg` again to see the removal message.

Congratulations: You've created an amazingly dangerous version of Hello World.