For this lab, you are not required to submit your answers to the questions. The questions, however, may be asked on subsequent quzzes and you are required to remember roughly what you did in completing this lab.

1 Installing Debian Linux

In order to get linux running on the lab computers, we are going to need to install it on a computer. Installing is actually a relatively simple task, but is completely documented below.

1. Boot the computer with the install CD in the drive. It should boot from the CD and ask you to press F1 for help or enter to boot. We want to run with a 2.6 kernel, so type "linux26" and then press enter.
2. Choose an install language of english.
3. We live in Canada. Choose Canada as a country.
4. Choose an American English keymap. The keymap is used to determine how to interpret keys pressed on the keyboard into actual letters.
5. The install will fail to detect the network card in the computers. This is ok, as we won't be using the network card for any of the labs. When it asks you if you wish to load network drivers off a floppy, choose no. It will give you a few errors and then you will be at a menu from which you can continue the install.
6. Choose the *Detect Hardware* option from the install menu.
7. The computers in the lab do not have a PCMCIA interface. Choose no when asked whether PC card services should be started.
8. Debian will start the partitioner in order to partition the hard drive. You can safely erase the entire disk.
9. When asked how to partition the disks, choose *Desktop Machine*
10. The install will display the partition setup it has automatically calculated for install. Since we do not have ext3 available as a filesystem type, you need to manually select each partition that has ext3 as a filesystem type and change the filesystem type to ext2. While you are in the menu for each partition, make sure that all partitions are set to be formatted as part of the install.
11. When you are finished changing the filesystem types for all partitions that have ext3 as a filesystem type, choose *Finish partitioning and write changes to disk*

12. The installer will set up the partitions, format the filesystems, and install the base system at this point.

13. After installing the base system, you will be asked about GRUB. GRUB is the boot-loader designed to boot the debian installation from the hard disk. There are no othe operating systems on the disk, so it is safe to say that GRUB can be installed on the master boot record. Recall from class that the MBR contains a small program designed to boot the operating system during the boot process.

14. The installer will eject the CD. Remove it and acknowledge the attempt by the installer to reboot the system.

15. The system will reboot and you will be presented with a series of configuration screens. After the system has booted off of the hard drive, put the CD back into the drive. You will need it later on in the install process.

16. The clock on these systems is not set to GMT.

17. We are in the eastern time zone.

18. For these systems, the root password should be set to `comp3000`

19. An additional user called `Student` should be created with a password of `student`

20. The system name can be left at debian

21. You are not connected to the internet. Do not use PPP to install the system.

22. There is not a second Debian CD. Choose no when asked if you wish to scan another CD.

23. Debian will inform you it can't access security updates. Usually, these security updates contain patches to software. Since these computers are not on the internet, it is not critical that the security updates be installed. If you were installing Debian outside the lab, however, this would be a different story.

24. For the collections of software currently installed, leave the selection with none of them selected and choose ok.

25. Exim, the mail server, should be configured for local delivery only.

26. All mail should be delivered to `student`.

27. At this point, the base install is finished. You will be presented with a login prompt. Log in as `root` with a password of `comp3000`.

28. The base install did not install all of the elements we are going to need for our system. at the prompt type `apt-get install kde-devel`. This will install the software development programs from the KDE project. For the most part, you can accept the default options, but note that we use letter size paper in Canada and not a4.

29. Install the manpages for development. `apt-get install manpages-dev`. These contain the documentation on *fork* and *exec* that you will need in order to complete the lab.

30. Install the X server (`apt-get install x-window-system`). This is used as a graphical interface under Linux. Allow it to autodetect the video hardware. The defaults on all options can be used. When it comes to asking what video modes you wish to use, select 1280x960 and all lower resolution modes.

31. If you wish to install vim or emacs21, use the same process of `apt-get install ...` to install these editors. This is an optional step, and if you don't know what these editors are, then don't worry.

32. We want to play with two utility programs, strace and ltrace, so install them with `apt-get install strace ltrace`.

33. Install KDM (`apt-get install kdm`), the K display manager. This will present you with a gui where you can enter your username and password. You should use the KDM and not XDM when presented with the option.

34. Copy over the files required for lab 2. To do this, you must first mount the CD (using `mount /cdrom`). The lab 2 files are located on the CD in a tar file. Untar the file into the `/usr/src` directory by using the `cd` command to change to the `/usr/src` directory and then typing `tar -xvf /cdrom/lab2.tar` to extract the files. Finish off by unmounting the CD (using `umount /cdrom`).

35. Start KDM. To do this, type `/etc/init.d/kdm start` at the prompt. This will result a graphical login prompt being displayed on the screen.

36. We still need to log out of the console. Switch back to the console by using *CTRL-ALT-F1*, then log out by typing exit at the prompt. To get back to the graphical login screen, type *CTRL-ALT-F7*

2 Fork

In this section of the lab, you will be introduced to the fork system call.

1. Log in as `student` (password `student`)

2. Pull up a terminal window. This can be done by clicking on the icon depicting a monitor with >_ in it. From the terminal window, use the manpages to gain an understanding of how the fork system call works. Do this by using the command `man fork`. Later on you will use `man` to learn about other Linux commands.

3. Copy over fork.c from `/usr/src/lab2/fork.c` into your home directory.

4. Compile fork.c using gcc (`gcc fork.c`). It will produce an executable called a.out. To run a.out, prepend a *./* telling the shell that you wish to run the file located in the current directory. What does running *a.out* display?

5. Make a Zombie process using fork.c. To see if you have successfully created a zombie process, use the `ps a` command. You will have to examine the process listing after the child has finished its processing, but before the parent completes. Use appropriate programming techniques to make this gap in finishing times large enough that another terminal window can be used to run the `ps a` command. What is displayed after *[a.out]* on the process listing? What is listed under the STAT column?

6. Modify the program so that the parent prints its text first. To do this, you will probably need to use the `man` command to learn the meaning of *wait*. You'll also need some way to force a context switch—try using *sleep* (you'll have to run `man 3 sleep` to see the right manual page).

7. While there are a number of ways of making a process give up its CPU time, an alternate method of ensuring correct execution flow is through the use of interprocess communication. Pipes (`man pipe`) are one method of communicating between a parent and child process. Using pipes as well as `read` and `write`, re-implement the previous step. The parent should wait until a byte has been written to the pipe before printing to the terminal, and the child should wait until after it prints before sending a byte on the pipe.

8. What are the values of x, y, and z when the parent and child processes exit? Add `printf` statements to see if you are right.

9. Run `strace -fo fork-syscalls.log ./a.out`. Look at *fork-syscalls.log* using *more*. Are there more system calls than you would expect?

10. Recompile `fork.c` using the command `gcc -static fork.c -o fork-static`. Run *fork-static* using *strace*. How does the output compare with the previous strace output?

11. Look at the manual page for *ltrace*. Run it on both compiled versions of `fork.c`. On one of the versions the command doesn't work—why? Hint: check out the `ldd` command.

3 Exec

In the previous section, we concentrated on the fork system call. Now we will focus on exec.

1. Copy over the exec.c file from `/usr/src/lab2/exec.c`
2. Build the sample exec.c file using gcc and run it. The program requires the name of an alternate executable to run with the exec system call. For our purposes, use echo. Try running `./a.out echo Hello`. What is printed on the screen?
3. Examine the exec.c file in a text editor. Is the line "Is this line printed?" printed on the screen when the program is run? Why or why not?
4. As an example of how we can use exec, modify argv[2] so that every character in the string is incremented by one. Recompile the exec.c file and run the result with `./a.out echo Hello`. Now run `echo Hello`. Verify that the input to the echo command has indeed been changed.
5. Modify the exec.c file so that the program specified by *argv[1]* is run on both the incremented and non-incremented forms of *argv[2]* (hint, in order to do this, you are going to have to *exec argv[1]* twice, requiring a fork). Display the non-incremented output first. How would you modify your program to display the incremented output first?
6. Optional task: modify the program to use just `execve`. Pay special attention to the somewhat unusual argument declaration!