CS2106: Operating Systems
AY20/21 Semester 1
Introduction to OS labs

Section 1. Introduction

If you used Unix or one of the Unix variants before, then most of the introductions below should be familiar to you. Feel free to skip all the parts which you already understood. For example, if you used SoC's unix server ***sunfire*** (by SSH into sunfire.comp.nus.edu.sg) before in other modules, then the differences are quite minor. Please note that **SunOS** (a Unix variant by Oracle) is used on the sunfire servers while **Linux** is used in our labs.

1.1 Getting access to Linux

Here's a couple of ways to get access to a Linux installation. Take note of **option** 1, as we will use it as the **standard test platform** for evaluating your lab submissions.

Option	Details
1	**Use SoC Compute Cluster** **Requirements:** • You need SoC account to use these nodes. Please go to https://mysoc.nus.edu.sg/~newacct/ to (re)activate your account. • You need to enable **SoC Compute Cluster.** Please go to https://mysoc.nus.edu.sg/~myacct/services.cgi to enable the facility. **Usage:** **1.** Use any SSH Client to connect to sunfire server using your SoC Credentials. **2.** With a successful login, ssh over to one of the Computer Cluster nodes. The nodes are named **xcne1** to **xcne7**, e.g. `xyz@sunfire ~$ ssh xcne5` will connects you to the node **xcne5.** Your files and folders are shared across all compute cluster nodes (i.e. you can login another node to continue your work). Note that the files and folders **are**

	not shared / linked to your sunfire account. *** Details of the nodes (e.g. hardware spec, OS spec etc) can be found on** https://dochub.comp.nus.edu.sg/cf/guides/compute-cluster/hardware **** As mentioned, we will evaluate your lab assignment using the compute cluster. So, it is your responsibility to check that your work compiles and run well on the nodes. ****
2	**Install Ubuntu 20.04 natively on your PC** A great way to experience Linux on your own machine. You can gain useful experience "playing" with Ubuntu to learn the common operations on Linux.
3	**Use VM on your PC to run Ubuntu 20.04** Instead of going for a full native installation. You can use VM tool, e.g. VirtualBox on your PC to run Ubuntu. We have provided a virtual machine image for Lubuntu (Ubuntu 20.04 with LXQt) on Luminus.

1.2 The Command Line Interface (CLI)

Whenever your login to the Compute Cluster Node (or start a terminal on Ubuntu). The terminal you started is actually composed of two programs. One is a graphical application which gives a window and a terminal emulation. The other is a **shell** which interprets ASCII command lines which is connected to the terminal. The **shell** uses a command line interface, because it is text based. You type in commands to a shell which performs the command(s) in that line of text. In Windows, there is also a command line shell, **cmd** which has a similar purpose.

There are a number of different shells available under Linux. **BASH** (**B**ourne **A**gain **SH**ell) is the default and is a GNU enhancement descended from the original Unix shell, the **Bourne shell** (**sh**). You can write ***shell scripts*** which are small programs in the shell language to run a series of shell commands.

1.3 Using the built-in manual page (man)

Most Unix commands and C built-in libraries have built-in documentation. Use the *Unix Manual* command: **man** to access these documentation:

```
man XXXX
```

where **XXXX** is either the command/C built-in library call/etc that you want to know more about. For example, the **man** command itself is self-documenting:

```
$ man man
```

The above means “get the manual page for the command **man**”

Try out:

```
$ man fprintf
```

Which explain the C-Library function **fprintf()**. Take note that Unix man pages are written in a terse and very technical fashion. So it works more like a reference rather than a user-friendly tutorial. For a taste of a very long and scary looking man page, try "**man gcc**"!

The manual pages are organized using sections, and an entry might be in more than one section, e.g. **man 1 login** documents the command line login program while **man 3 login** documents the files which record the users of the system. You can search for keywords with the **-k** option, eg. **man -k login**.

The various manual sections have their own introduction. Try **man intro** or **man 1 intro**. System calls are described in **man 2 intro**. The **man** uses a **pager**, which is another program to display one page at a time on a terminal. You can page forward with **[SPACE]**, backward with '**b**' and quit the manual page at any time with ‘**q**’.

1.4 Listing Files and Directories

The **`ls`** command will list all the files in the specified directories. You can use the **`-l`** option for a long listing format which displays information like size, permissions, owner, group, creation date, etc. If you do not specify a directory, by default it uses the current directory. The **`-R`** option causes **`ls`** to list recursively.

Some examples to try:

```
$ ls
$ ls -l
$ ls /
$ ls -l /
```

1.5 Editing Files

You can use various text editors either directly from the desktop or the terminal. Some editors are strictly text based and some have a graphical interface in X. Many editors are available such as: **emacs, vi, vim, gedit**, etc. Editors like **emacs** and **vi** / **vim** are not so easy to use but are fairly powerful and popular with programmers. You are encouraged to learn **vi**/**vim** or **emacs,** many tutorials are available on the web. (Note: **vim** and **emacs** are available on Windows too).

If you are new to **Unix**, you can use **gedit** / **nano** which is more user friendly.

1.6 Backup and Transferring Files

If you need to transfer files and folders to / from the compute cluster node, there are a few common choices:

- scp: Secure Copy. A command you can type to move files between Compute Cluster Node and Sunfire.
- git: Popular version control software. ** Please ensure your git repository for CS2106 labs is **private.** **

1.7 Some Things to Remember!

The following are notable points for using Unix in general and also applicable to Linux:

- Unix is *case sensitive*. Most commands are lowercase.
- Unlike Windows, Unix **has no drive letters** (i.e. no C:, D:, E: etc). Everything is in some directory (i.e. folder).
- Unix uses forward slash (**/**) to separate directory names, while Windows uses backslash (****).
- The ***** wildcard is treated uniformly by Unix shells. In Windows, ***** works differently for different programs.
- It is worthwhile to look first at the **man** page of a command
- Try using various command line programs. Some programs need administrator privileges (in Unix, this is the root user who has *superuser privileges*) to run.

Section 2 Sample Usage Session

Skip this section if you already know how to use Unix / Linux.

1. **`mkdir junk`** // makes a new directory (i.e. folder) ***junk***
2. **`cd junk`** //change directory to it, you are now in ***junk***
3. **`pwd`** //prints the path of your current directory
4. **`echo abcd > test1.txt`** // makes a new file **test1.txt** with contents "**abcd**"
5. **`less test1.txt`** //display **test1.txt** using the pager. `**q**' will quit from less and `**h**' will give the help screen.

Select an editor to use (see previous editing section).

1. edit the file **test1.txt**, eg.

   `nano test1.txt` //or any editor of your choice
2. change the content of the file "**abcd**" to something else
3. save the file in the editor (this is editor specific)
4. **`cat test1.txt`** //concatenate 1 or more files, and print to output, so this displays **test1.txt** because output is the terminal
5. **`cat test1.txt test1.txt > test2.txt`** //**test1.txt** is concatenated twice and the output saved to **test2.txt**
6. **`cp test1.txt test3.txt`** //copies **test1.txt** to **test3.txt**
7. **`ls`** //listing should show 3 fiels
8. **`rm test1.txt`** //deletes **test1.txt**
9. **`ls`** //only **test2.txt** and **test3.txt left**
10. **`rmdir ../junk`** //try to remove directory, complains about non-empty directory
11. **`rm test*.txt; cd .. ; rmdir junk`** //no more files in directory, go up to parent directory, remove junk directory. You can perform multiple commands in a single line by using the ";" separator.
12. **`logout`** // or you can just press <Ctrl-D>

Section 3 Lab Assignment Format and Organization

3.1 Lab Assignment Duration & Lab Demonstration

Each lab assignment spans **two weeks** and consists of multiple exercises. One of the exercises is chosen to be the "lab demo exercise" which you need to demonstrate to your lab TA. For example, in Lab 1, you will need to demo exercise 1. The demonstration serves as a way to "kick start" your effort as well as a way to mark your **lab attendance**. You are **strongly encouraged to** finish the demo exercise before coming to the lab.

Your lab TA will arrange a schedule with the group for the lab exercise demonstration. Once you have demonstrated for that particular assignment, it is not compulsory to attend both weeks of lab.

The remaining lab exercises are usually quite intensive, please do not expect to finish the exercise during the allocated lab session. **The main purpose of the lab session is to clarify doubts with the lab TAs.**

3.2 Encouraging Exploration Spirit vs "No marks left behind Syndrome"

We observed that there is a strong "mark anxiety" among the students, i.e. you are too anxious to get the "perfect" submission to get full marks in CA. This creates unnecessary anxiety and hinders the exploration and learning process.

We will implement the following measures:

a. All lab **submission** can be done in a group of up to 2 students: Find a buddy, learn together! Discuss and make sure you understand the ideas deeply. The details of group forming will be included in each lab assignment write-up.

b. All CA components (Tutorial and Lab) have **overflow buffer**, e.g. the Lab component has a tentative 30% worth of work, but the cap is at 25%. Any extra marks beyond the cap can flow over to other component (e.g. Lab → Tutorial or vice versa). This buffer also mean that **you can afford to make mistake**, you don't need perfect score for every single lab ☺.

c. Last but not least, we will utilize the learning from these CA work in the midterm / final assessment, i.e. instead of just "work to be done", you should take it as "lesson to be learned".

3.3 Setting up the Lab exercises

For every lab, we will release two files in Luminus Files folder "Labs":

- **lab*X*.pdf**: A document to describe the lab question, including the specification and the expected output for all the exercises.
- **lab*X*.tar.gz**: An archive for setting up the directories and skeleton files given for the lab.

For unpacking the archive:

1. Copy the archive **lab*X*.tar.gz** into your account.
2. Enter the following command:
   gunzip –c lab*X*.tar.gz | tar xvf –
   Remember to replace the ***X*** with the actual lab number.
3. The above command should setup the files in the following structure:

L*X*/		topmost directory for lab X
ex1/		subdirectory for exercise 1
	ex1.c	skeleton file for exercise 1
	testY.in	sample test inputs, Y= 1, 2, 3, ...
	testZ.out	sample outputs, Z = 1, 2, 3, …
ex2/		
	...	Similar to ex1
ex3/		Similar to ex1
...		

3.4 Testing your labs

For most of the exercises, a number of sample input/output are given. The sample input/output are usually simple text file, which you can view them using any editor or pager program.

You can opt to manually type in the sample input and check the output with the standard answer. However, a more efficient and less tedious way is to make use of **redirection** in Unix.

Let us assume that you have produced the executable **answer.exe** for a particular exercise. You can make use of the sample test case with the input redirection:

```
$ ./answer.exe < test1.in
```

The above "tricks" the executable **answer.exe** to read from **test1.in as if it was the keyboard input**. The output is shown on the screen, which you can manually check with **test1.out**.

Similarly, make use of output redirection to store the output in a file to facilitate comparison.

```
$ answer.exe < test1.in > myOut1.txt
```

The effect of the above command is:

- Take **`test1.in`** as if it is the standard input device
- Store all output to **`myOut1.txt`** as if it is the standard output device
  - Obviously, any other filename can be used
  - Just be careful not to overwrite an existing file

With the output store in **`myOut1.txt`**, you can utilize the **`diff`** command in unix to compare two files easily:

```
$ diff test1.out myOut1.txt
```

which compares the output produced by your program with the sample output. Do a "**`man diff`**" to understand more about the result you see on screen.

You are **strong encouraged** to check the output in this fashion. By making sure your answer follow the standard answer (especially the format), you free up more time for the Lab TA to give better comments and feedback on your program.

Note that we will use **additional test cases** during marking to further stress test your submission. It is also part of your training to come up with "killer tests" for your own code. ☺

3.5 Lab Submission and Late Submission policy

Each lab has a submission folder under the main "Lab Submission" folder in Luminus. The submission deadline will be clearly spelt out in the lab sheets.

**** We impose late penalty as a cap of that lab score at 0.8^N, where N is the number of days late. The cap is rounded down to the nearest 10%. N is bounded by 5 (i.e. we do not accept late submission more than 5 days late).**

Late by 1 day (<= 24 hours)	Score capped at 80%.
Late by 2 days (> 24 hours and <= 48 hours)	Score capped at 80^2% = ~64% round down to 60%.
Late by 3 days	Score capped at 80^3% = ~51% round down to 50%.
….	…….
Late by more than 5 days	**NOT ACCEPTED**

~~~ End of Document ~~~