CPSC 3600 Homework1 Name/s: **Ashlyn Cooper**
Spring 2021 (50 points) **Kristopher Luo**

Due: 2/16/2021 before 9pm EST.

Instructions:
Can be done individually or in teams of two!!! If you are doing it in teams of two, please make sure that both of you know how to do all the exercises, and make only one submission per team. A great way to collaborate is to use a Zoom session and share screen while you work! You *cannot* talk to other students or teams, you can only talk to your teammate, teacher, or TA. Teammates share the grade. All the skills in this homework are very useful! When submitting your homework, clearly indicate the names on top of the file (no name, no grade). Your screenshots should be trimmed to remove the unneeded areas, so the images are clearly readable. ALL explanations should be in your own words, not copy-pasted from Internet. This file is in the editable document format, so you can make space between questions and type your answers right there. Then save it as a pdf and submit to canvas. Please do not use red font!

Question 1 (15 points)
Read the paper "The Unix Time Sharing System" in Files section of canvas. Based on the paper, and possibly with a bit more digging into additional material on Unix/Linux, answer the following questions.

1. Define a process. (1 point)

A process is the execution of an image, which is a computer execution environment.

2. Identify three types of files in Linux. Briefly describe each type in your own words. Find an example of each and paste it here along with its path. (3 points)

- Ordinary Files are normal files that contain whatever information the user puts on it. They are not expected to be any kind of structure and the structures are controlled by programs that use them and not the system itself. They can contain data, text, or program instructions.
    - Example: a normal text file on the user's desktop called *filename.txt*
    - Path: */home/cpsc3600/Desktop/filename.txt*
- Directories are files that are used to track and locate other files and directories. They provide a structure to the file system as a whole and map between the names of the files and the actual files themselves. They can store both ordinary and special files. They contain entries for every file and subdirectory that they house and are made up of 2 components which are the filename and the i-node number.
    - Example: One directory is the *Downloads* directory for the user

    - Path: */home/cpsc3600/Downloads*
- Special files are files that represent physical devices and are used for Input/Output (I/O) operations. Special files help keep file and device I/O as similar as possible.
    - Example: *stdin* is a special character file that deals with standard input
    - Path: */dev/stdin*

3. Summarize the access control system provided by Unix. (4 points)

Unix uses access control lists or ACLs. Basically, every file in a Unix system has an owner or group and a set of permissions. The ACLs are used to simplify who has what permissions to each file. So, processes on Unix each are associated with or is a member of some groups and those groups control who has access to those files or processes and what permissions that have.

4. What is an i-node? (1 point)

The i-node is a unique identification number that describes a file or directory. This number is created when a file or file system is created. The i-node contain several pieces of information including the owner, protection information, the size, the address for the file contents, timestamp information, file type, and number of links to the file.

5. What is a shell? (1 point)

A shell is a command line interpreter that reads what is typed by the user and interprets them as execution requests to other programs.

6. What is I/O redirection? Explain what standard input, standard output and standard error streams are. Please show an example or two of each. (5 points)

I/O redirection is when the input and output that may be normally intended for standard input or standard output is redirected or diverted to another file or device.

- Standard input is the source of input data for command line programs and is often abbreviated stdin. It is by default any text that is entered from the keyboard.
    - Example: take input from a file and display to the screen

```
cat < file.txt
```

    - Example: Redirecting input from a file into a program (instead of program reading from command line it reads from a file)

```
./myprogram < inputFile.txt
```

- Standard output is in reference to the streams of data that are produced by command line programs. Standard output's default destination is the display screen on the computer that initiated the program.
    - Example: Redirecting a program's output to a file and not the screen:

```
./a.out > output.txt
```

    - Example: Redirecting to a device:

```
cat music.mp3 > /dev/audio
```

- Standard error is the destination of error messages from command line programs. It also goes to the display screen by default. It is different from standard output because the two data streams can be redirected separately.
    - Example: Redirect error messages to a file using 2 (the file descriptor for standard error)

```
myProgram 2>errorfile
```

Question 2 (9 points)

1. Explain the results you see when you run the two pipelines below. A pipeline is a sequence of commands/filters/programs each having its standard output redirected to the standard input of the next command in the sequence. (3 points)

To understand what is going on in the command line, please first research and describe what *wc, head,* and *od* commands are and what their options (*-c -x -ending=big*) mean. (3 points)

- wc gives you either the line (-l), word (-w), or character (-m) count of the file. The option -c gives you the byte count.
- head allows you to display the first part of a specified file (the default is displaying 10 lines). The options allow you to choose how many lines to display.
- od command is used to convert an input into different formats. The default format is octal. It is useful in debugging scripts. -c gives a format specification, specifically to select ASCII characters. -x selects hexadecimal 2-byte units.

a. head -c 5 /dev/urandom | wc. -> prints first 5 bytes

When I run the above pipeline, I get a sequence of three numbers, a 0 followed by a 1 followed by a 5 on the same line. When the pipeline is run, the *head -c 5 /dev/urandom* selects the first 5 bytes of the *urandom* file as input. Then *wc* outputs certain counts to the command line. The first number is the number of lines from the input, the second is the number of words, and then third is the number of bytes, which is 0, 1, and 5 in this case.

b. head -c 5 /dev/urandom | od -x –endian=big

This pipeline takes the same input as the first pipeline, but outputs different information. It displays the first 5 bytes outputted as hexadecimal two-byte units. The string of zeroes at the beginning and the number on the second line of output indicated the byte offset, in this case 5. The command *–endian=big* puts the bytes in big-endian order. So each byte has the most significant digit stored first.

```
cpsc3600@vm1-ubuntu-1804:/$ head -c 5 /dev/urandom |od -x --endian=big
0000000 4718 73c2 3700
0000005
```

2. What is the purpose of the /dev directory in Unix/Linux? What is */dev/urandom* and what is its purpose? (3 points)

The /dev directory contains the special device files for all the devices and it points to those devices.

/dev/urandom is a character special file that provides an interface to the kernel's random number generator. /dev/urandom is used for most practical purposes over /dev/random. This is because it can be used where there is a constant need of random numbers and there is less of a security concern because the output of random numbers will never be blocked.

Question 3 (9 points)

Using your VM or a standalone Linux installation, please answer the following questions.

1. Issue the 'man man' command on the command line. What information did you find? How many sections does the manual have? Briefly summarize all the findings. (5 points)

The 'man man' command gives the user information about the manual command 'man'. The man command brings up the accompanying manuals for many of the protocols, functions, and commands in Unix. So, by issuing 'man man' the user is given the manual for the 'man' command itself. The manual has nine sections. These sections contain information on executable programs or shell command, system calls, library calls, special files, file formats and conventions, games, miscellaneous, system administration commands, and kernel routines. After issuing 'man man' the user can also see generic information for what a manual for different commands may look like, such as section names and conventions for information in the manuals. Issuing 'man man' also gives detailed information on the man command including a description, history, its environments, defaults, and options.

2. What is the difference between a 'man 1 printf' and 'man 3 printf' ? (1 point)

The difference between the 2 commands is that they each access a different section of the printf manual. ‘man 1 printf’ pulls up the section of the manual dealing with printf as a shell command or executable, so it displays information about printf in that context. ‘man 3 printf’ pulls up information from the third section of the printf manual, which is library calls. So ‘man 3 printf’ pulls up C library function specific information for printf.

3. What does command *apropos* do? Give two example of how to use it. What is the difference between *man* and *apropos*? (3 point)

The *aporopos* command helps a user figure out a command related to something they are trying to do. So, if a user can’t remember a specific command to do something, but they remember some keywords, apropos searches the Linux man page to find possible commands the user could use. So, for example issuing ‘apropos compress’ would show a bunch of commands in the terminal about how to compress a file, expand a compressed file, etc. Another example is ‘apropos email’ which will display all the commands that contain the ‘email’ in its man page. The difference between *man* and *apropos* is that *apropos* gives a list of all commands related to a specific keyword and the user doesn’t know the command while *man* displays all specific information about a specific command that the user knows.

Question 4 (12 points)

1. Find out what *traceroute* command does and explain it in your own words. When will this command be useful? (2 points)

The command *traceroute* is used to print out the route that a packet takes to reach the host. It tells the user the identities of the routers and devices on the route taken by the packet. It also gives time information about how long it took to send and receive data to each device on the path. The command *traceroute* can be useful when you need to determine if there are any response delays or routing loops in a network path. Also *traceroute* can help to locate any points of failure along a route.

On the *command line*, please issue a command
traceroute www.clemson.edu

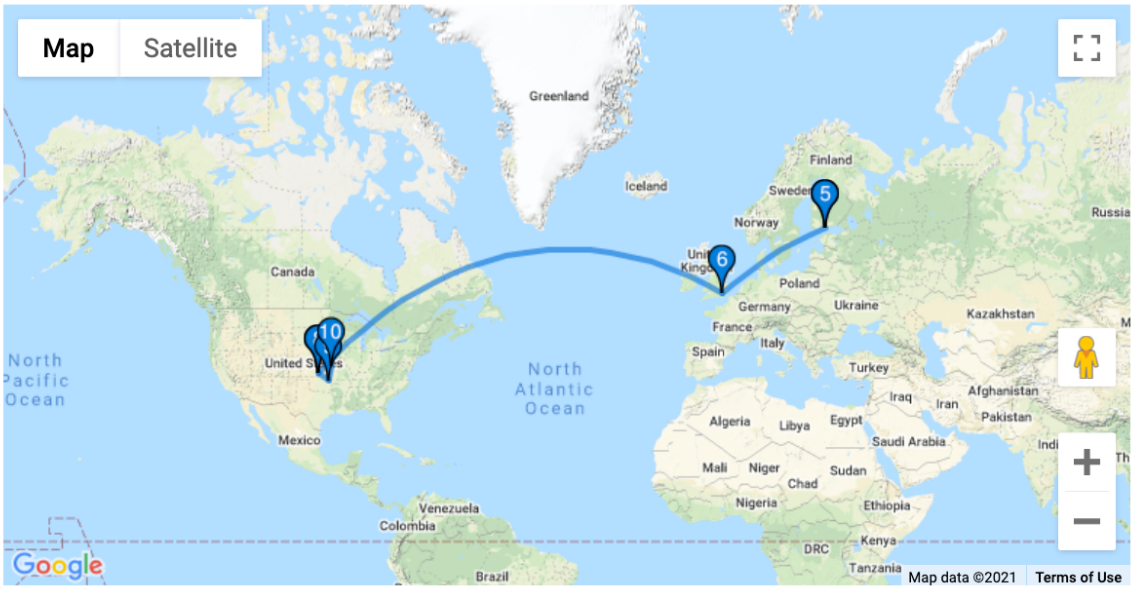
Paste a screenshot here. (Plz edit it to trim the areas that are not needed, so that the output is readable) Explain every entry of the output. Did you see any asterisks? What do they mean? There are three sets of time on each line. What are those times? (4 points)

```
cpsc3600@vm1-ubuntu-1804:~$ traceroute www.clemson.edu
traceroute to www.clemson.edu (130.127.204.30), 30 hops max, 60 byte packets
 1  _gateway (10.0.2.2)  0.308 ms  0.247 ms  0.218 ms
 2  130.127.144.3 (130.127.144.3)  3.981 ms  3.954 ms  6.578 ms
 3  130.127.3.43 (130.127.3.43)  6.542 ms  6.648 ms  6.619 ms
 4  130.127.3.74 (130.127.3.74)  6.408 ms 130.127.3.96 (130.127.3.96)  6.379 ms
130.127.3.70 (130.127.3.70)  6.482 ms
 5  130.127.204.30 (130.127.204.30)  6.250 ms  6.217 ms  6.188 ms
cpsc3600@vm1-ubuntu-1804:~$
```

In the output after running the *traceroute* command there are several pieces of information. The first column/entry of each line of output represents the hop count. In the case above, the packets took 5 hops along the path. The second entry in the line represents the address where the hop was made. In the first line this is given the special designation of gateway, which is the source address. Following the addresses of each hop, each line has three separate times in milliseconds. Not only does *traceroute* show the location and path that packets travel, it also shows timing. The *traceroute* command sends out three packets to the hop and each time refers to the time taken by each packet to reach the hop. So because there are three packets sent, there are three times. In our case there are no asterisks in the output, but if there were that would be because *traceroute* has a "time to live" feature. This time-to-live is the period of time that a packet can experience before being discarded. So if an asterisks appears the time-to-live was reached and the packet was discarded.

2. Using your favorite search engine, find a visual *traceroute* tool online. (no need to download/install anything, you can use it on the website). Use a known website, other than amazon.com or google.com to run the trace. What information did you find? Does this provide more information than the command-line traceroute? Paste a trimmed screenshot here. (5 points)

For our example we chose to trace www.spotify.com. The visual traceroute gives a lot of the same information as the command line *traceroute.* Both methods show the hop count, the IP address of the hop and the time it took for the packet to reach the hop. The visual traceroute had more information describing hop locations like some of the names associated with the addresses and it obviously shows visually where around the world the packets traveled to. The command-line traceroute shows a little more timing data, as it displays times for three packets while the visual traceroute only shows one time.



traceroute to www.spotify.com (35.186.224.25), 30 hops max, 60 byte packets

Hop	Host	IP	Time (ms)
1	_gateway	94.237.52.1	0.170
2	100.69.38.177	100.69.38.177	0.339
3	172.17.255.213	172.17.255.213	0.347
4	172.17.255.249	172.17.255.249	0.335
5	r1-lon1-po1.uk.net.upcloud.com	94.237.0.120	0.333
6	5-1-33.ear2.London1.Level3.net	212.187.165.105	0.405
7	72.14.212.120	72.14.212.120	0.752
8	108.170.246.129	108.170.246.129	0.651
9	216.239.56.193	216.239.56.193	0.612
10	25.224.186.35.bc.googleusercontent.com	35.186.224.25	0.584

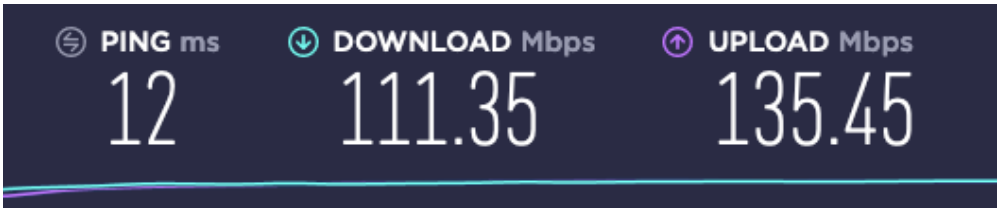
Question 5 (6 points)

1. Search for an online speed test to determine your network bandwidth. Measure your asymmetrical bandwidth upsteam and downstream. Which one is higher and why? Run the test 4 times and calculate the average upload and download rates. Paste one of the screenshots with test results here. (3 points)

In out testing the upstream rates were actually higher than the downstream rates. This is odd because in most scenarios the download rates should be faster because there is typically more bandwidth available for download because people generally download than they upload. In this case the upload speeds could be faster because there is more congestion on the download side with more people streaming and downloading content on the weekend.

Test Using Ookla:

Average download speed: 108.585 Mbps
Average upload speed: 136.7375



2. Now find another online speed test. Repeat the same. Get the average of the 4 tests again. Are the results consistent between the two different speed tests, or are there variations? Explain why it would be so. (3 points)

Using Google Speed Test:
Average download speed: 112.425 Mbps
Average upload speed: 134.725 Mbps

The overall results of both of the speed tests are similar in that they show the upload rates being high than the download rates and that the download rates are around 110 Mbps and the upload rates are around 135 Mbps. The numbers did slightly vary for each test. This could be due to changes in network conditions between tests, differences in the locations of the testing servers, and differences in testing methods for each service.

Extra Credit (3 points)
Using your VM or Linux installation machine, find out how to obtain information about your computer's network interface using command line utilities. Which file/s in Linux contain your machine's IP address and other network info. Show a screenshot of the command you ran that shown this info and a screenshot of a file you found.

The IP address and other network info of the machine is found in the *ifconfig* file which is located in the */sbin* directory. This information can be accessed by either typing in the command *ifconfig* or by navigating to */sbin/ifconfig.*

The two options are:

```
cpsc3600@vm1-ubuntu-1804:/$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::3874:491c:5d4d:5b76  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:bf:77:1c  txqueuelen 1000  (Ethernet)
        RX packets 1246  bytes 1451203 (1.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 627  bytes 63732 (63.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 158  bytes 14890 (14.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 158  bytes 14890 (14.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
cpsc3600@vm1-ubuntu-1804:/$ /sbin/ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::3874:491c:5d4d:5b76  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:bf:77:1c  txqueuelen 1000  (Ethernet)
        RX packets 1246  bytes 1451203 (1.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 627  bytes 63732 (63.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 158  bytes 14890 (14.8 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 158  bytes 14890 (14.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```