CPSC 3600 HW #1 Anthony Jones

Part a: Define a process: **A process is the execution of the image, aka the computer execution environment. It is an active running of a computer program, in which the core image is altered through patterns related to the process’ resources.**

Part b: Identify three types of files in Linux. Add a short sentence to define the type.

**An ordinary disk file is a type which represents a collection of information stored on the computer.**

**Directories are types that add structure by linking to other files via address information.**

**Special files are UNIX I/O device files that activate the device and correlate them to the file system.**

Part c. Summarize the access control system provided by Unix. **Basically in UNIX there are two protection schemes: the first is a unique user identification number that tracks which user created a file. That user is the owner of the file. In addition, the file has a 7 bit protection block that specifies user permissions for the owner and other users. The 7th bit acts as a switch which may temporarily allow certain calls to function like an owner call; this allows a “super-user” effect to be initialized by programs wishing to alter or create files on the system.**

Part d. What is an i-node ? **When a directory entry is accessed, the pointer of that entry (the i-number) will search a system table (an i-list) to find information about the file being accessed. This is the i-node, which ultimately helps to abstract information and allow short, unambiguous naming conventions that relate to the protection, addressing, and other information needed to access a file.**

Part e. What is a shell? **A shell is a program that acts as an interface and interpreter for command execution via user input. It waits for a user to input a command (and arguments / filters) and then it reads, executes, and forks a process as talked about in class. This allows a process to inherit the core image of all files associated with its parent, meaning filters act like extended functionalities of pipelines where the shell simply automates the redirection actions when necessary.**

Part f. What is I/O redirection? Please show several examples in your answer. **Programs executed by the shell start off with two open files that have the file descriptor 0 and 1; 1 is the open writing file which is understood as standard output (O), while the 0 descriptor is the open reading file that is standard input (I). The shell can change these assignments to be different files using redirection commands and pipes:**

**ls > output : redirects the standard output to the file output**

**wc -l < input : redirects the standard input to the file input**

**touch test | wc -l : redirects touch output (test) to wc input, prints 0**

Part g: What is the difference between a pipe and a fifo in Unix ?

**A fifo is a named pipe that has a few important differences in UNIX. The first is that a fifo is a part of the file system, and therefore it has a unique address and file located somewhere on the system. It can be read bidirectionally, meaning that either end can be called by unrelated processes. Also, and perhaps most fundamentally to this class, fifo files are not restricted to local use. This means that computers across a network system can use fifo files to redirect input and output.**

Question 2 (15) : Explain the results you see when you run the two pipelines.

A pipeline is a sequence of commands/filters/programs each having its standard output

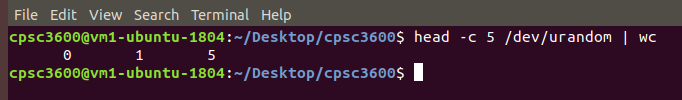
redirected to the standard input of the next command in the sequence.

1. head -c 5 /dev/urandom | wc

**“head -c 5 /dev/urandom” will print the first 5 bits of the urandom file to standard output.**

**“wc” will print newline, word, and byte counts for standard input.**

**Therefore, “head -c 5 /dev/urandom | wc” will print the newline, word, and byte counts of the piped input of the first 5 bits of the urandom file. This should be 0 newline, 1 word, and 5 bytes.**

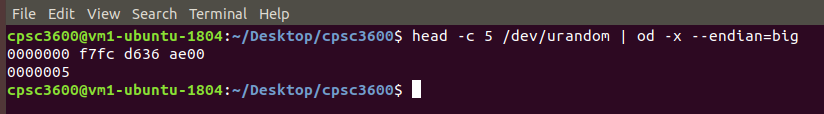


1. head -c 5 /dev/urandom | od -x --endian=big

**“head -c 5 /dev/urandom” will print the first 5 bits of the urandom file to standard output.**

**“od -x –endian=big” will print an octal dump of standard input in hexadecimal 2-byte format.**

**Therefore, “head -c 5 /dev/urandom | od -x endian=big” will print the octal dump of the first 5 bits of the urandom file in hexadecimal 2-byte format. Big endian means the most significant byte pair will come first.**



Question 4 (25 points) - Using your VM, please answer the following questions.

Part a: To find all man pages, issue ‘man –k . ‘ Pipe the output to the wc filter to learn how many commands/programs have man pages. **There are 6387 commands that have man pages**

• What is the difference when we issue a ‘man 1 printf’ and ‘man 3 printf’ ? **The first section (man 1) details executable programs and shell commands, while the third section (man 3) details library calls aka functions within program libraries.**

• What is the difference between ‘man ping’, ‘man -k . | grep ping’ and ‘apropos ping’ ? ‘**man ping’ will bring up the default man page for ping, detailing information about the command. ‘man -k . | grep ping’ will first apropos every man command into standard out, and then pipe that into standard in where grep finds all patterns involving the word “ping” and outputs it to standard out. ‘apropos ping’ simply prints out all manual pages that contain the word “ping”.**

Part b: Issue ‘ping –s 1000 –c 5 8.8.8.8 &> ping.out

• Copy and paste the final summary stats. Provide a sentence for each value that is displayed (e.g., the mean is …. )

**PING 8.8.8.8 (8.8.8.8) 1000(1028) bytes of data.**

**76 bytes from 8.8.8.8: icmp\_seq=1 ttl=119 (truncated)**

**76 bytes from 8.8.8.8: icmp\_seq=2 ttl=119 (truncated)**

**76 bytes from 8.8.8.8: icmp\_seq=3 ttl=119 (truncated)**

**76 bytes from 8.8.8.8: icmp\_seq=4 ttl=119 (truncated)**

**76 bytes from 8.8.8.8: icmp\_seq=5 ttl=119 (truncated)**

**--- 8.8.8.8 ping statistics ---**

**5 packets transmitted, 5 received, 0% packet loss, time 4035ms**

**rtt min/avg/max/mdev = 11.624/13.664/14.850/1.098 ms**

**~ rtt stands for round trip time, so the min/avg/max are all the minimum, average, and maximum times out of all the requests. The mdev is basically like standard deviation.**

• What does the –s parameter do? What does the –c parameter do? **‘-s’ specifies packet size (in this example packet size is 1000) while ‘-c’ specifies how many times to ping (the count is 5 in the example above).**

• What exactly is the &> doing **This is redirecting both standard err and standard out to the file “ping.out”. Because “ping.out” didn’t exist yet on my computer, it created the file in the local directory and then outputted.**?

Part c:

• Issue ‘traceroute 8.8.8.8 &> traceroute.out ‘

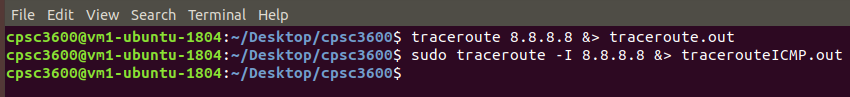
• Then issue ‘sudo traceroute –I 8.8.8.8 &> tracerouteICMP.out ‘

Copy and paste the results observed in the terminal and in each output file.

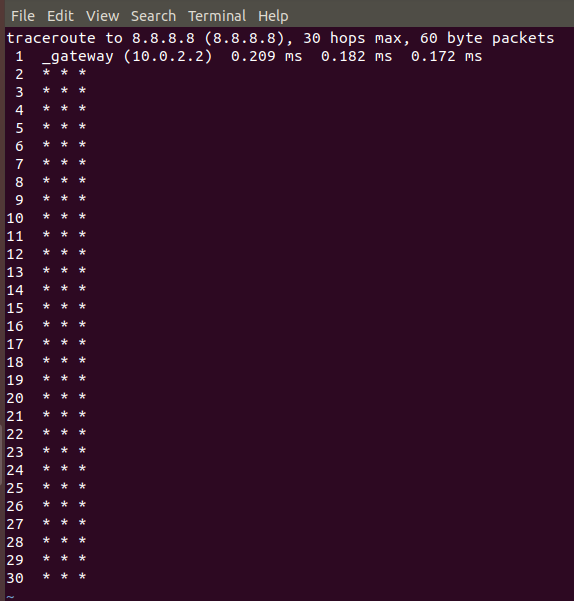
Aside from minor differences in the statistical RTTs that traceroute shows, explain

any significant differences between the two invocations of traceroute.

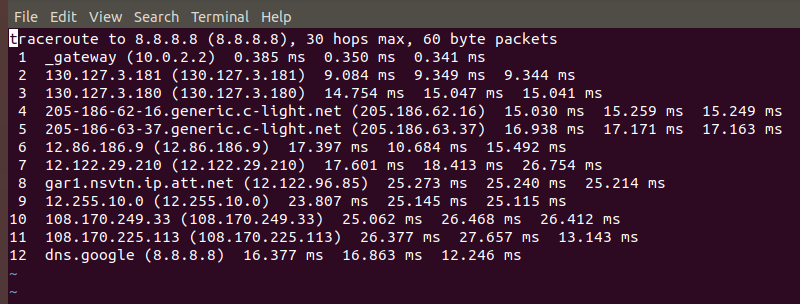
**Terminal**



**Traceroute.out**



**TracerouteICMP.out**



**Basically, the first invocation of traceroute uses UDP packets which are likely being blocked by the firewall. Therefore, the server requests are timing out and giving \* \* \*. Most firewalls block traffic that is not specific to the default IP port for web traffic.**

**In the second invocation, we use the tag -I to instead deliver ICMP packets, which are shown to be permitted in our firewall. This is why all of the requests are shown with their timestamps. Pretty neat!**

Q5 programming (25 points) Consider the CPPex1 example.

The code is located in our git repo at location code/CPPCode/CPPex1.

Issue a ‘make clean; make’

Corrected 2/7/2021…..

The original question did not match the latest CPPCode/CPPex1 code- our git repo

already had the modified loop.cpp. I’ll correct and simplify the question to the following:

Study the CPPex1 code. Make sure you understand how we have used a ‘soft link’ to

facilitate using common code. Also, make sure you understand the Makefile as well as in

loop.cpp and the helper Objects the program uses.

Look at the script ‘ExSol.sh’ – it calls runEXPs.sh twice. Run ‘go’ and look at the two data

files produced. In 4 sentences or less, explain the possible impacts of using the third

parameter (the mode parameter passed on in the call to loop). Examine the data. Do you

see the expected difference?

**Well the third parameter of the runEXPs.sh call is the loop mode, which basically uses two timing methods to calculate the response time. getCurrentTime() returns the systems wall clock time using a system call that is only accurate to a microsecond. To bypass this, we use the functionality of CLOCK\_REALTIME; but any user or program can change the value of the systems wall clock software clock. However, using getTimestampD(), we use the functionality of CLOCK\_MONOTONIC\_RAW, which will never be modified by any software. This is best to find the difference between two events and would be notable here if any system or network application changes our system clock in any way.**

**However, after examining the data, we cannot really notice the expected difference. The differences between the two outputs could simply be because of the variability of the requests themselves; furthermore, it is pretty unreasonable to expect any major system clock change to happen in such a short span of requests. But if we were constantly monitoring the delivery of large data over days and weeks, this difference would be more noticeable.**