CS304
Assignment 1

Vipul Yuvraj Nikam
180010041

Q1. more /proc/cpuinfo

(a) Run command more /proc/cpuinfo and explain the following terms:

a. Processor: It is an assemblage of cores. In my system there are 8 CPUs for which processors numbered from 0 - 7 are there. A chip can contain one or more processors.
b. Cores: A core is the smallest independent unit that implements a general purpose processor Command used - more proc/cpuinfo and lscpu. Multiple cores allow multiple processes to run at the same time multi-tasking.

(b) How many cores does your machine have?

Answer :

4

(c) How many processors does your machine have?

Answer :

8

(d) What is the frequency of each processor?

Answer :

800.070 MHz

(e) How much physical memory does your system have ?

Answer :

4 GB

(f) How much of this memory is free ?

Answer :

2GB

(g) What is the total number of forks since the boot in the system ?

Answer :

40709 forks

(h) How many context switches has the system performed since bootup ??

Answer :

310531085

Q2. $ gcc cpu.c -o cpu $ ./cpu

(a) What is the PID of the process running the cpu command?

Answer :

40835

(b) How much CPU and memory does this process consume?

Answer :

%CPU	%MEM
63.0	0.0
100	0.0

(c) What is the current state of the process? For example, is it running or in a blocked state or a zombie state?

Answer :

State: T (stopped)

Q3. $ gcc cpu-print.c -o cpu-print $ ./cpu-print

(a) This program runs in an infinite loop printing output to the screen.Now, open another terminal and use the ps command with suitable options to find out the pid of the process spawned by the shell to run the cpu-print executable. You may want to explore the ps command thoroughly to understand the various output fields it shows.

Answer :

41057

(b) Find the PID of the parent of the cpu-print process, i.e., the shell process. Next, find the PIDs of all the ancestors, going back at least 5 generations (or until you reach the init process).

Answer :

$ ps -C cpu-print -o ppid= 40647

```
$ pstree -sap 299639:          systemd,1 splash
                                 └─systemd,1868 --user
                                     └─gnome-terminal-,40640
                                     └─bash,40647
                                         ├─cpu,40835
                                         ├─cpu,40867
                                         └─cpu-print,41057
```

(c) To understand how the shell performs output redirection. Run the following com- Mand. ./cpu-print > /tmp/tmp.txt & Look at the proc file system information of the newly spawned process. Pay particu- lar attention to where it's file descriptors 0, 1, and 2 (standard input, output, and error) are pointing to. Using this information, can you describe how I/O redirection is being implemented by the shell?

Answer :

```
$ ./cpu-print > /tmp/tmp.txt &                    41321
$ ls -l /proc/381593/fd
total 0
lrwx------ 1 vipul vipul 64 Jan 24 23:15 0 -> /dev/pts/1
l-wx------ 1 vipul vipul 64 Jan 24 23:15 1 -> /tmp/tmp.txt
lrwx------ 1 vipul vipul 64 Jan 24 23:15 2 -> /dev/pts/1
```

Shell makes the input file descriptor of one process and the output file descriptor of another process for I/O redirection.

(d) To understand how the shell implements pipes. Run the following command. ./cpu-print | grep hello & Once again, identify the newly spawned processes, and find out where their standard input/output/error file descriptors are pointing to. Use this information to explain how pipes are implemented by the shell.

Answer :

```
$ ./cpu-print | grep hello &                      41518
```

‘|’ is used to pass the outputs of a program. Flow of data is allowed by input stream is kept the same as output of other programs. Difference in

the file descriptors between redirection and pipe is, very frequently Redirection is for files you redirect streams to/from files. Whereas Piping is there for edirect streams from one process to another.

(e) When you type in a command into the shell, the shell does one of two things. For some commands, executables that perform that functionality already come built into the Linux kernel.For such commands, the shell simply invokes the executable like it runs the executables of your own programs. For other commands where the executable does not exist, the shell implements the command itself within its code. Con-CS304 Assignment No 1 Page 2sider the following commands that you can type in the bash shell: cd,ls,history,ps. Which of these commands already exist as built-in executables in the Linux kernel that are then simply executed by the bash shell, and which are implemented by the bash code itself?

Answer :

$ type cd cd is a shell builtin

$ type history history is a shell builtin

Shell built-in are cd, history whereas Commands executed by bash code are ps, ls.

Q4. Consider the two programs memory1.c and memory2.c given to you. Compile and run them one after the other. Both programs allocate a large array in memory. One of them accesses the array and the other doesn't. Both programs pause before exiting to let you inspect their memory usage. You can inspect the memory used by a process with the ps command. In particular, the output will tell you what the total size of the "virtual" memory of the process is, and how much of this is actually physically resident in memory. You will learn later that the virtual memory of the process is the memory the process thinks it has, while the OS only allocates a subset of this memory physically in RAM. Compare the virtual and physical memory usage of both programs, and explain your observations.You can also inspect the code to understand your observations.

Answer :

$ gcc memory1.c -o memory1

$ ./memory1

Program : 'memory_1'

PID : 4402

Size of int : 4

Press Enter Key to exit.

ps -C memory1 -o pmem,size,rss,vsize:

%MEM	SIZE	RSS	VSZ
0.1	4096	4948	6272

$ gcc memory2.c -o memory2

$ ./memory2

Program : 'memory_2'

PID : 4470

Size of int : 4

Press Enter Key to exit.

$ ps -C memory2 -o pmem,size,rss,vsize

%MEM	SIZE	RSS	VSZ
0.1	4096	4948	6272

We can conclude from ‘size - 4096 & virtual mem. - 6272 (For memory1.c)’ that, it makes the process feel as if all memory is available for its use while the actual physical memory available to use is significantly lesser.

5. In this question, you will compile and run the programs disk.c and disk1.c given to you. These programs read a large number of files from disk, and

you must first create these files as follows. Create a folder disk-files and place the file foo.pdf in that folder. Then use the script make-copies.sh to make 5000 copies of the same file in that folder, with different filenames. The disk programs will read these files. Now, run the disk programs one after the other.For each program, measure the utilization of the disk while the program is running. Report and explain your observations. You will find a tool like iostat useful for measuring disk utilization. Also read through the code of the programs to help explain your observations. Note that for this exercise to work correctly, you must be reading from a directory on the local disk.If your disk-files directory is not on a local disk (but, say, mounted via NFS), then you must alter the location of the files in the code provided to you to enable reading from a local disk. Also, modern operating systems store recently read files in a cache in memory (called disk buffer cache)for faster access to the same files in the future. In order to ensure that you are making observations while actually reading from disk, you must clear your disk buffer cache between multiple runs of disk.c. If you do not clear the disk buffer cache between successive runs of disk.c, you will be reading the files not from disk but from memory. Look up online for commands on how to clear your disk buffer cache, and note that you will need superuser permissions to execute these commands.

Answer :

$ iostat -xtc disk

$ iostat -xtc disk1

We can conclude from codes of disk.c & disk1.c that I/O operations on a single file are performed by disk1.c, whereas every timed isk.c performs I/O on multiple files. This causes the difference in disk utilization, more for disk.c compared to disk1.c.

XXX