Assignment 1 - Processes

CS240
Cory Holt

3/12/2021

0.1 Program Designs

This assignment consists of 3 separate C programs. Each is a simple forever loop. The first program, "part1.c" simply loops and prints its own name and process ID once per second. It also keeps track of how many loops it has made. The second program, "arith_loop.c" only increments a counter each loop. It has no delay. The third program, "file_loop.c" opens a file, writes the character "a" into the file, and then closes the file. It loops forever.

0.2 Process Management

0.2.1 top Utility

The top utility is very useful for monitoring process status and resource usage. The normal usage shows several columns:

- PID: the process ID
- PR: Priority
- NI: Still Priority?
- VIRT: Virtual Memory of task
- SHR: Shared memory of process
- %CPU: Percent of CPU usage for each process
- %MEM: RAM usage of each process
- TIME+: CPU time used by each process
- COMMAND: name of the process

0.2.2 kill Utility

Once you get the PID from either 'top' or 'ps', you can suspend and resume the Process using the kill command. Using the kill command with the switch '-STOP' will suspend the task. Similarly, using kill with the switch '-CONT' will resume the specified process. The kill command can also intuitively kill the given process if you just use the PID as the only argument.

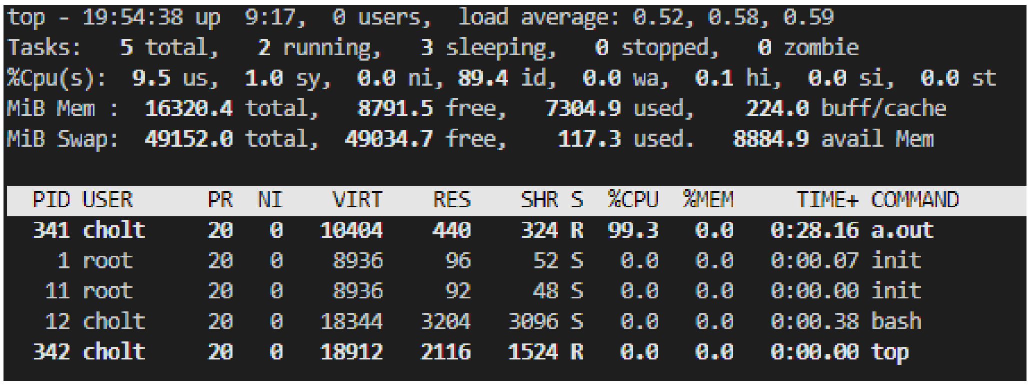
0.2.3 fg/bg

You can send a process to the foreground or background by using the fg or bg commands respectively.

0.3 Process Resource Usage

I wrote these programs using Windows Subsystem for Linux. I presume that the utilities will behave the same, but the CPU usage seemed a little strange for the case where both programs were running. The results were found using the 'top' command for each case.

0.3.1 Only Arithmetic

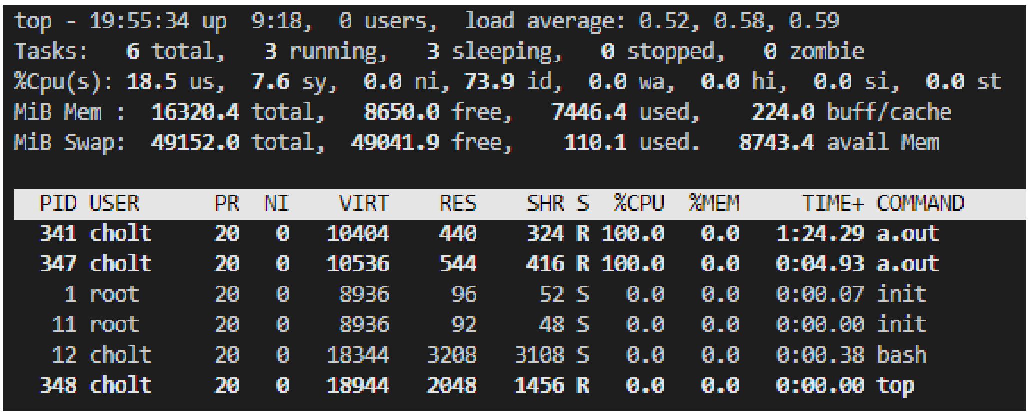


```
top - 19:54:38 up  9:17,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   5 total,   2 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  9.5 us,  1.0 sy,  0.0 ni, 89.4 id,  0.0 wa,  0.1 hi,  0.0 si,  0.0 st
MiB Mem :  16320.4 total,   8791.5 free,   7304.9 used,    224.0 buff/cache
MiB Swap:  49152.0 total,  49034.7 free,    117.3 used.   8884.9 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  341 cholt     20   0   10404    440    324 R  99.3   0.0   0:28.16 a.out
    1 root      20   0    8936     96     52 S   0.0   0.0   0:00.07 init
   11 root      20   0    8936     92     48 S   0.0   0.0   0:00.00 init
   12 cholt     20   0   18344   3204   3096 S   0.0   0.0   0:00.38 bash
  342 cholt     20   0   18912   2116   1524 R   0.0   0.0   0:00.00 top
```

Figure 1: top with only arithmetic

0.3.2 Arithmetic and IO



```
top - 19:55:34 up  9:18,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   6 total,   3 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s): 18.5 us,  7.6 sy,  0.0 ni, 73.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :  16320.4 total,   8650.0 free,   7446.4 used,    224.0 buff/cache
MiB Swap:  49152.0 total,  49041.9 free,    110.1 used.   8743.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  341 cholt     20   0   10404    440    324 R 100.0   0.0   1:24.29 a.out
  347 cholt     20   0   10536    544    416 R 100.0   0.0   0:04.93 a.out
    1 root      20   0    8936     96     52 S   0.0   0.0   0:00.07 init
   11 root      20   0    8936     92     48 S   0.0   0.0   0:00.00 init
   12 cholt     20   0   18344   3208   3108 S   0.0   0.0   0:00.38 bash
  348 cholt     20   0   18944   2048   1456 R   0.0   0.0   0:00.00 top
```

Figure 2: top with arithmetic and IO tasks

As you can see, the readings indicate that both tasks are quite intensive. They both consume 100% of the CPU and a considerable chunk of the CPU time. Neither consumes much real memory, but it appears each uses around 10000 Bytes of virtual memory, which seems high, but it could be a result of the virtual system.

0.4 Program size

Using objdump to decode the elf headers of each program will give us a look at the size of each program. In particular, we want to look at the .data and the .text sections. The .data section tells us the size of variables, and the .text section is the size of the executable instructions. The outputs of objdump are in their respective .txt files in each directory. For example, the executable text for the arithmetic loop is 0x175 bytes, while the file loop is 0x1c5. The data of the arithmetic loop is 16 bytes, and the data of the file loop is also 16 bytes.

0.5 Programming Log

Total Time Spent: $\approx$ 4 Hrs.

3/9/21: I wrote my functions today. They are all very short. It took so long since I had to research how to do some of the tasks such as sleep, print to stderr, and write a file. All seem to function well, and they run properly in the bg.

3/11/21: I spent today learning how to use the various utilities, and learning how to manage stopping, starting and killing processes. I also wrote the report and sent the relevant outputs to their respective log text files.