COMP4511: System and Kernel Programming in Linux Spring 2021

Programming Assignment 1

Release: February 8 Due: February 21, 23:59

Observing the OS through the `/proc` file system

The Linux kernel is a collection of data structures. The collective kernel variables define the kernel's perspective of the state of the entire computer system. Each externally invoked function could cause the system state to be changed by having the kernel code change its kernel variables. You can determine the performance and other behaviour of the OS by inspecting those states. In this assignment, we study some aspects of the organization and behaviour of the Linux system by observing values of kernel data structures exposed through the `/proc` virtual file system.

1. The `/proc` file system

The `/proc` filesystem is a virtual filesystem that permits a novel approach for communication between the Linux kernel and user space. In the `/proc` filesystem, virtual files can be read from or written to as a means of communicating with entities in the kernel. But unlike regular files, the content of these virtual files is dynamically created. You can read a wide range of information on kernel data structures from the `/proc` filesystem. If you `cd` into `/proc`, you will find several files and directories. The directories with numeric names correspond to processes in the system. For example, since the first process created in Linux is the `init` process with id of 1, the directory named "1" represents the `init`. It should be noted that files in `/proc` can be read just like oridinary files, like `fgets(), fscanf()`.

The `proc(5)` manual page explains the virtual files and their content available through the `/proc` filesystem.

```
$ man proc
```

To finish this assignment, you need to check this manual page: figuring out which file can provide the desired information.

2. Problem statement

You are required to write two C programs to report some behaviour of the Linux kernel.

You must read information from the `/proc` file system. You cannot run any shell command in your program.

Task 1 (40 points)

Write the `info` program, in the source file `info.c`, which displays system information every 3 seconds:

- System uptime, in the format: `Uptime: D day(s), H:MM:SS`. (10 points)

  For example,

  ```
  Uptime: 1 day(s), 7:45:07
  ```

  /proc/uptime

- Memory usage in kB (defined as total memory minus free memory), total memory in kB and the percentage of used memory, in the format: `Memory: used kB / total kB (%)`. The percentage should be displayed with 1 decimal place. (10 points)

  For example, /proc/meminfo

  ```
  Memory: 902176 kB / 4035964 kB (22.4%)
  ```

  https://www.binarytides.com/linux-command-check-memory-usage/

- CPU usage

https://stackoverflow.com/questions/41224738/how-to-calculate-system-memory-usage-from-proc-meminfo-like-htop https://www.kgoettler.com/post/proc-stat/

  - Displays the CPU usage of *each* CPU in the 3-second period spent in user mode and kernel mode. (5 points)
  - It should not assume the number of CPUs, meaning it should work on machines with different number of CPUs (5 points)
  - The usage should be displayed with 1 decimal place, left-padded with space to 5 characters. There should be 2 spaces between each CPU (5 points):

    ```
    CPU0: 100.0%  CPU1:   0.0%  CPU2:   0.0%  CPU3:   0.0%
    ```

- Accepts an optional integer argument to specify the period: `./info N` now shows infomation every `N` seconds. You can assume `N` is an integer. Report error if `N` is less than or equal to zero. (5 points)

Sample output:

```
$ ./info
(3 seconds later)
Uptime: 1 day(s), 7:59:44
Memory: 902188 kB / 4035964 kB (22.4%)
CPU0: 100.0%  CPU1:   0.0%  CPU2:   0.0%  CPU3:   0.0%

(3 seconds later)
Uptime: 1 day(s), 7:59:47
Memory: 902188 kB / 4035964 kB (22.4%)
CPU0: 100.0%  CPU1:   0.0%  CPU2:   0.0%  CPU3:   0.0%
```

Task 2 (60 points)

Write the `myps` program, in the source file `myps.c`, which displays information for each processes, similar to the utilities `ps` and `top`.

- For each process, display the following information in a row. The rows should be sorted in ascending process ID: (10 points)
  - Process ID (PID)
  - User name of the user who owns this process
  - Total CPU time spent in user and kernel mode
  - Size of virtual memory
  - Command
- The output should be formatted:
  - The PID is left-padded to 7 characters (2 points)
  - The User name is right-padded to 20 characters. (2 points)
  - CPU time is displayed in the format, `HH:MM:SS`, assumed CPU time never exceeds 100 hours. (2 points)
  - Size of virtual memory is displayed in kB, left-padded to 10 characters. (2 points)
  - Shows the header header in the first row: (2 points)

    ```
      PID USER           TIME    VIRT CMD
    ```

  - There is a space between each column

- Accepts the `-m` option to sort the processes in descending size of virtual memory. (10 points)

- Accepts the `-p` option to sort the processes in descending CPU time. Assume `-m` and `-p` are not used at the same time. (10 points)

- Accepts the `-u user` option to show processes whose *effective user* is `user`. (10 points)

  - `-u` should work with `-m` and `-p`. (5 points)
  - Report error if `user` does not exist. (5 points)
    For example,

    ```
    $ ./myps -u no_such_user
    invalid user: no_such_user
    ```

Sample output:

```
$ ./myps -u student -m
    PID USER                     TIME       VIRT CMD
  16901 student              00:00:00     169840 (sd-pam)
  16898 student              00:00:00      18572 systemd
  18287 student              00:00:00      13908 sshd
  16999 student              00:00:00      13904 sshd
  17270 student              00:00:00      13904 sshd
  17083 student              00:00:00      13900 sshd
  17084 student              00:00:00      10264 bash
  17271 student              00:00:00      10240 bash
  18288 student              00:00:00      10132 bash
  17000 student              00:00:00       5888 sftp-server
  18307 student              00:00:00       3100 myps
```

3. Hints

- You can write a simple program containing an infinite loop to introduce CPU load.

- For conversion of user name to user ID, and vice versa, check `<pwd.h>`.

- For directory traversal, check `<dirent.h>`.

- For sorting, check `qsort()` in `<stdlib.h>`.

4. Submission

Submit a zip file containing `info.c`, `myps.c` and a document or Makefile showing how to compile your files.

Your submission will be graded in the machines of CS Lab 2 (csl2wkXX.cse.ust.hk, where XX=[01..53])

Note: For all of the programming assignments, only the programming language C will be used. Since we are learning the Linux operating system, which is implemented with C, other languages (e.g., C++, Java, Python, and Ruby) simply play no role. Please spend some time refresh the knowledge of C, if you haven't used it for a long time.