Term Project1:

Process Analysis in Linux using PROCPS open-source project
(Making my_ps application)

Due: 30th Oct. (Mon), 11:59 PM

Submit via I-Campus.

1. Project Introduction

In this project, you will see how processes are managed in Linux, and make a program which logs the information of current running processes by modifying 'ps' application. The ps application shows the information about the current running processes via /proc filesystem, which is a special filesystem to show the information about the system and processes. The several useful utilities such as ps, top, or free are maintained in the open source project 'procps'.

2. What to do

2.0 Environments

2.0.1 Do the project in Ubuntu environment (kernel version 4.9.x)
2.0.2 You can install Ubuntu or use virtual machine for the project.
2.0.3 If you use virtual machine, allocate enough memory to the machine (at least 2GB).

2.1 Analyze the core functions to understand the life of process in Linux

2.1.1 Download the linux kernel from https://www.kernel.org/
2.1.2 Find the functions of '**copy_process()**' and '**__schedule()**' in the kernel
2.1.3 Analyze the two functions and explain briefly (core parts) in the report
(You don't need to explain everything in those functions. I will only check if you understand the functions properly)

2.2 Make your own ps application called 'my_ps'

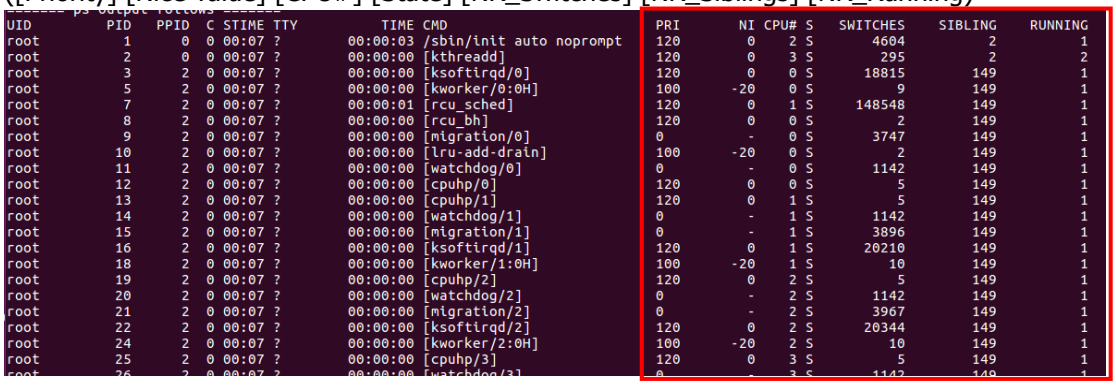
2.2.1 Download the open source code from https://gitlab.com/procps-ng/procps.
$ git clone https://gitlab.com/procps-ng/procps
2.2.2 Build an executable binary of ps. Check the markdown files (.md) to find out how to build the project code. You will need to install several dependent libraries.
2.2.3 Analyze the source code and its functions
(You need to find the core functions to make outputs)
2.2.4 Add new columns in the output to show some additional information.
(The specs are described in 2.3)

2.3 Output of 'my_ps' program

2.3.1 You must print 7 new outputs by adding new columns
1) Priority: Priority of current process (in range of 0~139)
2) Nice value: Nice value of current process (-20~19)
3) CPU number: CPU # where the process is scheduled
4) Process State: Current State of the process (S: Sleepling, R: Running, Z: Zombie)
5) NR_Switches: # of context switchings the process did
6) NR_Siblings: # of siblings of the process
7) NR_Running: # of running processes in a runqueue where the process is in.

2.3.2 Example output

([Priority] [Nice value] [CPU#] [State] [NR_Switches] [NR_Siblings] [NR_Running)



```
------- ps output follows -------
UID          PID    PPID  C STIME TTY          TIME CMD                     PRI      NI CPU# S    SWITCHES    SIBLING    RUNNING
root           1       0  0 00:07 ?        00:00:03 /sbin/init auto noprompt 120       0    2 S        4604          2          1
root           2       0  0 00:07 ?        00:00:00 [kthreadd]               120       0    3 S         295          2          2
root           3       2  0 00:07 ?        00:00:00 [ksoftirqd/0]            120       0    0 S       18815        149          1
root           5       2  0 00:07 ?        00:00:00 [kworker/0:0H]           100     -20    0 S           9        149          1
root           7       2  0 00:07 ?        00:00:01 [rcu_sched]              120       0    1 S      148548        149          1
root           8       2  0 00:07 ?        00:00:00 [rcu_bh]                 120       0    0 S           2        149          1
root           9       2  0 00:07 ?        00:00:00 [migration/0]            0         -    0 S        3747        149          1
root          10       2  0 00:07 ?        00:00:00 [lru-add-drain]          100     -20    0 S           2        149          1
root          11       2  0 00:07 ?        00:00:00 [watchdog/0]             0         -    0 S        1142        149          1
root          12       2  0 00:07 ?        00:00:00 [cpuhp/0]                120       0    0 S           5        149          1
root          13       2  0 00:07 ?        00:00:00 [cpuhp/1]                120       0    1 S           5        149          1
root          14       2  0 00:07 ?        00:00:00 [watchdog/1]             0         -    1 S        1142        149          1
root          15       2  0 00:07 ?        00:00:00 [migration/1]            0         -    1 S        3896        149          1
root          16       2  0 00:07 ?        00:00:00 [ksoftirqd/1]            120       0    1 S       20210        149          1
root          18       2  0 00:07 ?        00:00:00 [kworker/1:0H]           100     -20    1 S          10        149          1
root          19       2  0 00:07 ?        00:00:00 [cpuhp/2]                120       0    2 S           5        149          1
root          20       2  0 00:07 ?        00:00:00 [watchdog/2]             0         -    2 S        1142        149          1
root          21       2  0 00:07 ?        00:00:00 [migration/2]            0         -    2 S        3967        149          1
root          22       2  0 00:07 ?        00:00:00 [ksoftirqd/2]            120       0    2 S       20344        149          1
root          24       2  0 00:07 ?        00:00:00 [kworker/2:0H]           100     -20    2 S          10        149          1
root          25       2  0 00:07 ?        00:00:00 [cpuhp/3]                120       0    3 S           5        149          1
root          26       2  0 00:07 ?        00:00:00 [watchdog/3]             0         -    3 S        1142        149          1
```

2.3.3 The above output must be generated for the input command of **"./my_ps -ef"**

2.4 Some TIPs (For Your Information)

2.4.1 The information of process is managed by *task_struct* structure in Linux.

2.4.2 The runqueue is managed by *rq* structure in Linux

2.4.3 You can use the information from the /proc filesystem. If /proc does not provide what you want, you may get from the internal structures in kernel.

2.4.4 ps is a user-level application. You need to make **your own system calls** to get the value from the kernel. Use **the return value** of system call.
(You need to find the way to create and use your own system calls)

2.5 Evaluation policy

2.5.1 If you analyzed completely copy_process() and __schedule(). (10 points for each)

2.5.2 If you successfully built the procps project and inserted new columns in the output. (10 points)

2.5.3 The my_ps program outputs **priority, nice** and **state** correctly. (5 points for each)

2.5.4 The my_ps program outputs **CPU#** and **nr_switches** correctly. (10 points for each)

2.5.5 The my_ps program outputs **nr_siblings** and **nr_running** correctly. (20 points for each)

Job	copy_process __schedule	New columns	Priority Nice, State	CPU # nr_switches	nr_siblings nr_running	Total
point	20	10	15	15	40	100

3. What to submit

- The final output should be a compressed zip file named [your_student_id]_proj1.zip, which includes [student_id]_report.pdf and several modified source codes (i.e., ps, kernel code)
  In case of kernel code, don't attach the whole kernel code, just modified code.
- Please describe the key parts of your code in the report (modified code)
  Don't attach the whole source codes in the report. Just key parts.
  (you will submit your codes too.)

4. Notices

- You must do the term project for yourself
- If you have a question, send email to sinban04@gmail.com with the title of [CSE3047-41] Term project1, [Student ID], [Your Name]
- Your email questions should describe the problem in detail with screenshots.
- If you cheat on the project, you will finally fail this course. (F grade)
  - If I find two similar source codes or reports, both students will fail this course.
- You will get -15 points per one day delay.

Have fun!