National Taiwan Normal University
Department of Computer Science and Information Engineering
CSC0016, Assignment 3

1. The assignment is worth 100 points.
2. Individual work
3. Due at 12:00 11/10, i.e., Sunday noon
4. Deliver your assignment as a report in English or Chinese MS Word or PDF, whose maximum size is 6 pages without the reference and appendix.

1 Content

1. (100 points) This is an individual work. In the assignment, you will add a system call to a Linux kernel for collecting process information. The function of the system call is to collect the following information of the current process into the following structure. After completing the system call, demonstrate it by calling it in user level under TWO different shells, and print the collected information on screen. Therefore, you have to write a test program in user level as well, AND run the test program under two different shells.

```
struct prinfo {
    long state; /* current state of process */
    long nice; /* process nice value */
    pid_t pid; /* process id */
    pid_t parent_pid; /* process id of parent */
    pid_t youngest_child_pid; /* pid of youngest child */
    unsigned long start_time; /* process start time */
    long user_time; /* CPU time spent in user mode */
    long sys_time; /* CPU time spent in system mode */
    long uid; /* user id of process owner */
    char comm[16]; /* name of program executed */
};
```

   Deliver your assignment as a report in English or Chinese MS Word or PDF, whose maximum size is 10 pages without the reference and appendix. Submit the report to the course website before 12:00 11/10, i.e., Sunday noon. Your report have to include the following directions at least.

   - Hardware and software environment. Skip this part if it is identical to the previous assignment. Otherwise, describe it clearly.

- Implementation steps. Sequentially record your steps and affected files, e.g., created or modified files or codes, which make the system call work. Briefly describe the goal of each step.
- Demonstration under two different shells. Show the used shell and call the user program in user level. Show the used shell and the printed information from the called program as a screen-shot. One screen-shot for one shell, so you have to show two screen-shots.
- Comments. At least one most challenging part and your solution, your comments, etc.
- Any reference if you use.
- Put full codes in the appendix with a tighter format.

2 Tips

1. The implementations of the system calls `getuid()` and `getpid()` in the file `kernel/timer.c` can provide some guidance.
2. Almost all of the process information that you need to fill in the structure `prinfo` can be found in the structure `task_struct`, defined in the file `include/linux/sched.h`.
3. Kernel file `include/asm/current.h` defines an inline function that returns the address of the structure `task_struct` of the current process.
4. Every system call must check the validity of arguments passed by a caller. Linux kernel provides two functions, `copy_from_user()` and `copy_to_user()`, that not only check the validity but also transfer information between kernel and user levels. Use them to move the process information.

3 Do

- Cover all directions in the report
- Record the implementation steps correctly and in a correct sequence
- Collect the information from the current process into the structure.
- Transfer the collected information from kernel level to user level.
- Print the collected information on screen in user level.
- Compile the modified kernel and make it runnable.
- Call the system call in a user-level program.
- Call the system call under two different shells.

- Include two screen-shots. Each shows one used shell and the printed information from the called program.
- Complete the submission in time.

4 Do not

- Lack any part of the report
- Miss any necessary step
- Collect the information from a process not the current one.
- Print the information from the system call in kernel level directly.
- Forget to print the collected information on screen.
- Make a modified kernel that cannot be executed.
- Call the system call in kernel level.
- Forget to show the used shell.
- Forget to use two different shells.
- Miss two screen-shots in which each shows one used shell and the printed information from the called program.
- Delay or miss the submission.