|  |
| --- |
| **Project 2: Process Management**  **CSE 330 - Operating Systems** |
| **Instructor: Ming Zhao**  **TAs: Eugene Kuznetsov, Sungho Hong, Yitao (Tom) Chen**  **Check the online version of this document from time to time. We will update it to provide more tips and suggestions based on student feedback.** |

### **Summary**

Starting from the second project, we will do some serious kernel development. All the projects

designed for our course are interesting and practical, which you can proudly put on your resume. In Project 2, we will implement a new process management function on the Linux and Pi that you prepared in Project 1. This project will help you understand how a real-world operating system like Linux manages processes and master the skills for process management in kernel space.

### **Description (Part 1)**

In the first part of this project, you will implement a new system call to list all the processes (and their scheduling information) that belong to a given user in the system. This simple task will help you learn the key kernel data structures and macros for process management. You can reuse the dummy system call and the user-space program that you implemented in Project 1.

Specifically, your system call and user-space program should implement the following:

1. Your user-space program should take a username as the only command-line argument and pass this information to your system call.
2. Your system call handler should search the system for all the processes that belong to the given user and return the required information of each process, including the process ID (PID), priority (PRIO), scheduler class (CLS), and the executable name (CMD), to the user-space program.
3. Your user-space program should then format the information given by the system call and print the output. The output should look like the following:

|  |
| --- |
| PID COMMAND PRI CLS  775 gvfs-goa-volume 138 RR // SCHED\_RR  776 gvfs-afc-volume 138 FF // SCHED\_FIFO  777 gvfs-gphoto2-vo 19 B // SCHED\_BATCH  778 udisksd 19 TS // SCHED\_NORMAL  ...  819 sshd 19 TS  822 bash 19 TS  849 list\_proc 19 TS // Your user space program  // SCHED\_FIFO: A first-in-first-out real-time policy  // SCHED\_RR: A round-robin real-time policy  // SCHED\_NORMAL: the default policy, completely fair scheduler  // SCHED\_BATCH: batch process |

To verify the correctness of your results, you can use *ps -u <username>* to find out all the processes that belong to a given user, and use *ps -o pid,comm,pri,class -T -p <process id>* to find out the required information about each process.

**Notes:**

* The output should be generated by your user-space program to standard output, not by your system call handler to the kernel log.
* You can define how your user-space program communicates with your system call handler, including how to pass the data and how to format the data.
* Your implementation should be able to get the processes by using username as the argument.
* Do not assume that you have enough memory to hold the information for all the processes.
* You can invoke your system call more than once if necessary.

### To test your implementation, write a program to generate lots of processes with different scheduling classes and priorities. A sample test program will be provided later.

### **Useful Kernel Programming Interfaces**

1. Useful task attributes in task\_struct

|  |
| --- |
| **struct task\_struct** is located in include/linux/sched.h  **task->cred->uid.val** provides the user-id of the process  **task->pid** provides process id  **task->comm** provides the command of the process  **task->prio** provides the priority of the process |

1. Useful kernel macro

|  |
| --- |
| **for\_each\_process**(struct task\_struct \*p) // a macro that iterates the entire task list. On each iteration, p points to the next task in the list. |

1. Useful kernel macros and functions for passing to/from user space

|  |
| --- |
| **get\_user**(x, addr) // a macro that gets a single value from user space  // the kernel variable *x* gets the value of the data stored at *addr* in user space  // returns zero on success, or -EFAULT on error.  **put\_user**(x, addr) // a macro that puts a single value to user space  // the value of the variable *x* is written to the address *addr* in user space.  // returns zero on success, or -EFAULT on error.  unsigned long **copy\_from\_user** (void \*to, const void \*from, unsigned long n);  // copies *n* bytes from the user space memory starting at address *from* to the kernel space memory starting at address *to*.  // returns number of bytes that could not be copied. On success, this will be zero.  unsigned long **copy\_to\_user**(void \*to, const void \*from, unsigned long n);  // copies *n* bytes from the kernel space memory starting at address *from* to the user space memory starting at address *to*.  // returns number of bytes that could not be copied. On success, this will be zero. |

1. Useful user space function for getting the uid of a given username

|  |
| --- |
| struct passwd \***getpwnam**(const char \*nam) // search the password database for the entry that matches the given username  // searches the username *nam*  // returns a pointer to a struct passwd which contains the user’s uid *pw\_uid*  // More info can be found here: https://linux.die.net/man/3/getpwnam |

### **Example Test Program**

The following Github link is an example test program. The program generates 8 processes with different scheduling policies, including one default policy (TS), one FIFO policy (FF), two Round Robin policies (RR) and four Batch policies (B).

<https://github.com/ychen404/process_generator>

Note that this is only an example test program. The one we use for grading will be more intensive.

### **Submission Requirements**

Part I will be submitted together with Part II. More instructions to come.

The entire project is due in **four weeks**. Finish Part I in **one to two weeks**.