

Project 2:

UNIX Shell Programming

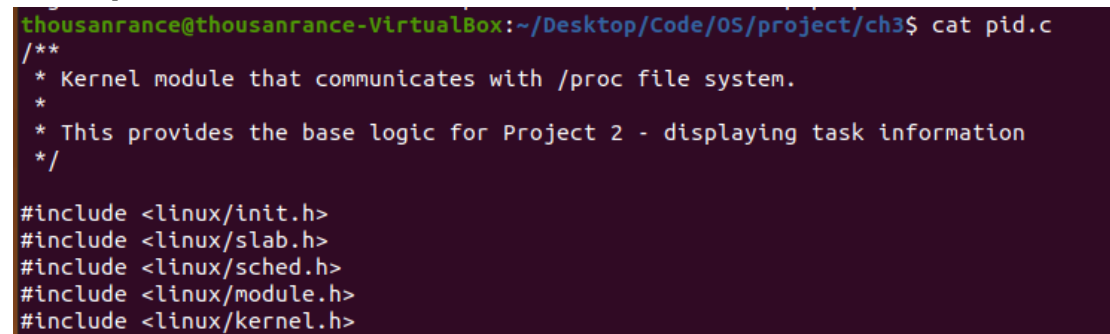
& Linux Kernel Module for Task Information

Project 2-1: UNIX Shell

I. Overview

注意：目录下并没有书上所说的文件 `prog.c`，故这里使用 `cat` 命令显示目录下的 `pid.c` 文件内容。

1、`cat pid.c`

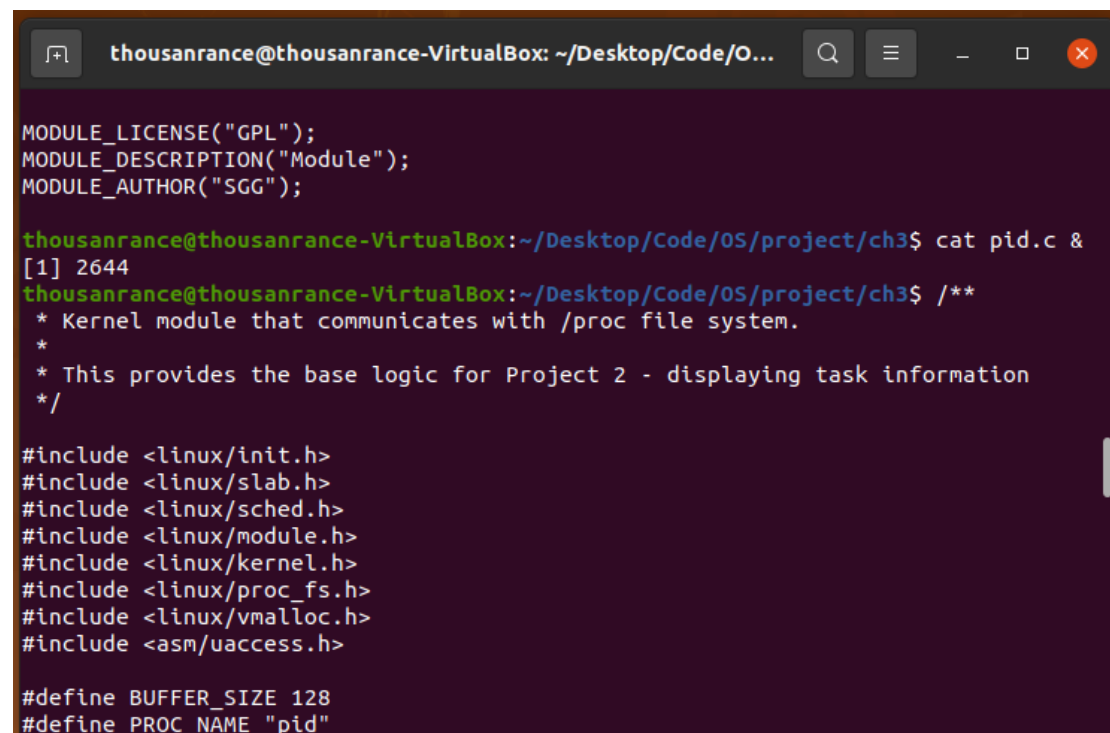


```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ cat pid.c
/**
 * Kernel module that communicates with /proc file system.
 *
 * This provides the base logic for Project 2 - displaying task information
 */

#include <linux/init.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/kernel.h>
```

直接显示等待下一个指令

2、`cat pid.c &`



```
thousanrance@thousanrance-VirtualBox: ~/Desktop/Code/O...
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Module");
MODULE_AUTHOR("SGG");

thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ cat pid.c &
[1] 2644
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ /**
 * Kernel module that communicates with /proc file system.
 *
 * This provides the base logic for Project 2 - displaying task information
 */

#include <linux/init.h>
#include <linux/slab.h>
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/vmalloc.h>
#include <asm/uaccess.h>

#define BUFFER_SIZE 128
#define PROC_NAME "pid"
```

不显示等待下一个指令，需要按回车键之后才显示等待下一个指令

II. Executing Command in a Child Process

一、实现细节

1、预处理

(1) 初始化

(2) 获取输入。若输入为“\n”则直接进入下一次循环。

2、输入处理

(3) 使用一个 for 循环处理输入，进行分词。

(4) 若最后一个分词为“&”，则更改进程并行条件 parent_wait 后将最后一个分词置空。

(5) 若第一个分词为 exit，程序退出。

3、在子进程中执行命令

(6) 创建子进程，在子进程中调用 execvp() 执行命令，父进程根据并行条件执行。

4、释放空间

(7) 循环末尾对存储输入的空间进行释放。

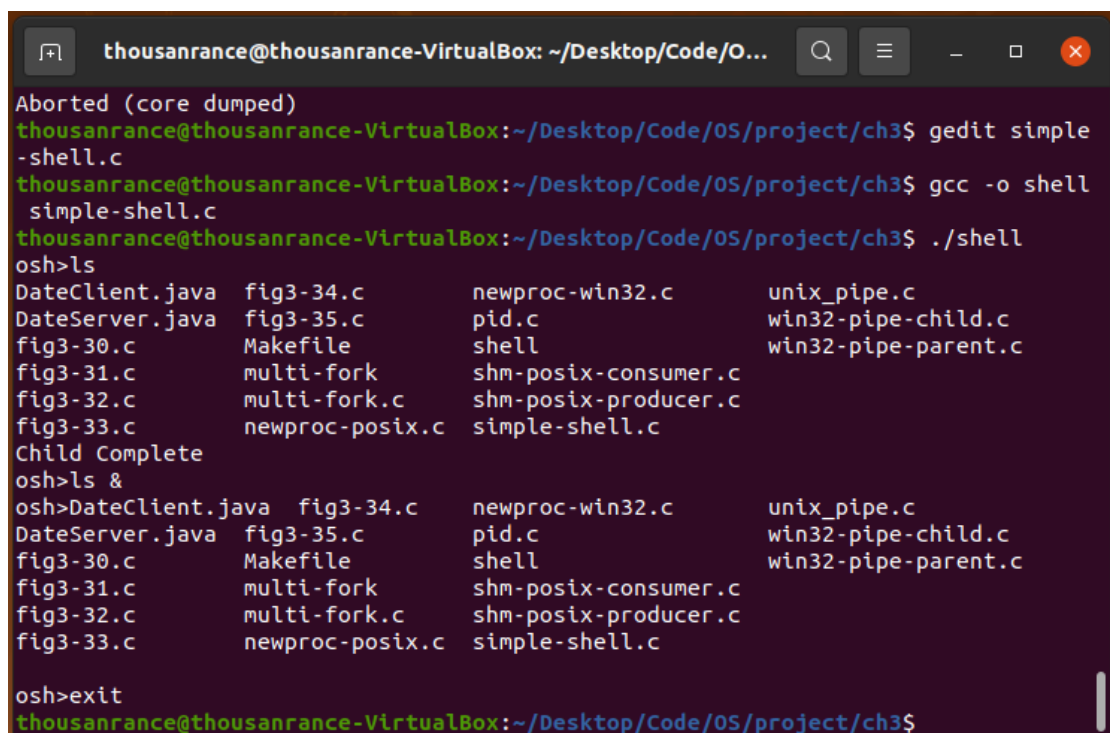
二、试运行结果

尝试指令：

ls

ls &

exit



```
thousanrance@thousanrance-VirtualBox: ~/Desktop/Code/O...
Aborted (core dumped)
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ gedit simple-shell.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ gcc -o shell simple-shell.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ./shell
osh>ls
DateClient.java  fig3-34.c      newproc-win32.c  unix_pipe.c
DateServer.java  fig3-35.c      pid.c            win32-pipe-child.c
fig3-30.c        Makefile       shell            win32-pipe-parent.c
fig3-31.c        multi-fork     shm-posix-consumer.c
fig3-32.c        multi-fork.c   shm-posix-producer.c
fig3-33.c        newproc-posix.c simple-shell.c
Child Complete
osh>ls &
osh>DateClient.java  fig3-34.c      newproc-win32.c  unix_pipe.c
DateServer.java  fig3-35.c      pid.c            win32-pipe-child.c
fig3-30.c        Makefile       shell            win32-pipe-parent.c
fig3-31.c        multi-fork     shm-posix-consumer.c
fig3-32.c        multi-fork.c   shm-posix-producer.c
fig3-33.c        newproc-posix.c simple-shell.c
osh>exit
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$
```

III. Creating a History Feature

一、实现细节

1、优先于输入处理，在输入处理之前添加代码，先判断输入是否为!!，否则进入输入处理

2、若输入是!!，判断是否有历史输入 history_exist，没有则提示错误信息

3、若有历史输入则将历史指令复制到输入，进入输入处理

4、输入处理后将输入保存到历史 history【保证了 history 里存的绝对是一条指令，而不是!!】

二、试运行结果

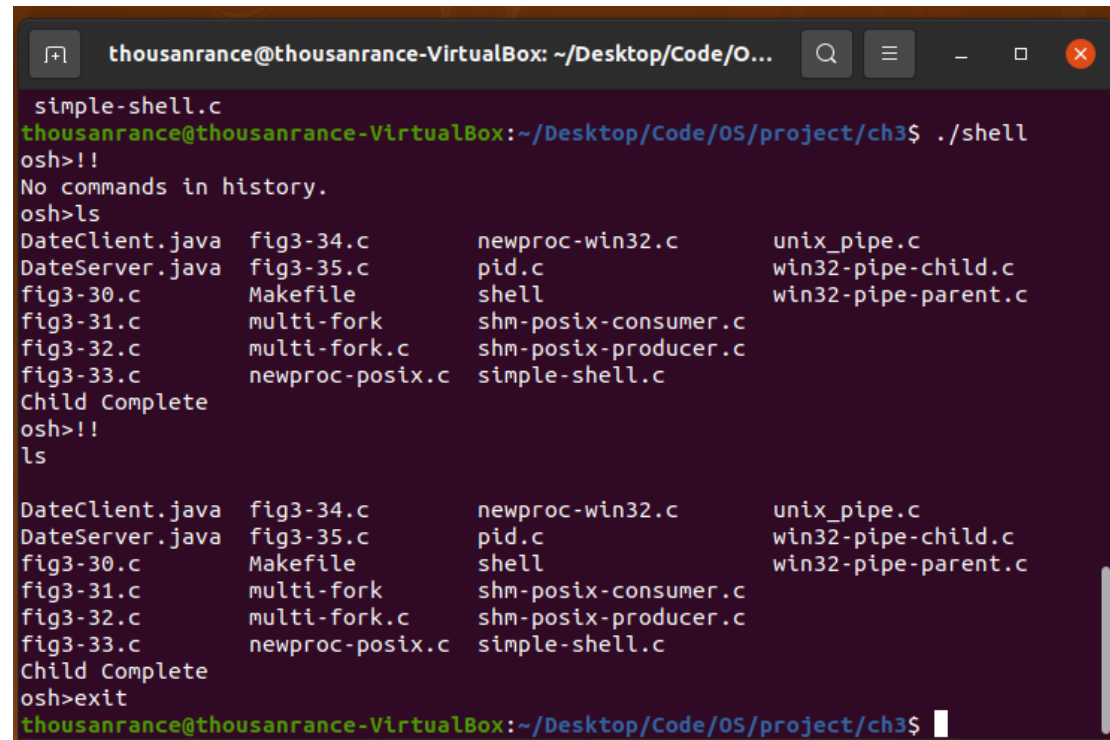
尝试指令：

!!

ls

!!

exit



```
thousanrance@thousanrance-VirtualBox: ~/Desktop/Code/O...
simple-shell.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ./shell
osh>!!
No commands in history.
osh>ls
DateClient.java  fig3-34.c      newproc-win32.c  unix_pipe.c
DateServer.java  fig3-35.c      pid.c            win32-pipe-child.c
fig3-30.c        Makefile       shell            win32-pipe-parent.c
fig3-31.c        multi-fork     shm-posix-consumer.c
fig3-32.c        multi-fork.c   shm-posix-producer.c
fig3-33.c        newproc-posix.c simple-shell.c
Child Complete
osh>!!
ls
DateClient.java  fig3-34.c      newproc-win32.c  unix_pipe.c
DateServer.java  fig3-35.c      pid.c            win32-pipe-child.c
fig3-30.c        Makefile       shell            win32-pipe-parent.c
fig3-31.c        multi-fork     shm-posix-consumer.c
fig3-32.c        multi-fork.c   shm-posix-producer.c
fig3-33.c        newproc-posix.c simple-shell.c
Child Complete
osh>exit
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$
```

IV. Redirecting Input and Output

一、实现细节

- 1、在子进程中，判断处理后的输入中是否有“<”“>”；因为这种情况下最后一个 args 肯定是文件名，所以只需要看倒数第二个 args 是否是“<”“>”。
- 2、如果有，则用 open 函数获取文件描述 file_input/file_output，使用该文件描述和函数 dup2() 对输入输出进行重定位，这一步需要在 execvp() 之前完成。
- 3、指令执行后，需要用 close() 函数关闭打开的文件。
- 4、最后需要将输入输出定位恢复到命令行，使用全局变量 shell_input/shell_output 中存下的命令行输入输出的定位，用 dup() 获取。

二、试运行结果

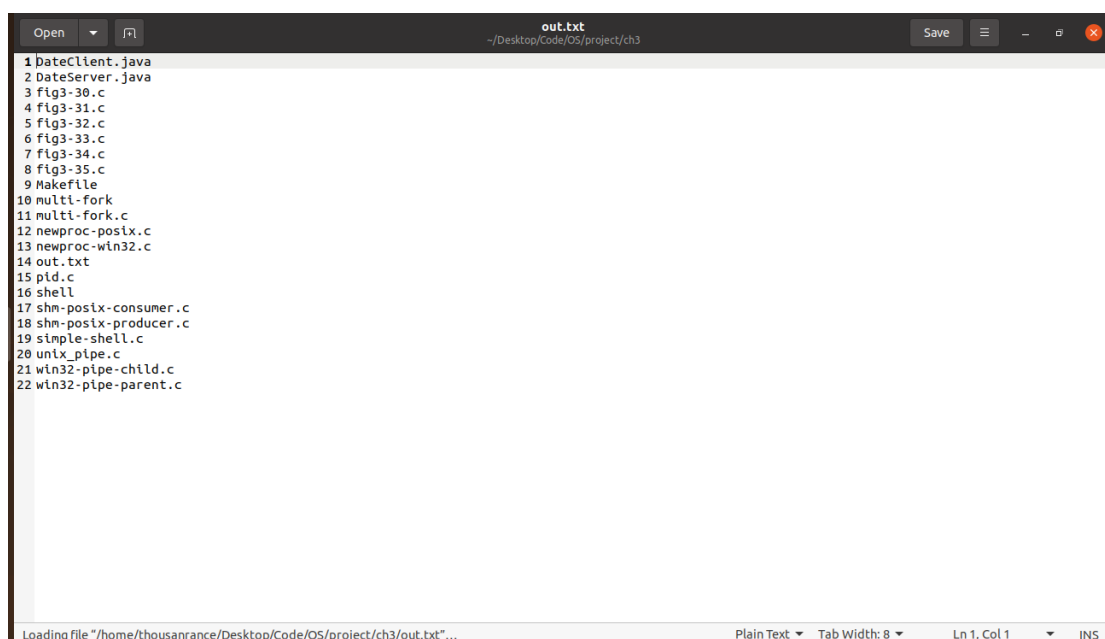
尝试指令：

ls > out.txt

```

thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ gedit simple
-shell.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ gcc -o shell
simple-shell.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ./shell
osh>ls > out.txt
Child Complete
osh>exit
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ls
DateClient.java  fig3-34.c      newproc-win32.c  simple-shell.c
DateServer.java  fig3-35.c      out.txt          unix_pipe.c
fig3-30.c         Makefile       pid.c           win32-pipe-child.c
fig3-31.c         multi-fork     shell           win32-pipe-parent.c
fig3-32.c         multi-fork.c   shm-posix-consumer.c
fig3-33.c         newproc-posix.c shm-posix-producer.c
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$

```



The screenshot shows a text editor window titled "out.txt" with the following content:

```

1 DateClient.java
2 DateServer.java
3 fig3-30.c
4 fig3-31.c
5 fig3-32.c
6 fig3-33.c
7 fig3-34.c
8 fig3-35.c
9 Makefile
10 multi-fork
11 multi-fork.c
12 newproc-posix.c
13 newproc-win32.c
14 out.txt
15 pid.c
16 shell
17 shm-posix-consumer.c
18 shm-posix-producer.c
19 simple-shell.c
20 unix_pipe.c
21 win32-pipe-child.c
22 win32-pipe-parent.c

```

The status bar at the bottom indicates: "Loading file "/home/thousanrance/Desktop/Code/OS/project/ch3/out.txt"... Plain Text Tab Width: 8 Ln 1, Col 1 INS".

sort < in.txt



The screenshot shows a text editor window titled "in.txt" with the following content:

```

1 opq
2 acb
3 abc
4 abcc

```

```

thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ gedit in.txt
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ./shell
osh>sort < in.txt
abc
abcc
acb
opq
Child Complete
osh>exit
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$

```

V. Communication via a Pipe

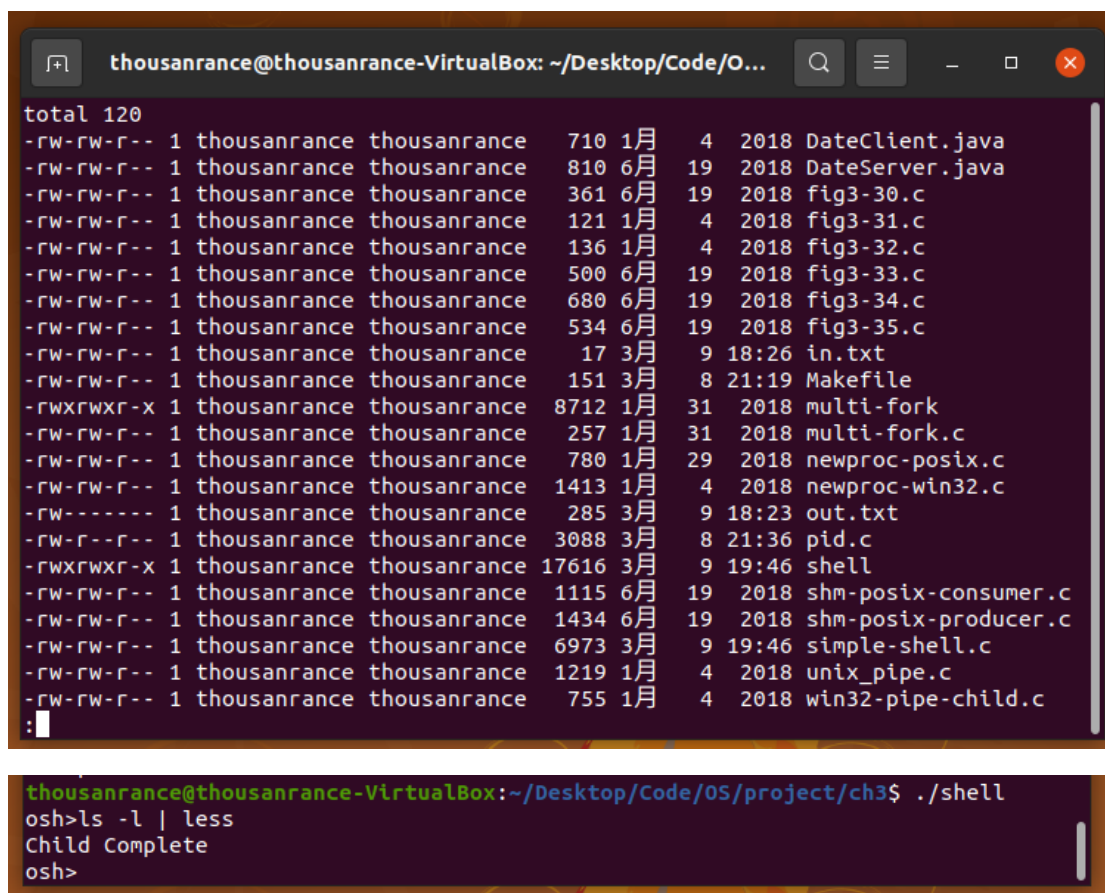
一、实现细节

- 1、在输入处理中增加代码，确定是否存在 pipe (pipe_exist)，并记下 pipe 符号在 args 中的位置 (pipe_pos)。
- 2、如果存在 pipe，处理完输入之后，需要将 | 前后的两个指令分开，后面的指令放入 pipe_args 中。
- 3、在子进程中创建 pipe。
- 4、在子进程中再创建子进程，在子进程的子进程中运行第一个指令，将输出重定位到 pipe 的写端，在当前进程中运行第二个指令，将 pipe 的读端读到的数据作为输入。

二、试运行结果

尝试指令：

```
ls -l | less
```



```
thousanrance@thousanrance-VirtualBox: ~/Desktop/Code/O...  
total 120  
-rw-rw-r-- 1 thousanrance thousanrance 710 1月 4 2018 DateClient.java  
-rw-rw-r-- 1 thousanrance thousanrance 810 6月 19 2018 DateServer.java  
-rw-rw-r-- 1 thousanrance thousanrance 361 6月 19 2018 fig3-30.c  
-rw-rw-r-- 1 thousanrance thousanrance 121 1月 4 2018 fig3-31.c  
-rw-rw-r-- 1 thousanrance thousanrance 136 1月 4 2018 fig3-32.c  
-rw-rw-r-- 1 thousanrance thousanrance 500 6月 19 2018 fig3-33.c  
-rw-rw-r-- 1 thousanrance thousanrance 680 6月 19 2018 fig3-34.c  
-rw-rw-r-- 1 thousanrance thousanrance 534 6月 19 2018 fig3-35.c  
-rw-rw-r-- 1 thousanrance thousanrance 17 3月 9 18:26 in.txt  
-rw-rw-r-- 1 thousanrance thousanrance 151 3月 8 21:19 Makefile  
-rwxrwxr-x 1 thousanrance thousanrance 8712 1月 31 2018 multi-fork  
-rw-rw-r-- 1 thousanrance thousanrance 257 1月 31 2018 multi-fork.c  
-rw-rw-r-- 1 thousanrance thousanrance 780 1月 29 2018 newproc-posix.c  
-rw-rw-r-- 1 thousanrance thousanrance 1413 1月 4 2018 newproc-win32.c  
-rw----- 1 thousanrance thousanrance 285 3月 9 18:23 out.txt  
-rw-r--r-- 1 thousanrance thousanrance 3088 3月 8 21:36 pid.c  
-rwxrwxr-x 1 thousanrance thousanrance 17616 3月 9 19:46 shell  
-rw-rw-r-- 1 thousanrance thousanrance 1115 6月 19 2018 shm-posix-consumer.c  
-rw-rw-r-- 1 thousanrance thousanrance 1434 6月 19 2018 shm-posix-producer.c  
-rw-rw-r-- 1 thousanrance thousanrance 6973 3月 9 19:46 simple-shell.c  
-rw-rw-r-- 1 thousanrance thousanrance 1219 1月 4 2018 unix_pipe.c  
-rw-rw-r-- 1 thousanrance thousanrance 755 1月 4 2018 win32-pipe-child.c  
:  
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ./shell  
osh>ls -l | less  
Child Complete  
osh>
```

Project 2-2: Linux Kernel Module for Task Information

I. Writing to the /proc File System

实现细节：

- 1、在 proc_ops 中添加 proc_write = proc_write
- 2、需修改 proc_write() 函数。因为无法保证从 usr_buf 复制 (copy_from_user()) 的字符串 str (用户通过 echo 输入的 pid) 以 null 结尾，所以需要用 sscanf() 将 str 再复制到 buffer，未被覆盖的部分已被初始化为 null。这样才能适用于 kstrtoul() 的第一个参数。
- 3、kstrtoul() 的第二个参数为进制，最后一个参数为第一个字符串参数转化成的整数，即需

要查询其信息的进程的 pid。

4、实例代码中用 kmalloc() 为字符串声明的空间，最后用 kfree() 释放。

II. Reading from the /proc File System

实现细节：

1、查看<linux/sched.h>文档中关于结构体 task_struct 的声明，可知需要的信息对应的变量名：command 为 comm，字符串；state 为 state，长整型。

```
struct task_struct {
    volatile long state; // 说明了该进程是否可以执行, 还是可中断等信息
    unsigned long flags; // Flage 是进程号, 在调用fork()时给出

    int ngroups; // 记录进程所在的组
    gid_t groups[NGROUPS]; // 记录进程所在的组
    // 进程的权限, 分别是有效位集合, 继承位集合, 允许位集合
    kernel_cap_t cap_effective, cap_inheritable, cap_permitted;
    int keep_capabilities;
    struct user_struct *user;
    struct rlimit rlim[RLIM_NLIMITS]; // 与进程相关的资源限制信息
    unsigned short used_math; // 是否使用FPU
    char comm[16]; // 进程正在运行的可执行文件名
    // 文件系统信息
    int link_count, total_link_count;
    // NULL if no tty 进程所在的控制终端, 如果不需要控制终端, 则该指针为空
    struct tty_struct *tty;
    unsigned int locks;
    // 进程间通信信息
    struct sem_undo *semundo; // 进程在信号灯上的所有undo操作
    struct sem_queue *semsleeping; // 当进程因为信号灯操作而挂起时, 他在该队列中记录等待的操作
    // 进程的CPU状态, 切换时, 要保存到停止进程的task_struct中
    struct thread_struct thread;
    // 文件系统信息
```

2、修改 proc_write()。如果 pid_task() 返回空, 说明没有传给它合法的 pid, 用 sprintf() 函数将错误信息写到 buffer, 并返回 0。如果返回正常, 则用 sprintf() 将要查询的信息写到 buffer, buffer 会被复制 (copy_to_user()) 到 usr_buffer, 用户能在 shell 看到查询的信息。

运行结果：

1、装载内核模块, 用 ps 指令查看当前有哪些进程, 选择一个 pid 准备写入

```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ dmesg
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ sudo insmod
pid.ko
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ dmesg
[15816.720276] /proc/pid created
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ ps
  PID TTY          TIME CMD
  4765 pts/0      00:00:00 bash
  6570 pts/0      00:00:00 ps
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$
```

2、写入 pid 并读取查询结果

```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ echo "4765"
> /proc/pid
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ cat /proc/pid
command = [bash] pid = [4765] state = [1]
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$
```

3、卸载内核模块

```
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ sudo rmmmod p
id
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$ dmesg
[15816.720276] /proc/pid created
[16121.528216] /proc/pid removed
thousanrance@thousanrance-VirtualBox:~/Desktop/Code/OS/project/ch3$
```