**Project 2:**

**UNIX Shell Programming**

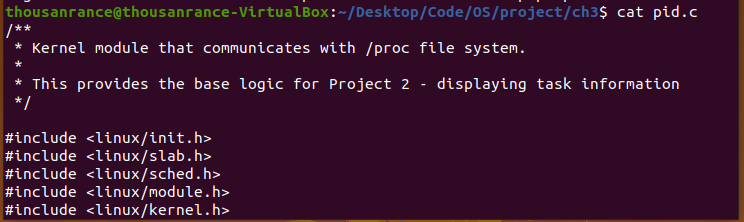
**& Linux Kernel Module for Task Information**

Project 2-1: UNIX Shell

Ⅰ. Overview

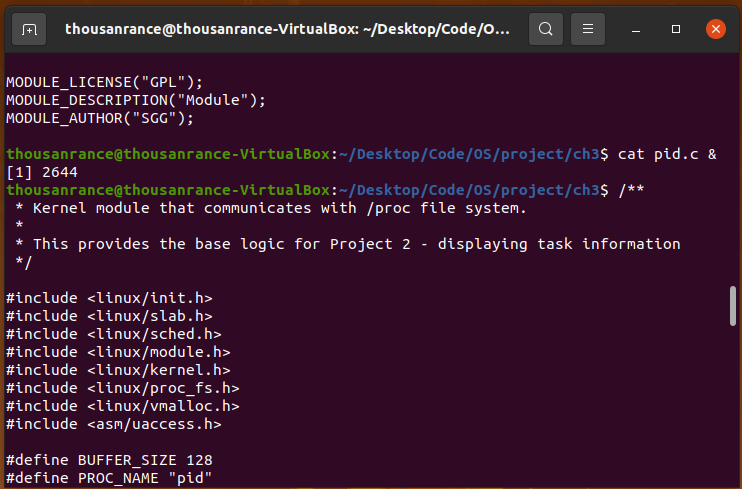
注意：目录下并没有书上所说的文件prog.c，故这里使用cat命令显示目录下的pid.c文件内容。

1、cat pid.c



直接显示等待下一个指令

2、cat pid.c &



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。

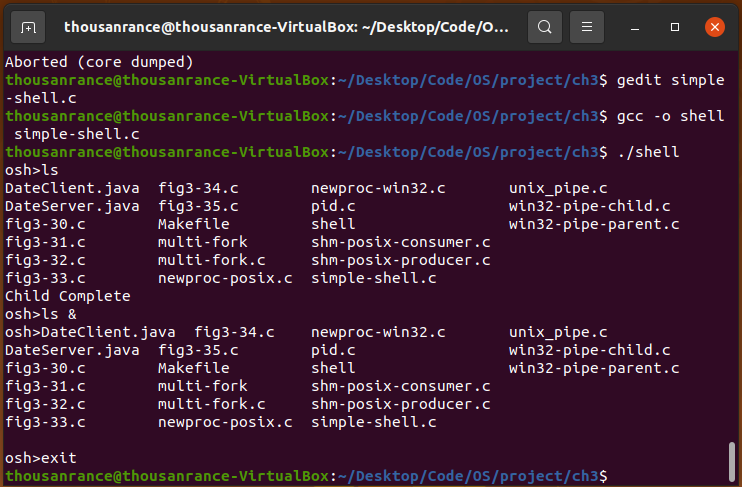
二、试运行结果

尝试指令：

ls

ls &

exit



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，而不会是！！】

二、试运行结果

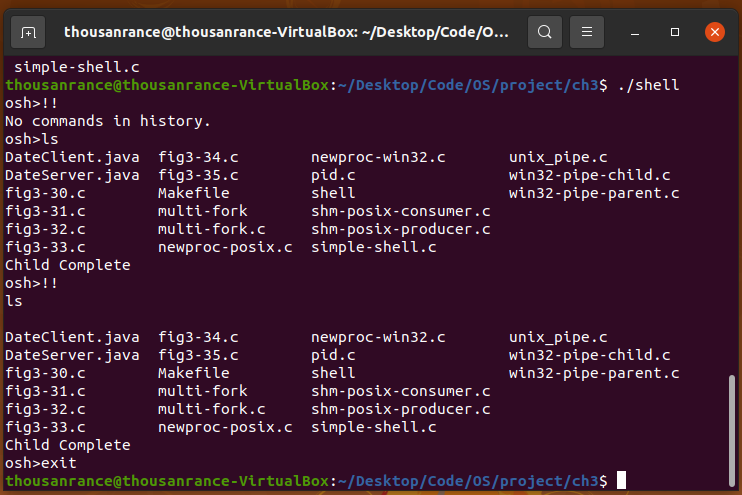
尝试指令：

！！

ls

！！

exit



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。

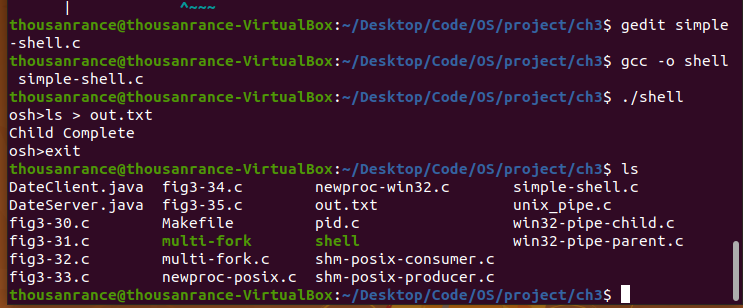
3、指令执行后，需要用close()函数关闭打开的文件。

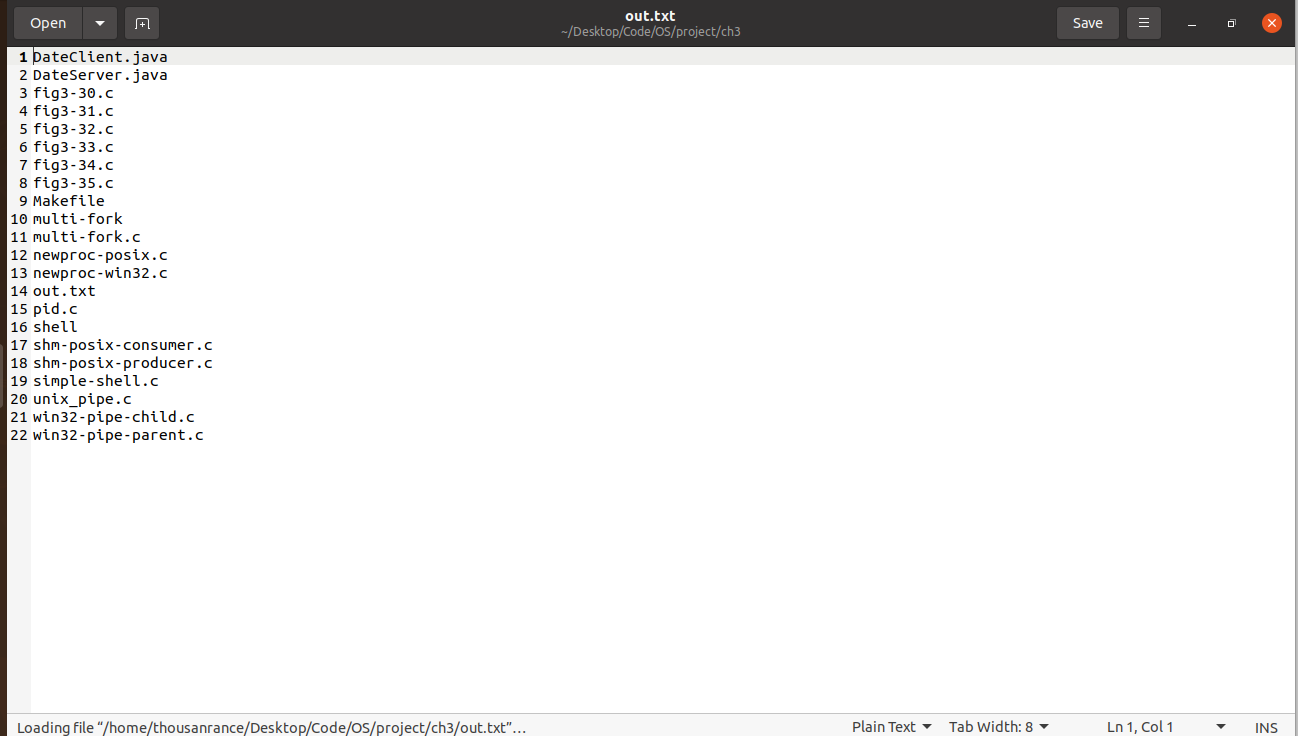
4、最后需要将输入输出定位恢复到命令行，使用全局变量shell\_input/shell\_output中存下的命令行输入输出的定位，用dup()获取。

二、试运行结果

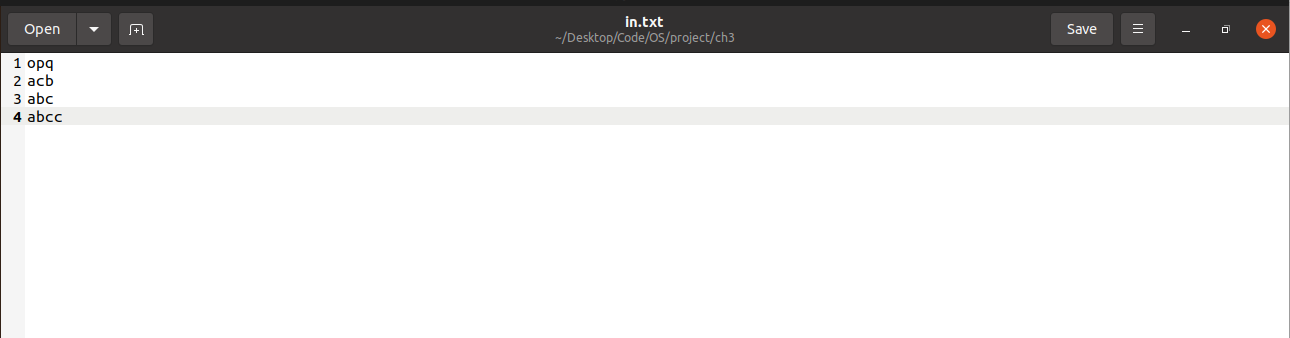
尝试指令：

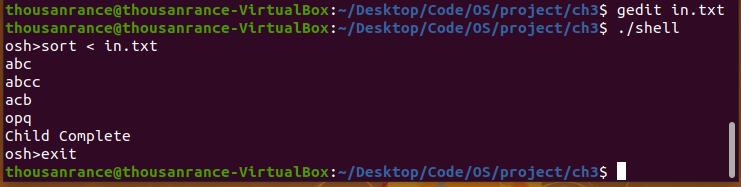
ls > out.txt





sort < in.txt





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。

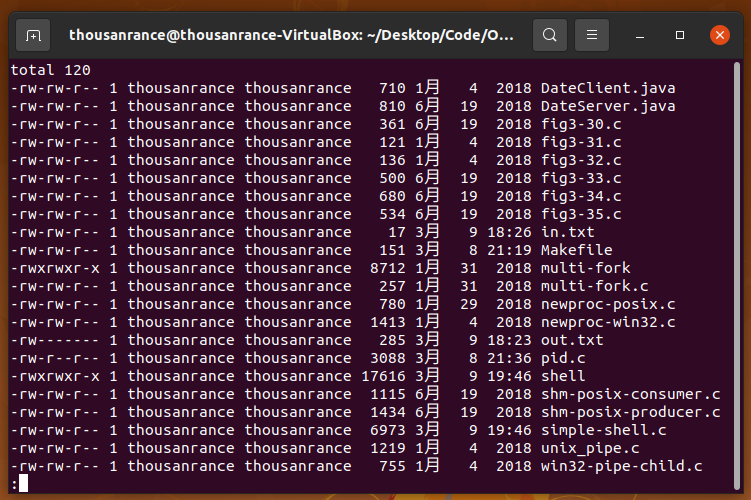
3、在子进程中创建pipe。

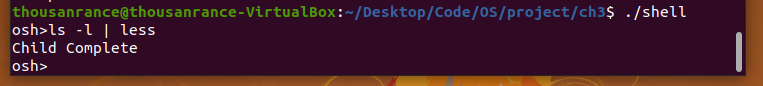
4、在子进程中再创建子进程，在子进程的子进程中运行第一个指令，将输出重定位到pipe的写端，在当前进程中运行第二个指令，将pipe的读端读到的数据作为输入。

二、试运行结果

尝试指令：

ls -l | less





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kstrtol()的第一个参数。

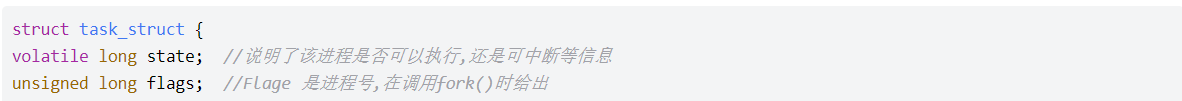
3、kstrtol()的第二个参数为进制，最后一个参数为第一个字符串参数转化为的整数，即需要查询其信息的进程的pid。

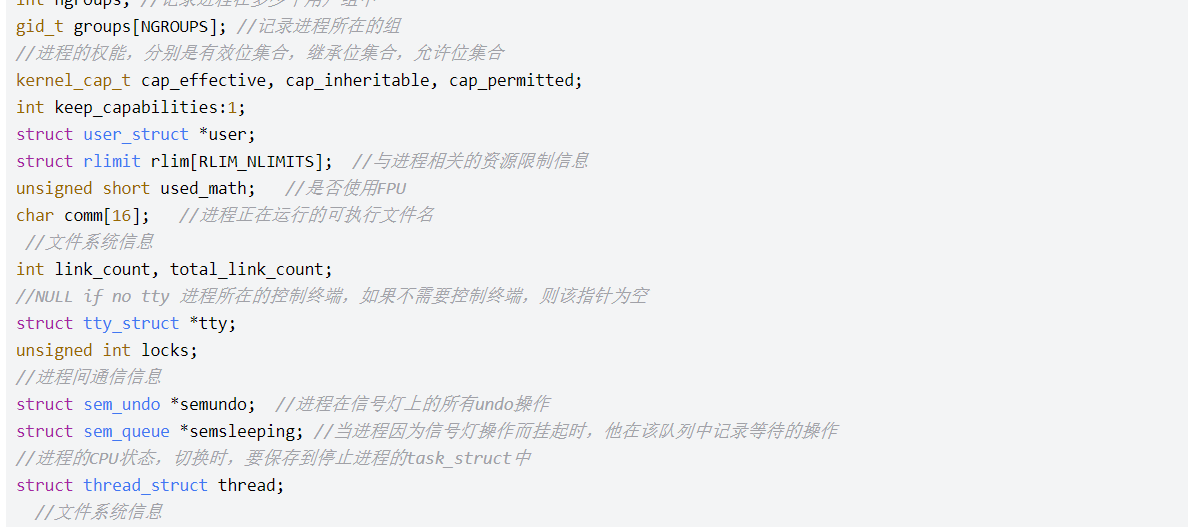
4、实例代码中用kmalloc()为字符串声明的空间，最后用kfree()释放。

Ⅱ. Reading form 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。

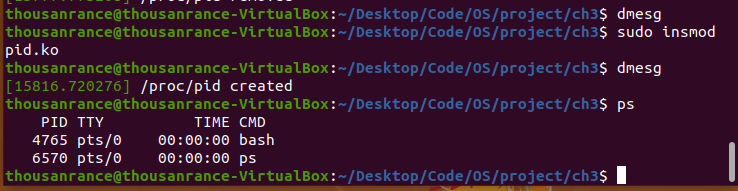




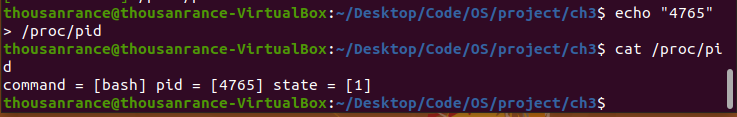
2、修改proc\_write()。如果pid\_task()返回空，说明没有传给它合法的pid，用sprintf()函数将错误信息写到buffer，并返回0。如果返回正常，则用sprintf()将要查询的信息写到buffer，buffer会被复制（copy\_to\_user()）到usr\_buffer，用户能在shell看到查询的信息。

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



2、写入pid并读取查询结果



3、卸载内核模块

