Project 2: Unix Shell 编程 & Linux 内核模块

Chentao Wu吴晨涛
Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

课程目标

- Project 2.1 Unix Shell
  - 创建子进程并在子进程中执行命令
  - 提供历史记录功能
  - 提供输入输出重定向功能
  - 允许父子进程通过管道进行通信
- Project 2.2 Linux Kernel Module for Task Informationn
  - 学习/proc文件系统的读、写
  - 使用/proc文件系统显示指定进程标识符的任务信息

Shell简介

- Shell 是一个命令解释器，在操作系统的最外层，负责与用户进行直接交互，把用户输入的命令解释给操作系统，然后进行处理和反馈。Linux默认的Shell是bash(GNU Bourne-Again Shell)

  e.g. ubuntu@VM-0-5-ubuntu:~$

- 下面的例子说明了提示符osh>和用户的下一条命令cat prog.c（该命令使用UNIX的cat命令在终端上显示文件prog.c）：

  osh>cat prog.c

- 实现Shell的一种技术是父进程读取命令行的输入，创建单独的子进程来执行该命令。父进程默认会等待子进程退出后再继续，UNIX shell也允许子进程在后台执行，在命令的末尾添加一个 (&)，父子进程会并行地执行

  osh>cat prog.c &

Project 2.1概览

- 功能：实现Shell接口 osh>，可以接受用户命令并执行
- 下图是教材图3.36的simple-shell.c代码：(解压final-src-osc10e.zip也可得到)

  should_run //值为1时一直循环，用户输入exit时修改值为0，退出循环

```c
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE        80 /* 80 chars per line, per command */

int main(void)
{
    char *args[MAX_LINE/2 + 1]; /* command line (of 80) has max of 40 arguments */
        int should_run = 1;

        while (should_run){
            printf("osh>");
            fflush(stdout);

            /**
             * After reading user input, the steps are:
             * (1) fork a child process
             * (2) the child process will invoke execvp()
             * (3) if command includes &, parent and child will run concurrently
             */
        }

    return 0;
}
```

创建子进程并在子进程中执行命令

- 步骤一：将用户输入的命令解析成单独的token，并将这些tokens存储在字符指针数组中（图3.36中的args）
- e.g. 本项目中用户在osh>提示符下输入ps -ael命令，存储在args中的值是：

  args[0] = "ps"，args[1] = "-ael"，args[2] = NULL

- 步骤二：调用fork()函数来创建子进程，子进程中调用execvp()函数执行用户指令，注意 (&)并行执行的情况
- pid_t fork(void);

  //父进程返回子进程id，子进程返回0
- int execvp(const char* command, char* params);

  //command表示要执行的命令，params表示存储该命令的参数

  //本项目子进程应该执行execvp(args[0], args);

提供历史记录功能

- Shell程序需要提供历史记录功能。
- 输入!!时执行上一条指令，没有指令时输出“No commands in history.”
- 当前的!!指令也应该存放在历史buffer中，作为下一条指令的历史指令

提供输入输出重定向功能

- Shell程序需要提供输入输出重定向功能。
- '>'将命令的输出重定向到一个文件，'<'将命令的输入重定向到一个文件

  e.g. osh>ls > out.txt，ls命令的输出将会被重定向到out.txt

  e.g. osh>sort < in.txt，sort命令的输入将会被重定向到in.txt

- 管理输入和输出的重定向将涉及到使用dup2()函数，它将一个现有的文件描述符复制到另一个文件描述符。
- int dup2(int oldfd , int newfd); //相当于用oldfd覆盖newfd
- e.g. dup2(fd, STDOUT FILENO);

  如果fd是指向文件out.txt的文件描述符，调用上述函数将fd复制到标准输出（终端）。这意味着任何写到终端的内容实际上都将被发送到out.txt文件中。

- 本题不需要考虑sort < in.txt > out.txt的复杂的情况，可以假设用户命令只有一个输入重定向或一个输出重定向

允许父子进程通过管道进行通信

- Shell程序需要允许一个命令的输出作为另一个命令的输入，使用管道来实现
- e.g.，osh>ls -l | less

  上述语句的功能是将ls -l命令的输出作为less命令的输入

  ls和less是两个独立的进程，使用管道进行通信。
- 一种简单实现方法是让父进程创建子进程（它将执行ls -l），这个子进程继续创建另一个子进程（将执行less），并在它自己和它创建的子进程之间建立一个管道。管道同样需要用到dup2()函数。

- int pipefd[2];
- int pipe(int pipefd[2]); //pipefd是传出参数

  pipefd[0] 对应的是管道的读端，pipefd[1] 对应的是管道的写端

  close(pipefd[1]); //关闭写端，close(pipefd[0]); //关闭读端
- 本题不需要多个管道、管道和重定向结合的复杂情况，假设用户命令只有一个管道命令即可

任务信息的Linux内核模块

- 设计一个内核模块：创建一个名为 /proc/pid 的 /proc 文件。将一个进程标识符写入/proc/pid，读取/proc/pid文件时将输出 (1)该任务正在运行的命令 (2)pid值 (3)当前状态

- e.g. echo "1395" > /proc/pid
  cat /proc/pid
  command = [bash] pid = [1395] state = [1]

- echo将"1395"写到/proc/pid文件，自定义的内核模块将读取这个字符串值并存储为整数。执行cat命令，从/proc/pid中读取，自定义的内核模块将检索出与pid值为1395的任务相关的上述三个字段

- 基于pid.c代码作修改：(解压final-src-osc10e.zip可得到)

pid.c

- 全局变量l_pid—查询的pid值
- 31行的 proc_create 传入了 proc_ops，这是一个 file_operations 结构体，初始化了 .owner 和 .read 两个成员。其中 ./read 赋值成了 proc_read，这意味着当读取 /proc/read 时，就会调用 proc_read 函数
- 本题/proc需要同时支持read和write，因此需要相应修改 proc_ops

```c
#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"

/* the current pid */
static long l_pid;

/**
 * Function prototypes
 */
static ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);
static ssize_t proc_write(struct file *file, const char __user *usr_buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};

/* This function is called when the module is loaded. */
static int proc_init(void)
{
        // creates the /proc/procfs entry
        proc_create(PROC_NAME, 0666, NULL, &proc_ops);

        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

    return 0;
}

/* This function is called when the module is removed. */
static void proc_exit(void)
{
        // removes the /proc/procfs entry
        remove_proc_entry(PROC_NAME, NULL);

        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}
```

proc_read

- 之前的/proc/hello，将Hello World写入了内核内存buffer
- 本题需要添加类似的代码语句，将查询的三个字段写入内核内存的buffer中

```c
/**
 * This function is called each time the /proc/pid is read.
 *
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 */
static ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;
        struct task_struct *tsk = NULL;

        if (completed) {
                completed = 0;
                return 0;
        }

        tsk = pid_task(find_vpid(l_pid), PIDTYPE_PID);

        completed = 1;

        // copies the contents of kernel buffer to userspace usr_buf
        if (copy_to_user(usr_buf, buffer, rv)) {
                rv = -1;
        }

        return rv;
}
```

proc_write

- 写入/proc/pid的是字符串类型，需要转为整型，所以要用到kstrtol函数
- 注释提示：如果传入的参数不是NULL结尾的，kstrol()将无法起作用。需要同学们注意

```c
/**
 * This function is called each time we write to the /proc file system.
 */
static ssize_t proc_write(struct file *file, const char __user *usr_buf, size_t count, loff_t
*pos)
        char *k_mem;

        // allocate kernel memory
        k_mem = kmalloc(count, GFP_KERNEL);

        /* copies user space usr_buf to kernel buffer */
        if (copy_from_user(k_mem, usr_buf, count)) {
        printk( KERN_INFO "Error copying from user\n");
                return -1;
        }

    /**
     * kstrol() will not work because the strings are not guaranteed
     * to be null-terminated.
     *
     * sscanf() must be used instead.
     */

        kfree(k_mem);

        return count;
}
```

作业及评分

自行阅读课本第三章的 Programming Projects 部分，并完成以下四个任务，完成后共计11分。

- （课本习题 4分）实现Shell接口 osh>，可以接受用户命令并执行

  创建子进程并在子进程中执行命令 (1分)

  提供历史记录功能 (1分)

  提供输入输出重定向功能 (1分)

  允许父子进程通过管道进行通信 (1分)

- （课本习题 4分）设计一个内核模块：创建一个名为 /proc/pid 的 /proc 文件。将一个进程标识符写入/proc/pid，读取/proc/pid文件时将输出 (1)该任务正在运行的命令 (2)pid值 (3)当前状态。结果可用cat获取，命令如下：

  cat /proc/pid

  确保在删除模块时删除 /proc/pid。

- （报告 2分）做一个简单的报告解释你的代码，报告要求不能超过2页（防内卷）。

- （Bonus 1分）本题中父子进程使用匿名管道通信，Linux系统中命名管道也经常会用到。请查阅相关资料，简单讨论二者的差异。

测试用例1

- 创建子进程并在子进程中执行命令

  osh>ls -l　　//显示当前路径下的目录和文件

  osh>ls -l &　　//显示当前路径下的目录和文件 (后台执行)

  osh>exit　　//退出shell

```
ygf@ubuntu:~/Proj2/proj2/unix_shell$ gcc osh.c -o osh
ygf@ubuntu:~/Proj2/proj2/unix_shell$ ./osh
osh>ls -l
total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 out.txt
osh>ls -l &
osh>total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 out.txt
ls
osh>in.txt      osh  osh.c  out.txt
ls
osh>in.txt      osh  osh.c  out.txt

osh>exit
ygf@ubuntu:~/Proj2/proj2/unix_shell$
```

测试用例2

- 提供历史记录功能

  osh>!!　　//没有指令时输出”No commands in history.”

  osh>ls -l　　//显示当前路径下的目录和文件 (后台执行)

  osh>!!　　//执行上一条指令

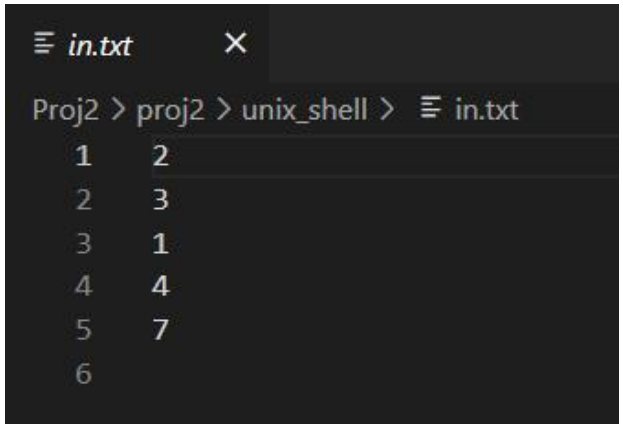
```
ygf@ubuntu:~/Proj2/proj2/unix_shell$ ./osh
osh>!!
No commands in history.
osh>!!
No commands in history.
osh>ls -l
total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 out.txt
osh>!!
total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 out.txt
osh>!!
total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 out.txt
osh>exit
ygf@ubuntu:~/Proj2/proj2/unix_shell$
```

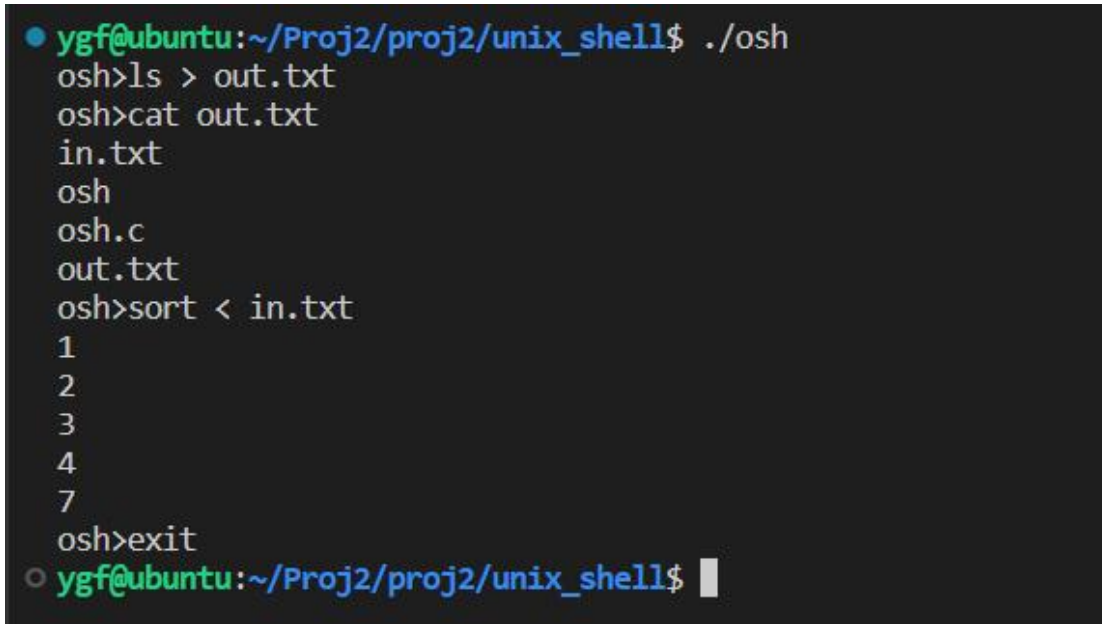
测试用例3

- 提供输入输出重定向功能

  osh>ls > out.txt　　//将ls命令的输出写入out.txt文件中

  osh>sort < in.txt　　//将in.txt作为sort的输入，在终端显示排序结果





测试用例4

- 允许父子进程通过管道进行通信

  osh>ls -l | sort　　//将ls -l的输出作为sort输入，在终端显示排序结果

  osh>ls -l | less　　//将ls -l的输出作为less输入，在终端查看文件内容

```
ygf@ubuntu:~/Proj2/proj2/unix_shell$ ./osh
osh>ls -l | sort
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rw-rw-r-- 1 ygf ygf    25 Apr 18 00:39 out.txt
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
total 32
osh>ls -l | less
osh>
```

```
total 32
-rw-rw-r-- 1 ygf ygf    10 Apr 17 01:01 in.txt
-rwxrwxr-x 1 ygf ygf 13736 Apr 17 23:58 osh
-rw-rw-r-- 1 ygf ygf  5016 Apr 17 01:01 osh.c
-rw-rw-r-- 1 ygf ygf    25 Apr 18 00:39 out.txt
~
~
~
~
~
~
~
~
~
~
~
~
~
(END)
```