# 操作系统课程设计 Project 2
# UNIX Shell Programming
# & Linux Kernel Module for Task Information

姓名：郭倩昀
班级：F1903303
学号：519021910095
Email：guoqianyun@sjtu.edu.cn

2021 年 5 月 21 日

## 目录

# 1 UNIX Shell Programming

## 1.1 实验内容与目标

本实验需要利用 C 语言设计可以接受并执行用户指令的 unix shell interface

- 创建子进程执行用户指令
- 支持历史特征
- 支持输入输出重定位
- 支持 pipe 通信

## 1.2 实验过程及步骤

- 指令标准化
  设计函数 void reorganize(char *inst) 标准化处理输入的指令，使得其中的空格个数等正常排列不存在特殊的多个空格或者换行情况。
- 指令分析并传输至 args
  设计函数 int parse(char *inst,char **args)，将标准化后的指令解析后放入 args 便于后续执行。
- 判断是否需要 concurrent 并行
  变量 concurrent 记录是否需要并行的情况，如果不需要并行，则父进程需要执行 wait(NULL) 指令等待子进程完成运行，否则不需要，父进程可以与子进程并行。
- 支持特殊指令!! 和 exit
  变量 last_inst 记录历史指令，have_last_inst 记录历史指令是否存在，这两个变量每次输入指令需要维护更新。检测到指令!! 则输出历史指令并将历史指令复制到当前指令执行，检测到指令 exit 则将 should_run 置零以跳出程序。
- 创建子进程执行用户命令
  使用 fork() 创建子进程，在子进程中调用 parse 函数将指令传入 args（根据是否并行判断是否舍去最后的'&'）。并使用 execvp 指令执行命令。
- 支持父子进程间使用 pipe 通信
  首先寻找是否有'|' 需要 pipe 通信。若需要 pipe 通信，创建 pipe（参考书本示例），使用 fork() 再次创建子进程用以执行'|' 之前的命令，并利用 dup2 将输出通过 pipe 定位给原来的子进程用以执行'|' 之后的命令。注意 pipe 的应用要注意端口父子进程各自使用的端口管理；不同的进程执行的命令范围不同需要调整相应的 args 和 argn 参数后才可以使用 execvp 指令执行命令；在执行完毕后及时 free 相应分配的空间。
- 支持输入输出重定位
  首先寻找是否有'>' 或者‘<’需要重定位，使用变量 in_redirect 和 out_redirect 记录是否需要重定位以及重定位类型。若需要重定位，相应使用 in_fd 或 out_fd 打开定位文件，调整相应的 args 和 argn 参数，利用 dup2 将输入或者输出定位到相应文件，再用 execvp 指令执行命令。

- 其他注意事项
  进程执行结束及时 free 所分配的空间，在分析指令的时候增加不合法指令的检测情况以及在打开文件、定位输入输出、创建子进程、创建 pipe 等关键步骤时候及时检测并及时报错。为了增加代码可读性，在代码中添加适量的注释。

## 1.3 实验代码

shell.c

```c
# include <stdio.h>
# include <fcntl.h>
# include <stdlib.h>
# include <string.h>
# include <unistd.h>
# include <sys/wait.h>
# include <sys/types.h>
#define MAX_LINE    80 /* 80 chars per line, per command */
#define READ_END    0  // for pipe read
#define WRITE_END   1  // for pipe write

void reorganize(char *inst);        //reorganize the instrction to a standard form
int parse(char *inst,char **args);  //parse the instruction to args
void clearstr(char *str);           //clear the string



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

    char *inst, *last_inst;
    int concurrent=0;           //whether concurrent
    int have_last_inst=0;       //whether have last inst
    char *in_file, *out_file;   //redirect filename

    inst=(char*) malloc(MAX_LINE * sizeof(char));       //instruction
    last_inst=(char*) malloc(MAX_LINE * sizeof(char));  //for history
    in_file=(char*) malloc(MAX_LINE * sizeof(char));    //redirect filename
    out_file=(char*) malloc(MAX_LINE * sizeof(char));   //redirect filename
    //initialize
    clearstr(last_inst);
    clearstr(inst);

    pid_t pid;

    while (should_run){
        printf("osh>");
        fflush(stdout);
        if(concurrent) wait(NULL);

        concurrent=0;
        clearstr(inst);

        fgets(inst,MAX_LINE,stdin);
```

```
47
48          reorganize(inst);
49
50          //check if concurrent
51          if(strlen(inst)>0 && inst[strlen(inst)-1]=='&')
52          {concurrent=1;}
53          else concurrent=0;
54
55          //exit
56          if(strcmp(inst,"exit")==0)
57          {
58              should_run=0;
59              continue;
60          }
61          //!! execute last inst
62          if(strcmp(inst,"!!")==0)
63          {
64              if(have_last_inst==0)
65              {
66                  fprintf(stderr,"  ERROR: No commands in history\n");
67                  continue;
68              }
69              else
70              {
71                  printf("%s\n",last_inst);
72                  strcpy(inst, last_inst);
73              }
74          }
75          //create child process
76          pid = fork();
77          if(pid<0){
78              fprintf(stderr,"  ERROR: Fork Failed\n");
79          }
80          else
81          {
82              if(pid==0)//child
83              {
84                  int error=0;
85                  //malloc args
86                  for(int i=0;i<MAX_LINE/2+1;i++)
87                  {
88                      args[i]=(char*)malloc(MAX_LINE*sizeof(char));
89                  }
90                  //parse to args
91                  int argn=parse(inst,args);
92
93                  for (int i = argn; i <= MAX_LINE / 2; ++ i) {
94                      free(args[i]);
95                      args[i] = NULL;
96                  }
97                  if (concurrent == 1) {
98                      -- argn;
99                      free(args[argn]);
100                     args[argn] = NULL;
101                 }
102
103                 //check | pipe
```

```
104                 int pipe_index=-1;
105                 for(int i=0;i<argn;i++)
106                 if(strcmp(args[i],"|")==0)
107                 {
108                     pipe_index=i;
109                     break;
110                 }
111                 if(pipe_index>=0)// found |
112                 {
113                     if(pipe_index==0||pipe_index>=argn-1)//
114                     {
115                         fprintf(stderr, "  ERROR: | illigal\n");
116                         error=1;
117                     }
118 
119                     //pipe fd create
120                     int pipe_fd[2];
121                     if(pipe(pipe_fd)==-1)
122                     {
123                         fprintf(stderr,"  ERROR: Pipe failed\n");
124                         error=1;
125                     }
126 
127                     if(error==0)
128                     {
129                         pid=fork();
130 
131                         if(pid<0){
132                         fprintf(stderr,"  ERROR: Fork Failed\n");
133                         error=1;
134                         }
135                         else if(pid==0)//grandchild
136                         {   //reorganize args
137                             for(int i=pipe_index;i<argn;i++)
138                             {
139                                 free(args[i]);
140                                 args[i]=NULL;
141                             }
142                             argn=pipe_index;
143                             close(pipe_fd[READ_END]);
144                             if(error==0&&dup2(pipe_fd[WRITE_END],STDOUT_FILENO)<0)
145                             {
146                                 fprintf(stderr,"  ERROR: dup2 Failed\n");
147                                 error=1;
148                             }
149                             if(error==0 && argn>0)
150                                 execvp(args[0],args);
151                             close(pipe_fd[WRITE_END]);
152 
153                             for(int i=0;i<argn;++i) free(args[i]);
154                             free(inst);
155                             free(last_inst);
156                             free(in_file);
157                             free(out_file);
158 
159                             exit(error);
160 
```

```
                    }
                    else//child
                    {
                        wait(NULL);
                        //reorganize args
                        for(int i=0;i<=pipe_index;i++) free(args[i]);
                        for(int i=pipe_index+1;i<argn;++i) args[i-pipe_index-1]=args[i];
                        for(int i=argn-pipe_index-1;i<argn;i++) args[i]=NULL;
                        argn=argn-pipe_index-1;

                        close(pipe_fd[WRITE_END]);
                        if(error==0&&dup2(pipe_fd[READ_END],STDIN_FILENO)<0)
                        {
                            fprintf(stderr,"  ERROR: dup2 Failed\n");
                            error=1;
                        }
                        if(error==0 && argn>0)
                            execvp(args[0],args);
                        close(pipe_fd[READ_END]);
                    }

                }

            }
            else// | not found
            {
                int in_redirect=0;
                int in_fd=-1;
                int out_redirect=0;
                int out_fd=-1;
                //check < and reorganize args
                if(argn>2 && strcmp(args[argn-2],"<")==0)
                {
                    in_redirect=1;
                    strcpy(in_file,args[argn-1]);
                    argn-=2;
                    free(args[argn]);
                    args[argn]=NULL;
                    free(args[argn+1]);
                    args[argn+1]=NULL;
                }
                //check > and reorganize args
                if(argn>2 && strcmp(args[argn-2],">")==0)
                {
                    out_redirect=1;
                    strcpy(out_file,args[argn-1]);
                    argn-=2;
                    free(args[argn]);
                    args[argn]=NULL;
                    free(args[argn+1]);
                    args[argn+1]=NULL;
                }
                //in_redirect file open and redirect
                if(error==0&&in_redirect==1)
                {
                    in_fd=open(in_file,O_RDONLY,0644);
                    if(error==0 && in_fd<0)
```

```
                        {
                            fprintf(stderr,"  ERROR: no file\n");
                            error=1;
                        }
                        if(error==0 && dup2(in_fd,STDIN_FILENO)<0)
                        {
                            fprintf(stderr,"  ERROR: dup2 Failed\n");
                            error=1;
                        }
                    }
                    //out_redirect file open and redirect
                    if(error==0&&out_redirect==1)
                    {
                        out_fd=open(out_file,O_WRONLY|O_TRUNC|O_CREAT,0644);

                        if(error==0 && dup2(out_fd,STDOUT_FILENO)<0)
                        {
                            fprintf(stderr,"  ERROR: dup2 Failed\n");
                            error=1;
                        }
                    }
                    //execute
                    if(error==0 && argn>0)
                        execvp(args[0],args);

                    if(in_redirect==1 && in_fd>0) close(in_fd);
                    if(out_redirect==1 && out_fd>0) close(out_fd);
                }

                if(error==0 && argn>0)
                    execvp(args[0],args);
                //free
                for(int i=0;i<argn;++i) free(args[i]);
                free(inst);
                free(last_inst);
                free(in_file);
                free(out_file);

                exit(error);
            }
            else//parent
            {
                if(concurrent==0) wait(NULL);
            }
        }
        //for history
        if(have_last_inst==0) have_last_inst=1;
        strcpy(last_inst,inst);
    }
    //free
    free(inst);
    free(last_inst);
    free(in_file);
    free(out_file);

    return 0;
}
```

```
275
276 //reorganize the instrction to a standard form
277 void reorganize(char *inst)
278 {
279     int len1=strlen(inst);
280     char *tmp=(char*) malloc (len1*sizeof(char));
281     for(int i=0;i<len1;i++)
282         tmp[i]=inst[i];
283     clearstr(inst);
284
285     int len2=0;
286     int last_blank=1;
287     for(int i=0;i<len1;++i)
288     {
289         if (tmp[i]==' '||tmp[i]=='\n'||tmp[i]=='\t')
290         {   if(last_blank==0)
291             {
292                 inst[len2]=' ';
293                 len2++;
294                 last_blank=1;
295             }
296         }
297         else
298         {
299             inst[len2]=tmp[i];
300             len2++;
301             last_blank=0;
302         }
303     }
304     if(inst[len2-1]==' ') inst[len2-1]=0;
305     free(tmp);
306 }
307
308 //parse the instruction to args
309 int parse(char *inst,char **args)
310 {
311     int instlen=strlen(inst);
312     int argn =0;
313     for(int i=0;i<instlen;i++)
314     {
315         clearstr(args[argn]);
316         int j=i;
317         for(j=i;j<instlen && inst[j]!=' ';j++)
318         {args[argn][j-i]=inst[j];}
319         //special case for < > |
320         if(j>i+1 && (args[argn][0]=='<'||args[argn][0]=='>'||args[argn][0]=='|'))
321         {
322             strcpy(args[argn+1],args[argn]+1);
323             for(int k=1;k<j-i;++k) args[argn][k]=0;
324             argn++;
325         }
326
327         i=j;
328         argn++;
329     }
330         return argn;
331 }
```

```
332
333 void clearstr(char *str)
334 {
335     memset(str, 0, sizeof(str));
336 }
```

## 1.4 实验测试

- shell 测试 (图 1)

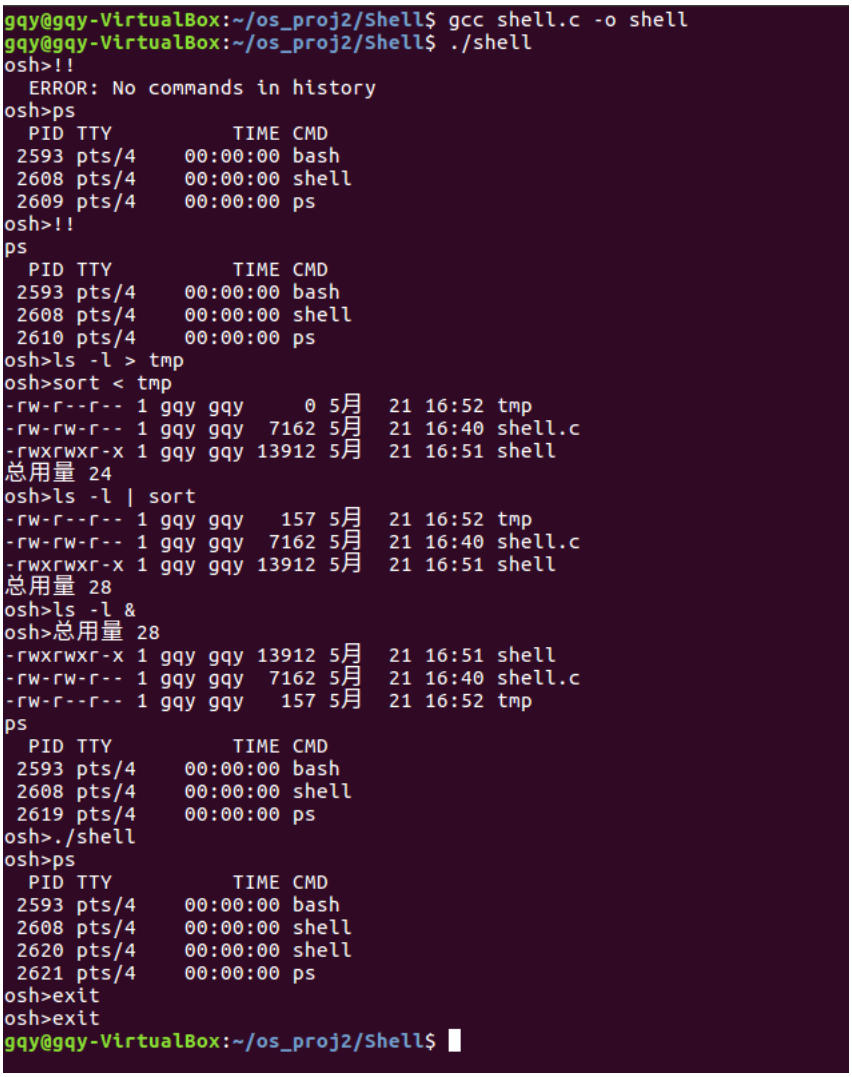


图 1: shell 测试

测试指令如下

```
1 gcc shell.c -o shell
2 ./shell
3 !!
4 ps
5 !!
```

```
ls -l > tmp
sort < tmp
ls -l | sort
ls -l &
ps
./shell
ps
exit
exit
```

首先编译 shell.c 文件并执行，输入!! 指令由于历史指令为空报错，之后正常执行 ps 指令，再输入!! 指令就输出了历史指令 ps 并执行。然后 ls -l > tmp 和 sort < tmp 测试输入输出重定位，ls -l | sort 测试 pipe 通信，ls -l & 测试并行。再次执行 shell，然后输入 ps 可以看到当前系统任务列表有两层 shell 在执行，输入两次 exit 可以正常退出程序。

# 2 Linux Kernel Module for Task Information

## 2.1 实验内容与目标

- 设计/proc 文件系统内核模块
- 支持通过进程 pid 查看任务信息

## 2.2 实验过程及步骤

- 读取用户输入的 pid
  首先模仿 project1 中的文件系统内核模块创建新的内核模块 pid，根据书本指导，在 file_operations 中增加.write = proc_write 语句，根据书本 Figure 3.37 创建 proc_write() 函数。用 kmalloc 给 k_mem 分配空间，用来存储 user buffer 的信息，在 copy_from_user 函数后增加 k_mem[count] = 0 语句人为添加字符串结尾字符，否则会出现 pid 不合法的情况。使用 kstrtol 函数将 k_mem 的信息传入 pid，然后使用 kfree 释放空间。
- 根据 pid 列出任务信息
  在 proc_read() 函数中创建 task_struct 类型的 PCB，用 pid_task(find_vpid(pid), PIDTYPE_PID) 来获取相应 pid 的 PCB 信息，并按格式输出任务信息然后 copy_to_user。
- 应对不合法 pid 指令
  如果 pid_task 函数返回 NULL，说明 pid 不合法，输出相应报错后退出。

## 2.3 实验代码

pid.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 <linux/uaccess.h>
# include <asm/uaccess.h>

# define BUFFER_SIZE 128
# define PROC_NAME "pid"

static long pid;

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,
    .write = proc_write,
};

static int proc_init(void) {

    proc_create(PROC_NAME, 0666, NULL, &proc_ops);
    printk(KERN_INFO "/proc/" PROC_NAME " is loaded!\n");
    return 0;
}

static void proc_exit(void) {

    remove_proc_entry(PROC_NAME, NULL);
    printk(KERN_INFO "/proc/" PROC_NAME " is removed!\n");
}

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 *PCB = NULL;

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

    PCB = pid_task(find_vpid(pid), PIDTYPE_PID);
    if (PCB == NULL) {
        printk(KERN_INFO "Invalid PID!\n");
        return 0;
    }

    completed = 1;

    rv = sprintf(buffer, "command = [%s] pid = [%ld] state = [%ld]\n", PCB -> comm, pid, PCB -> state);

    copy_to_user(usr_buf, buffer, rv);

    return rv;
}
```

```
63
64 static ssize_t proc_write(struct file *file, const char __user *usr_buf, size_t count, loff_t *pos)
65 {
66     char *k_mem;
67
68     // allocate kernel memory
69     k_mem = kmalloc(count, GFP_KERNEL);
70
71     // copies user space usr_buf to kernel buffer
72     copy_from_user(k_mem, usr_buf, count);
73     k_mem[count] = 0;
74
75     kstrtol(k_mem, 10, &pid);
76
77     // free the memory
78     kfree(k_mem);
79
80     return count;
81 }
82
83 module_init(proc_init);
84 module_exit(proc_exit);
85
86 MODULE_LICENSE("GPL");
87 MODULE_DESCRIPTION("Pid Module");
88 MODULE_AUTHOR("GQY");
```

## 2.4 实验测试

- pid 测试 (图 2)
  测试指令如下

```
1  make
2  sudo dmesg -C
3  sudo insmod pid.ko
4  sudo dmesg
5  echo "1" > /proc/pid
6  cat /proc/pid
7  echo "1395" > /proc/pid
8  cat /proc/pid
9  echo "9" > /proc/pid
10 cat /proc/pid
11 echo "132" > /proc/pid
12 cat /proc/pid
13 sudo dmesg
14 sudo rmmod pid
15 sudo dmesg
```

首先用 Makefile 文件编译内核模块，然后清空内核日志缓冲区并加载 pid 模块，输入 dmesg 可以看到内核模块成功加载。然后先后输入 pid 为 1，1395，9 查询，都可以通过 cat /proc/pid 正常看到任务信息，最后输入 pid 为 132 查询发现无法通过 cat /proc/pid 查询到任务信息，输入 dmesg 查看发现是因为 pid 不合法报错。最后卸载 pid 模块，输入 dmesg 可以看到内核模块成功卸载。

```
gqy@gqy-VirtualBox:~/os_proj2/Pid$ make
make -C /lib/modules/4.15.0-112-generic/build M=/home/gqy/os_proj2/Pid modules
make[1]: Entering directory '/usr/src/linux-headers-4.15.0-112-generic'
  CC [M]  /home/gqy/os_proj2/Pid/pid.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/gqy/os_proj2/Pid/pid.mod.o
  LD [M]  /home/gqy/os_proj2/Pid/pid.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.15.0-112-generic'
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo dmesg -C
[sudo] gqy 的密码:
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo insmod pid.ko
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo dmesg
[ 1620.966927] /proc/pid is loaded!
gqy@gqy-VirtualBox:~/os_proj2/Pid$ echo "1" > /proc/pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ cat /proc/pid
command = [systemd] pid = [1] state = [1]
gqy@gqy-VirtualBox:~/os_proj2/Pid$ echo "1395" > /proc/pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ cat /proc/pid
command = [gmain] pid = [1395] state = [1]
gqy@gqy-VirtualBox:~/os_proj2/Pid$ echo "9" > /proc/pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ cat /proc/pid
command = [rcu_bh] pid = [9] state = [1026]
gqy@gqy-VirtualBox:~/os_proj2/Pid$ echo "132" > /proc/pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ cat /proc/pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo dmesg
[ 1620.966927] /proc/pid is loaded!
[ 1680.292232] Invalid PID!
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo rmmod pid
gqy@gqy-VirtualBox:~/os_proj2/Pid$ sudo dmesg
[ 1620.966927] /proc/pid is loaded!
[ 1680.292232] Invalid PID!
[ 1695.603356] /proc/pid is removed!
gqy@gqy-VirtualBox:~/os_proj2/Pid$
```

图 2: pid 测试

# 3 Conclusion

## 3.1 问题与解决方案

本次 project2 的 UNIX shell 部分稍有一点难度。首先分析指令部分需要对字符串操作比较熟悉；在设计支持重定位和 pipe 通信的时候遇到了许多新的函数，最开始并没有什么头绪，但是在仔细研读书本示例并经过多次尝试后最终成功完成。

project2 的另一个部分 pid 内核模块的设计是 project1 延伸，书上的指导也比较细致，就是在设计 proc_write() 函数的时候，如果仅仅根据书本上的操作会发现最后 pid 读取都非法，后来发现需要人为添加字符串结尾字符能避免该问题。

## 3.2 实验心得

本次 project2 是对所学知识一次很好的运用，比如设计 UNIX shell 需要有一个全局观念，要综合运用变量设计，空间分配，进程管理等方式，同时在实践的过程中也让我对 pipe 通信有了更深入的了解。虽然过程中也遇到了一些问题，但在耐心检查反复尝试的过程中都顺利解决了，总的来说本次 project 很好地锻炼了动手能力并加深了对理论知识的理解，让我受益匪浅。