Project 2

秦啸涵　**521021910604**

Part 1: 实现shell接口osh>

1-1: 创建子进程并在子进程中执行命令

```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$ gcc simple-shell.c -o shell
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$ ./shell
osh>ls -l
total 32
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke     0 Apr 18 02:00 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
osh>ls -l &
osh>total 32
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke     0 Apr 18 02:00 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
ls
osh>in.txt      out.txt  shell  simple-shell.c

osh>exit
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$
```

实现了`parse(char* input, char* args[])`函数用于解析命令，父进程读入命令经过解析和判断后执行`fork()`，子进程根据命令的类别执行不同的操作，当解析到args中含有token `&`时，会将bool型变量concurrent设置为true，则父进程与子进程并行执行

```
1  else if(!concurrent)  //当concurrent为true时，父进程不会wait(NULL)而是并行执行
2      wait(NULL);
```

1-2: 提供历史记录功能

```
1  if (strcmp(input, "!!") == 0) {
2      if (have_history) {
3          printf("%s\n", last_input);
```

```
        strcpy(input, last_input);
    }
    else {
        printf("No commands in history.\n");
        continue;
    }
}
else{
    strcpy(last_input, input);
    have_history = true;
}
```

对传入的命令进行简单的判断并在满足条件时更新历史记录

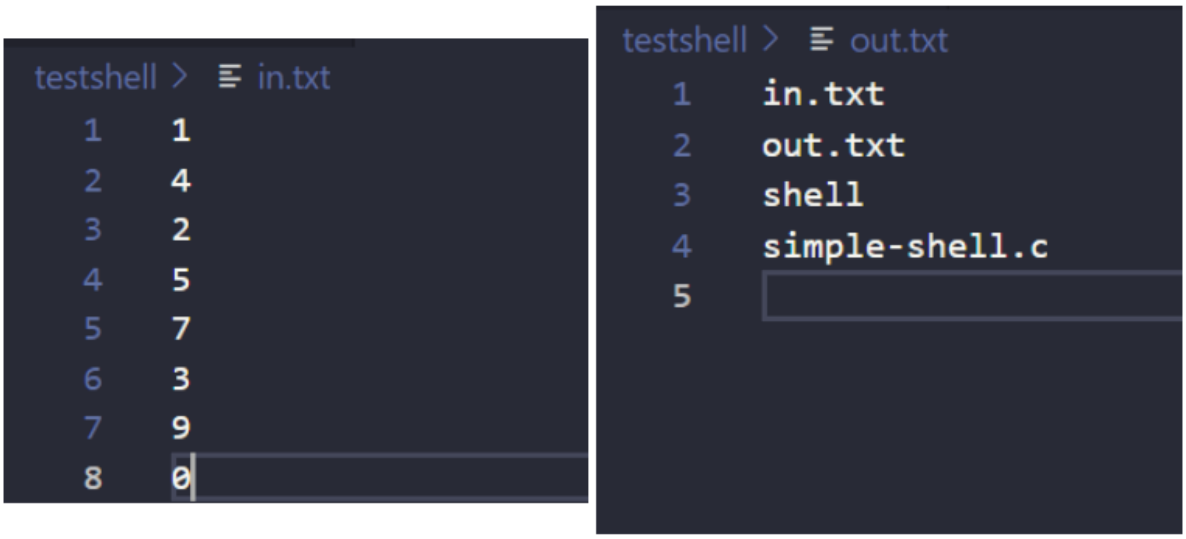
```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$ ./shell
osh>!!
No commands in history.
osh>ls -l
total 32
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke     0 Apr 18 02:00 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
osh>!!
ls -l
total 32
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke     0 Apr 18 02:00 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
osh>!!
ls -l
total 32
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke     0 Apr 18 02:00 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
osh>exit
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$
```

1-3: 提供输入输出重定向功能

```
int in_redirect = 0, out_redirect = 0;
if (cnt >= 3 && (strcmp(args[cnt - 2], ">") == 0 ||
strcmp(args[cnt - 2], "<") == 0))
```

```
{
    if (strcmp(args[cnt - 2], ">") == 0)
    {
        out_redirect = 1;
        strcpy(out_file, args[cnt - 1]);
    }
    else
    {
        in_redirect = 1;
        strcpy(in_file, args[cnt - 1]);
    }
    args[cnt - 2] = NULL;
    args[cnt - 1] = NULL;
    free(args[cnt - 2]);
    free(args[cnt - 1]);
    cnt -= 2;
}
if (in_redirect) {
    int fd = open(in_file, O_RDONLY);
    if (fd < 0) {
        printf("Error: Cannot open file %s.\n", in_file);
        error_flag = 1;
    }
    if (error_flag) continue;
    dup2(fd, STDIN_FILENO);
    close(fd);
}
if (out_redirect) {
    int fd = open(out_file, O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (fd < 0) {
        printf("Error: Cannot open file %s.\n", out_file);
        error_flag = 1;
    }
    if (error_flag) continue;
    dup2(fd, STDOUT_FILENO);
    execvp(args[0], args);
    close(fd);
}
execvp(args[0], args);
```

通过 `dup2()` 函数管理输入输出的重定向，并在产生错误时及时continue终止操作。(`in_file` 和 `out_file` 的free在放在最后实现)




```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$ ./shell
osh>ls > out.txt
osh>sort < in.txt
0
1
2
3
4
5
7
9
osh>exit
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$
```

1-4: 允许父子进程通过管道进行通信

```c
int pipe_pos = -1;
for (int i = 0; i < cnt; ++ i)
    if (strcmp(args[i], "|") == 0) {
        pipe_pos = i;
        break;
    }
if (pipe_pos >=0)
{
    if (pipe_pos == 0 || pipe_pos == cnt - 1)
    {
        printf("Error: Unexpected syntax '|'.\n");
```

```
            error_flag = 1;
        }
        if (error_flag) continue;
        int fd[2];
        pipe(fd);
        pid_t pid1 = fork();
        if (pid1 < 0) printf("Error: Fork failed!\n");
        else if (pid1 == 0)
        {
            for (int i=pipe_pos; i<cnt; i++)
                args[i] = NULL;
            close(fd[0]);
            dup2(fd[1], STDOUT_FILENO);
            execvp(args[0], args);
            close(fd[1]);
        }
        else
        {
            wait(NULL);
            for (int i = pipe_pos + 1; i < cnt; ++ i) args[i - pipe_pos - 1] = args[i];
            for (int i = cnt - pipe_pos - 1; i < cnt; ++ i) args[i] = NULL;
            close(fd[1]);
            dup2(fd[0], STDIN_FILENO);
            close(fd[0]);
            execvp(args[0], args);
        }
    }
```

对于pipe操作，以 `ls -l | less` 为例，使子进程再次创建一个子进程来执行 `ls -l` 操作，而它自己将通过pipe获得 `ls -l` 的输出并执行 `less` （所有的free操作依然在最后执行），测试如下：

```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/testshell$ ./shell
osh>ls -l | sort
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke    36 Apr 18 02:19 out.txt
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
total 36
osh>ls -l | less
osh>
```

```
total 36
-rw-rw-r-- 1 jianke jianke    15 Apr 18 02:00 in.txt
-rw-rw-r-- 1 jianke jianke    36 Apr 18 02:19 out.txt
-rwxrwxr-x 1 jianke jianke 17600 Apr 18 01:59 shell
-rwxrw-rw- 1 jianke jianke  5191 Apr 18 01:59 simple-shell.c
~
~
~
~
~
~
~
~
~
~
~
(END)
```

Part 2: 设计pid内核模块读取进程信息

需要对 `proc_read()` 函数和 `proc_write()` 函数进行修改

proc_write()函数

在该函数中我们需要将用户输入的进程标识符pid写入/proc/pid文件，我使用了 `sscanf()` 函数来直接读入(使用 `kstrol()` 函数需要满足读入字符串末尾为 `Null`)

```
1   sscanf(k_mem, "%ld", &l_pid);
```

proc_read()函数

当用户使用cat命令读取/proc/pid文件时，将调用proc_read()函数输出进程相关信息，增加的代码如下：

```
1  if (tsk == NULL)
2  {
3      printk(KERN_INFO "No such pid\n");
4      return 0;
5  }
6  completed = 1;  \\
7  rv = sprintf(buffer, "pid: [%d], name: [%s], state: [%ld], prio:
   [%d], vruntime: [%llu]\n",
8  tsk->pid, tsk->comm, tsk->__state, tsk->prio, tsk->se.vruntime);
```

具体而言，当输入的pid没有对应的进程时，函数直接返回0，此时`dmesg`在内核缓冲区可以看到"No such pid"的输出

否则将`completed`设为1(避免重复调用)，之后将进程相关信息如pid、命令、当前状态等打印到buffer中并在之后调用`copy_to_user()`输出。具体运行结果如下：

```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ sudo dmesg -C
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ sudo insmod pid.ko
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ dmesg
[  827.988863] /proc/pid created
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ echo "1" > /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ cat /proc/pid
pid: [1], name: [systemd], state: [1], prio: [120], vruntime: [811797682]
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ echo "1395" > /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ cat /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ dmesg
[  827.988863] /proc/pid created
[  863.738013] No such pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ echo "1657" > /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ cat /proc/pid
pid: [1657], name: [ibus-daemon], state: [1], prio: [120], vruntime: [1501095302]
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ echo "1392" > /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ cat /proc/pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ dmesg
[  827.988863] /proc/pid created
[  863.738013] No such pid
[  895.866086] No such pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$ sudo rmmod pid
jianke@ubuntu:~/Desktop/final-src-osc10e/ch3/pid$
```

Bonus:匿名管道与命名管道的差异

匿名管道和命名管道都是进程间通信的机制，但它们有一些重要的区别。

1. 命名管道可以在文件系统中创建一个管道文件，而匿名管道没有对应的文件系统节点。
2. 命名管道可以被多个进程打开和使用，而匿名管道只能在父子进程间使用。
3. 命名管道可以持久存在，即使创建它的进程终止，也能被其他进程打开和使用。而匿名管道只在创建它的进程存在时有效，当该进程终止后，管道也被销毁。
4. 在使用命名管道时，进程可以通过文件描述符来引用它，因此可以使用类似于文件I/O的操作进行读写。而匿名管道只能使用一组特殊的系统调用（如 `pipe()` 、 `read()` 、 `write()` 等）进行读写操作。

总之，命名管道比匿名管道更为灵活和通用，但它们也有一些额外的开销，如文件系统开销和文件描述符的使用等。匿名管道则更加轻量级，适用于只需要父子进程之间通信的场景。