OS-Project2

一、Unix Shell

1、实验目标：

修改 simple-shell.c ，编译运行实现以下功能：（1）创建子进程并在子进程中执行命令 （2）历史记录功能 （3）输入输出重定向功能 （4）pipe 通信

2、代码设计：

（1） 创建子进程并执行命令：

首先设计以下初始化函数，分别初始化输入的参数列表和历史记录。

char** init_args (char** args, char* inst, char* last) —将命令解析为token

char **init_buff (char** args, char** buffer, char* last)—保存输入命令历史

之后在 main 函数中初始化各个变量（用 c 语言的 malloc & free 来申请动态空间）。在 main 函数的 注释part 1中首先在终端输出 “osh>" ，之后将输入指令保存在 args 中，历史保存在 buff 之中。

完成解析/保存历史记录后，在 main 函数的 注释4.2 和 4.3 部分进行相关代码的执行。此部分内容包括进程创建/调用等。根据其他信息，在子进程中按要求调用 execvp（）函数来执行保存在 args 中的命令。

（2） 历史记录：

在main函数 part 2 中，完成第一步的初始化内容后，检查参数列表 args 。如果检查到 “!!” 则将检查 buff 。如果没有历史输入，给出报错信息，否则将历史输入加载到 args 中。当main函数向后运行时，该条历史记录将按照相应信息进行子进程的调用。

（3） 输入输出重定向：

首先检查是否有 pipe 通信，若没有，则在 4.3 查询 args 中是否有 “<" 或者 “>”，这分别是输入与输出的重定向符号。若检测到对应符号，首先将符号后面的文件名保存，清空 args 中的该符号以及文件名，并进行重定向，将文件中的内容作为输入的命令（4.3.1）或输出的内容（4.3.2）。注意以上部分在子进程中执行。

此时父进程要检测是否在命令结尾输入了”&”，若有，则要并行，否则需要等待。

（4） pipe 通信：

在 args 中检测到输入 “|" 后保存该符号的位置，并在该位置创建管道 4.2。该位置之前的指令由子进程执行，后面的指令由孙子进程执行，同时此处

要进行重定向，将子进程的输出重定向作为孙子进程的输入，在对应进程中调用 execvp（）函数执行命令。

3、结果验证：

以下展示部分命令运行结果（左上至右下：ls -l;less;ls -l |sort;sort < in.txt）

```
osh>ls -l | less
osh>
```

```
-rw-rw-r-- 1 parallels parallels   1413 Jan  4  2018 newproc-win32.c
-rwx------ 1 parallels parallels    408 May 10 20:55 out.txt
-rw-r--r-- 1 parallels parallels   3089 May  9 13:20 pid.c
-rw-rw-r-- 1 parallels parallels 315056 May  9 13:03 pid.ko
-rw-rw-r-- 1 parallels parallels     44 May  9 13:03 pid.mod
-rw-rw-r-- 1 parallels parallels   1461 May  9 13:03 pid.mod.c
:
```

```
-rw-rw-r-- 1 parallels parallels 315056 May  9 13:03 pid.ko
-rwx------ 1 parallels parallels    408 May 10 20:55 out.txt
-rwxrwxr-x 1 parallels parallels   8712 Jan 31  2018 multi-fork
-rwxrwxr-x 1 parallels parallels  14064 May 10 20:51 simple-shell
-rwxrwxr-x 1 parallels parallels  14096 May 10 20:45 a.out
total 792
```

```
osh>sort < in.txt
1
2
3
5
7
```

二、Linux Kernel Module for Task Information

1、实验目标：

学习 /proc 文件系统的读写，展示进程标识符信息。

2、代码设计：

（1）修改结构体：

```
static struct proc_ops my_fops = {
    .proc_read = proc_read,
    .proc_write = proc_write};
```

（2）修改 proc_read 函数：

```
tsk = pid_task(find_vpid(l_pid), PIDTYPE_PID);
if (tsk == NULL)
        return -1;
rv = sprintf(buffer, "command = [%s], pid = [%ld], state = [%ld]\n", tsk->comm, l_pid, tsk->__state);

completed = 1;
```

（3）修改 proc_write 函数：

```
char buffer[BUFFER_SIZE];
sscanf(k_mem, "%s", buffer);
kstrtol(buffer, 10, &l_pid);
kfree(k_mem);

return count;
```

3、结果验证：

```
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ sudo insmod pid.ko
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ ps
    PID TTY          TIME CMD
 169061 pts/0    00:00:00 bash
 171868 pts/0    00:00:00 ps
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ echo "169061" > /pr
oc/pid
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ cat /proc/pid
command = [bash], pid = [169061], state = [1]
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ sudo rmmod pid
parallels@ubuntu-linux-22-04-desktop:~/final-src-osc10e/ch3$ sudo dmesg -c
[51007.606085] /proc/pid created
[51068.360454] /proc/pid removed
```

Bonus：

匿名管道通信只能用于父子进程或兄弟进程间通信，数据只能单向流动。命名管道通信可用于任意进程间通信，数据可双向流动，但需事先创建并指定一个独立的管道文件。