

# 操作系统课程设计报告

Unixshell历史特性

## 1 项目简介

### 1.1 项目目的

- 学习Linux系统进程创建和终止的机制。
- 学习UNIX进程的信号处理。

### 1.2 项目要求

实现简单的带有历史特性的Shell程序。

## 2 项目实施

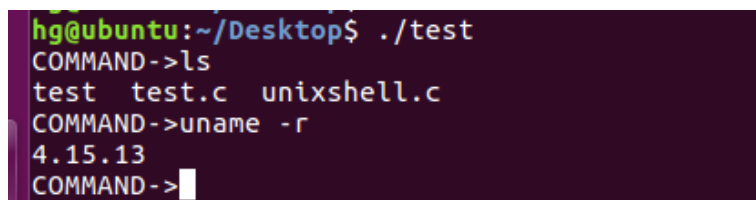
### 2.1 系统版本

- Ubuntu: 16.04
- Linux内核: 4.15.13

### 2.2 实现过程

#### 2.2.1 创建子进程并执行命令

修改书中提供的简单Shell框架，`setup()`将命令及参数装入`args`数组后，使用`fork()`函数创建子进程，在子进程中将`args`数组中的内容传递给`execvp()`函数以执行命令。



```
hg@ubuntu:~/Desktop$ ./test
COMMAND->ls
test test.c unixshell.c
COMMAND->uname -r
4.15.13
COMMAND->
```

图 1: 创建子进程并执行命令

### 2.2.2 父子进程并行执行

background初始值为0。setup()检查命令是否以“&”结尾，若是则将background设为1。在父进程中检查background值，若为1则不调用wait()，从而父子进程并行执行。

测试：输入“gedit”时，父进程等待子进程结束，因此只有关闭gedit才能输入并执行下一条命令。而输入“gedit &”时，父子进程并行执行，可直接输入执行下一条命令。

```
hg@ubuntu:~/Desktop$ ./test
COMMAND->pwd
/home/hg/Desktop
COMMAND->ls
test test.c unixshell.c
COMMAND->gedit
COMMAND->gedit &
COMMAND->
```

图 2: 父子进程并行执行

### 2.2.3 创建历史特性

1. 建立信号处理函数。历史缓存记录最近的10个命令。编写handle\_SIGINT()函数，使得当用户按快捷键Ctrl+C，信号处理器输出最近的10个命令。

```
hg@ubuntu:~/Desktop$ ./test
COMMAND->ls
test test.c unixshell.c
COMMAND->pwd
/home/hg/Desktop
COMMAND->uname -r
4.15.13
COMMAND->^C
Command History:
3. uname -r
2. pwd
1. ls
COMMAND->|
```

图 3: Ctrl+C输出最近命令

2. “r x”与“r”的实现。输入“r”运行最近的一个命令。输入“r x”运行最近一个以“x”开头的命令。并将它们指向的实际命令放入历史缓存。

```
hg@ubuntu:~/Desktop$ ./test
COMMAND->pwd
/home/hg/Desktop
COMMAND->ls
test test.c unixshell.c
COMMAND->cal
  April 2018
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

COMMAND->r
  April 2018
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30

COMMAND->r p
/home/hg/Desktop
COMMAND->^C
Command History:
5. pwd
4. cal
3. cal
2. ls
1. pwd

COMMAND->
```

图 4: “r x” 与 “r” 及其历史记录

3. “r” 运行错误命令的处理。当使用“r”或“r x”调用错误命令时，应不调用execvp()函数，该命令也不进入历史列表。

测试：输入“wrong”作为错误命令，返回“Error command. Execution fail.”，再使用“r”运行该错误命令，返回“Recall wrong command.”，查看历史列表可发现使用“r”对应的命令不在历史缓存中。

```
hg@ubuntu:~/Desktop$ ./test
COMMAND->ls
test test.c unixshell.c
COMMAND->pwd
/home/hg/Desktop
COMMAND->wrong
Error command. Execution fail.
COMMAND->r
Recall a wrong command.
COMMAND->^C
Command History:
3. wrong
2. pwd
1. ls

COMMAND->
```

图 5: “r” 运行错误命令。

## 2.3 项目中出现的问题及解决方案

### 1. wait()等待第一个终止的子进程

测试并行处理时发现，当并行运行一次后，其后的非并行命令也会被并行执行。

输入ps可发现其原因是：在一轮循环中，若并行运行，子进程结束而父进程没有wait，该子进程成为<defunct>僵尸进程，当父进程进入其后的循环时，由于wait() 等待第一个终止的子进程，其实是将最早的僵尸进程释放，而本轮循环中创建的子进程没有得到wait，从而并行执行，成为僵尸进程...

如下图所示，21738为第一个非并行的ps，它的父进程wait()释放了最早的僵尸进程21733，导致21738并行执行，并在本轮结束后成为僵尸进程：

```
ps &
COMMAND->  PID TTY          TIME CMD
21229 pts/0    00:00:00 bash
21731 pts/0    00:00:00 test
21733 pts/0    00:00:00 ps <defunct>
21734 pts/0    00:00:00 ps <defunct>
21735 pts/0    00:00:00 ps <defunct>
21736 pts/0    00:00:00 ps

ps &
COMMAND->  PID TTY          TIME CMD
21229 pts/0    00:00:00 bash
21731 pts/0    00:00:00 test
21733 pts/0    00:00:00 ps <defunct>
21734 pts/0    00:00:00 ps <defunct>
21735 pts/0    00:00:00 ps <defunct>
21736 pts/0    00:00:00 ps <defunct>
21737 pts/0    00:00:00 ps

ps
COMMAND->  PID TTY          TIME CMD
21229 pts/0    00:00:00 bash
21731 pts/0    00:00:00 test
21734 pts/0    00:00:00 ps <defunct>
21735 pts/0    00:00:00 ps <defunct>
21736 pts/0    00:00:00 ps <defunct>
21737 pts/0    00:00:00 ps <defunct>
21738 pts/0    00:00:00 ps
21739 pts/0    00:00:00 ps

ps
COMMAND->  PID TTY          TIME CMD
21229 pts/0    00:00:00 bash
21731 pts/0    00:00:00 test
21736 pts/0    00:00:00 ps <defunct>
21737 pts/0    00:00:00 ps <defunct>
21738 pts/0    00:00:00 ps <defunct>
```

图 6: 僵尸进程

解决方案为使用waitpid()指定pid，使父进程wait释放本轮创建的子进程，而非之前的僵尸进程。

如下图非并行的21807，21808被父进程正确的wait()终止而非成为僵尸进程：

```
ps &
COMMAND-> PID TTY          TIME CMD
21229 pts/0    00:00:00 bash
21797 pts/0    00:00:00 test
21798 pts/0    00:00:00 ps <defunct>
21801 pts/0    00:00:00 ps <defunct>
21802 pts/0    00:00:00 ps <defunct>
21805 pts/0    00:00:00 ps <defunct>
21806 pts/0    00:00:00 ps
ps
  PID TTY          TIME CMD
 21229 pts/0    00:00:00 bash
 21797 pts/0    00:00:00 test
 21798 pts/0    00:00:00 ps <defunct>
 21801 pts/0    00:00:00 ps <defunct>
 21802 pts/0    00:00:00 ps <defunct>
 21805 pts/0    00:00:00 ps <defunct>
 21806 pts/0    00:00:00 ps <defunct>
 21807 pts/0    00:00:00 ps
COMMAND->ps
  PID TTY          TIME CMD
 21229 pts/0    00:00:00 bash
 21797 pts/0    00:00:00 test
 21798 pts/0    00:00:00 ps <defunct>
 21801 pts/0    00:00:00 ps <defunct>
 21802 pts/0    00:00:00 ps <defunct>
 21805 pts/0    00:00:00 ps <defunct>
 21806 pts/0    00:00:00 ps <defunct>
 21808 pts/0    00:00:00 ps
COMMAND->ps
  PID TTY          TIME CMD
 21229 pts/0    00:00:00 bash
 21797 pts/0    00:00:00 test
 21798 pts/0    00:00:00 ps <defunct>
 21801 pts/0    00:00:00 ps <defunct>
 21802 pts/0    00:00:00 ps <defunct>
 21805 pts/0    00:00:00 ps <defunct>
 21806 pts/0    00:00:00 ps <defunct>
 21809 pts/0    00:00:00 ps
```

图 7: 正确处理

进一步的改进为将并行遗留的僵尸进程pid值记录下来，并在一定时间后释放，从而避免资源的占用。

## 2. 父子进程间通信

当运次错误命令时，子进程中execvp()返回-1，需要记录错误命令从而在使用“r”或“r

x”再次调用时，不调用execvp()且不记入历史缓冲。问题在于：子进程得到的是父进程的数据副本，因此子进程中改变数据，在父进程中相应数据是不变的。因此为了让父进程知道子进程中运行命令错误这一信息，可以用pipe实现父子进程间的通信。

在使用pipe读写时还有一个小问题：读写都是默认阻塞的，即在缓冲空而需要读(缓冲满而需要写)的时候，将会阻塞在该语句不继续执行，直到缓冲非空(非满)，可使用如下语句设置读者非阻塞：

```
fcntl(*read_fd, F_SETFL, O_NONBLOCK);
```

### 3 总结及反思

通过这次项目我了解了Linux系统进程创建、终止、通信的机制。