

# 哈尔滨工业大学

# 实验报告

## 实验（六）

题 目 TinyShell

微壳

专 业 计算机

学 号 1170300825

班 级 1703008

学 生 李大鑫

指 导 教 师 郑贵滨

实 验 地 点 \_\_\_\_\_

实 验 日 期 \_\_\_\_\_

计算机科学与技术学院

# 目 录

|                                       |               |
|---------------------------------------|---------------|
| <b>第 1 章 实验基本信息 .....</b>             | <b>- 3 -</b>  |
| 1.1 实验目的.....                         | - 3 -         |
| 1.2 实验环境与工具.....                      | - 3 -         |
| 1.2.1 硬件环境.....                       | - 3 -         |
| 1.2.2 软件环境.....                       | - 3 -         |
| 1.2.3 开发工具.....                       | - 3 -         |
| 1.3 实验预习.....                         | - 3 -         |
| <b>第 2 章 实验预习 .....</b>               | <b>- 5 -</b>  |
| 2.1 进程的概念、创建和回收方法（5 分） .....          | - 5 -         |
| 2.2 信号的机制、种类（5 分） .....               | - 5 -         |
| 2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分） ..... | - 6 -         |
| 2.4 什么是 SHELL，功能和处理流程（5 分） .....      | - 7 -         |
| <b>第 3 章 TINY SHELL 测试.....</b>       | <b>- 9 -</b>  |
| 3.1 TINY SHELL 设计 .....               | - 9 -         |
| <b>第 4 章 总结 .....</b>                 | <b>- 19 -</b> |
| 4.1 请总结本次实验的收获.....                   | - 19 -        |
| 4.2 请给出对本次实验内容的建议.....                | - 19 -        |
| <b>参考文献.....</b>                      | <b>- 21 -</b> |

## 第 1 章 实验基本信息

### 1.1 实验目的

- 理解现代计算机系统进程与并发的基本知识
- 掌握 linux 异常控制流和信号机制的基本原理和相关系统函数
- 掌握 shell 的基本原理和实现方法
- 深入理解 Linux 信号响应可能导致的并发冲突及解决方法
- 培养 Linux 下的软件系统开发与测试能力

### 1.2 实验环境与工具

#### 1.2.1 硬件环境

Windows7 64 位以上； VirtualBox/Vmware 11 以上； Ubuntu 16.04 LTS 64 位/  
优麒麟 64 位

#### 1.2.2 软件环境

Windows7 64 位以上； VirtualBox/Vmware 11 以上； 1.2.3 开发工具

### 1.3 实验预习

- 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。
- 了解进程、作业、信号的基本概念和原理
- 了解 shell 的基本原理

■ 熟知进程创建、回收的方法和相关系统函数

熟知信号机制和信号处理相关的系统函数

## 第 2 章 实验预习

总分 20 分

### 2.1 进程的概念、创建和回收方法（5 分）

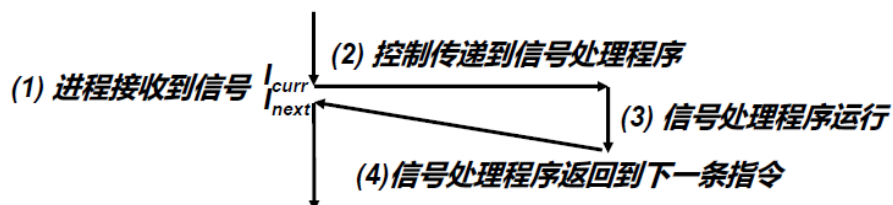
进程概念：一个执行中程序的实例。进程所提供的：我们的程序好像是系统中当前运行的唯一程序一样，我们的程序好像是独占的使用处理器和内存，处理器好像是无间断的执行我们程序中的指令，我们程序中的代码和数据好像是系统内存中唯一的对象。

创建进程：fork 函数：父进程通过调用 fork 函数创建一个新的运行的子进程。子进程得到与父进程用户级虚拟地址空间相同的一份副本。

回收进程：1) 当一个进程由于某种原因终止时，进程保持在一种已终止的状态中，直到被它的父进程回收，当父进程回收已终止的子进程时，内核将子进程的退出状态传递给父进程，然后抛弃已终止的进程，从此是开始，该进程就不存在了。2) 如果一个父进程终止了，内核会安排 init 进程成为它的孤儿进程的养父。

### 2.2 信号的机制、种类（5 分）

信号的机制：信号就是一条小消息，他通知进程系统中发生了一个某种类型的事件，类似于异常，从内核发送到（有时是在另一个进程的请求下）一个进程，信号类型是用小整数 ID 来标识的，信号中唯一的信息是它的 ID 和它的到达。发送信号：内核通过更新目的进程上下文中的某个状态，发送一个信号给目的进程。接收信号：当目的进程被内核强迫以某种方式对信号的发送作出反应时，他就接收了信号。反应方式：



信号的种类：

| 序号 | 名称        | 默认行为                         | 相应事件              |
|----|-----------|------------------------------|-------------------|
| 1  | SIGHUP    | 终止                           | 终端线挂断             |
| 2  | SIGINT    | 终止                           | 来自键盘的中断           |
| 3  | SIGQUIT   | 终止                           | 来自键盘的退出           |
| 4  | SIGILL    | 终止                           | 非法指令              |
| 5  | SIGTRAP   | 终止并转储内存 <sup>①</sup>         | 跟踪陷阱              |
| 6  | SIGABRT   | 终止并转储内存 <sup>①</sup>         | 来自 abort 函数的终止信号  |
| 7  | SIGBUS    | 终止                           | 总线错误              |
| 8  | SIGFPE    | 终止并转储内存 <sup>①</sup>         | 浮点异常              |
| 9  | SIGKILL   | 终止 <sup>②</sup>              | 杀死程序              |
| 10 | SIGUSR1   | 终止                           | 用户定义的信号 1         |
| 11 | SIGSEGV   | 终止并转储内存 <sup>①</sup>         | 无效的内存引用 (段故障)     |
| 12 | SIGUSR2   | 终止                           | 用户定义的信号 2         |
| 13 | SIGPIPE   | 终止                           | 向一个没有读用户的管道做写操作   |
| 14 | SIGALRM   | 终止                           | 来自 alarm 函数的定时器信号 |
| 15 | SIGTERM   | 终止                           | 软件终止信号            |
| 16 | SIGSTKFLT | 终止                           | 协处理器上的栈故障         |
| 17 | SIGCHLD   | 忽略                           | 一个子进程停止或者终止       |
| 18 | SIGCONT   | 忽略                           | 继续进程如果该进程停止       |
| 19 | SIGSTOP   | 停止直到下一个 SIGCONT <sup>②</sup> | 不是来自终端的停止信号       |
| 20 | SIGTSTP   | 停止直到下一个 SIGCONT              | 来自终端的停止信号         |
| 21 | SIGTTIN   | 停止直到下一个 SIGCONT              | 后台进程从终端读          |
| 22 | SIGTTOU   | 停止直到下一个 SIGCONT              | 后台进程向终端写          |
| 23 | SIGURG    | 忽略                           | 套接字上的紧急情况         |
| 24 | SIGXCPU   | 终止                           | CPU 时间限制超出        |
| 25 | SIGXFSZ   | 终止                           | 文件大小限制超出          |
| 26 | SIGVTALRM | 终止                           | 虚拟定时器期满           |
| 27 | SIGPROF   | 终止                           | 剖析定时器期满           |
| 28 | SIGWINCH  | 忽略                           | 窗口大小变化            |
| 29 | SIGIO     | 终止                           | 在某个描述符上可执行 I/O 操作 |
| 30 | SIGPWR    | 终止                           | 电源故障              |

## 2.3 信号的发送方法、阻塞方法、处理程序的设置方法 (5 分)

发送方法：1) 用/bin/kill 程序发送信号，/bin/kill 程序可以向另外的进程发送任意的信号。2) 从键盘发送信号，在键盘上输入 ctrl-c 会导致内核发送一个 SIGINT 信号到前台进程组中的每个进程。Ctrl-z 会发送一个 SIGSTP 信号到前台进程组中的每个进程。3) 用 kill 函数发送信号，进程通过调用 kill 函数发送信号给其他进程 (包括自己) 4) 用 alarm 发送信号，进程可以通过调用 alarm 函数在指定 secs 秒后发送一个 SIGALRM 信号给调用进程。

阻塞方法：隐式阻塞机制：内核默认阻塞任何当前处理程序正在处理信号类和待处理信号。显示阻塞进程：应用程序可以调用 `sigprocmask` 函数和它的辅助函数，明确地阻塞和解除阻塞选定的信号。调用要求如下：

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);

int sigismember(const sigset_t *set, int signum);

sigprocmask 函数改变当前阻塞的信号集合(8.5.1 节中描述的 blocked 位向量)。具体的行为依赖于 how 的值：
SIG_BLOCK：把 set 中的信号添加到 blocked 中(blocked=blocked | set)。
SIG_UNBLOCK：从 blocked 中删除 set 中的信号(blocked=blocked &~set)。
SIG_SETMASK：block=set。
```

返回：如果成功则为 0，若出错则为 -1。  
 返回：若 signum 是 set 的成员则为 1，如果不是则为 0，若出错则为 -1。

*保存 oldset*

处理程序的设置方法：

- 1) 调用 `signal` 函数，调用 `signal(SIG, handler)`，SIG 代表信号类型，handler 代表接收到 SIG 信号之后对应的处理程序。
- 2) 因为 `signal` 的语义各有不同，所以我们需要一个可移植的信号处理函数设置方法，Posix 标准定义了 `sigaction` 函数，它允许用户在设置信号处理时明确指定他们想要的信号处理语义，调用如下：

```
handler_t *Signal(int signum, handler_t *handler)
{
    struct sigaction action, old_action;

    action.sa_handler = handler;
    sigemptyset(&action.sa_mask); /* Block sigs of type being handled */
    action.sa_flags = SA_RESTART; /* Restart syscalls if possible */

    if (sigaction(signum, &action, &old_action) < 0)
        unix_error("Signal error");
    return (old_action.sa_handler);
}
```

## 2.4 什么是 shell，功能和处理流程（5 分）

什么是 shell: Shell 是一个用 C 语言编写的程序,他是用户使用 Linux 的桥梁。Shell 既是一种命令语言,又是一种程序设计语言,Shell 是指一种应用程序。

功能: Shell 应用程序提供了一个界面,用户通过这个界面访问操作系统内核的服务。

处理流程:

- 1) 从终端读入输入的命令。
- 2) 将输入字符串切分获得所有的参数
- 3) 如果是内置命令则立即执行
- 4) 否则调用相应的程序执行
- 5) shell 应该接受键盘输入信号,并对这些信号进行相应处理。



## 第 3 章 TinyShell 的设计与实现

总分 45 分

### 3.1 设计

#### 3.1.1 void eval(char \*cmdline) 函数 (10 分)

函数功能：解析从命令行输入的命令。

参 数：Char\* cmdline

处理流程：

- 1) 首先调用 `parseline` 函数。将 `cmdline` 字符串切分为参数数组 `argv`，另外 `parseline` 函数返回 `bg`，可以得知该操作是否需要后台运行。
- 2) 调用 `builtin_cmd` 判断该函数是否为内置函数。如果是内置函数则立即执行。
- 3) 如果不是内置函数，创建一个子进程，然后在子进程的上下文中运行命令行要求运行的函数（调用 `execve` 函数）。

要点分析：

- 1) 如何创建子进程并运行命令：首先将 `SIGCHLD`、`SIGINT`、`SIGSTP` 信号阻塞，因为在这些信号的信号处理函数中我们将使用到 `job` 相关的函数，这些函数会对全局变量工作列表 `jobs` 进行操作，而我们接下来在父进程中还要 `addjob` 将 `job` 加入的 `jobs` 中，所以为了避免子进程与信号处理函数竞争 `jobs`，我们需要使用 `sigprocmask` 阻塞以上信号。
- 2) 父进程创建完成子进程并用 `addjob` 记录后，需要用 `sigprocmask` 解除阻塞。
- 3) 子进程从父进程继承了信号阻塞向量，所以子进程必须确保在执行新程序之前接触对信号的阻塞。
- 4) `Ctrl-c` 会给所有前台进程组中的进程发送 `SIGINT` 信号，包括 `tsh` 和 `tsh` 创建的进程，但是这样使不正确的。所以我们需要在子进程中，`execve` 之前，调用函数 `setpgid(0,0)` 将子进程加入到一个 `pgid=pid` 的进程组之中。

#### 3.1.2 int builtin\_cmd(char \*\*argv) 函数 (5 分)

函数功能：判断当前函数是否是内置函数，如果是则立即执行

参 数：char \*\*argv

处理流程：

依次判断用户输入的命令名称 `argv` 是否是

- 1) 内置函数“quit”，如果是则调用 `exit(0)` 退出 `tsh`。
- 2) 内置函数“bg”|“fg”，如果是则调用函数 `do_fgbg`。返回值 1，`tsh` 继续。

- 3) 内置函数“jobs”，如果是则调用 listjobs 列出所有的 job 信息。返回值 1，tsh 继续。
- 4) 如果以上都不是，则不是内置函数，返回 0，tsh 会将它作为一个 job 处理。

要点分析：

1) 分别对三种不同的情况调用相应的处理函数。填对参数。了解三种内置函数的意义，注意只有 quit 是需要 exit(0)的，其他两种情况都需要返回 1，这是 tsh 会继续自己的处理流程。

### 3. 1.3 void do\_bgfg(char \*\*argv) 函数（5 分）

函数功能：执行内置命令 fg 和 bg

参 数：char \*\*argv

处理流程：

- 1) 首先对传入的命令行输入进行解析。首先 bg fg 的调用有两种格式，对这两种格式分别判断然后读入输入的 job pid，通过 pid 调用 getjobpid 获得 job。
- 2) 如果是 bg 命令，向 job 所在的进程组发送 SIGCONT 信号，更改 job 的 state 为 BG。
- 3) 如果是 fg 命令，向 job 所在的进程组发送 SIGCONT 信号，更改 job 的 state 为 FG，然后等待当前的程序运行直到当前的 job 不再是前台程序。

要点分析：

- 1) 对例外进行报错，分别有：命令为空，找不到 PID 的 job，不符合 bgfg 的格式，命令不是 bgfg。
- 2) 按照 bgfg 命令的特性，我们需要向目标 job 所在的进程组发送 SIGCONT 信号，代表如果该进程组中的进程停止，则需要重新进行。
- 3) 在 fg 命令中，我们是要将目标 job 放到前台运行，因此我们需要调用 waitfg 阻塞 tsh 进行，使前台一直都是该 job，直到该 job 不再是前台进程。触发 job 退出前台的条件是 job 结束运行，从而退出子进程使父进程调用 SIGCHLD 的处理函数，处理函数中对 job 进行了回收，此时前台程序不再是 job。

### 3. 1.4 void waitfg(pid\_t pid) 函数（5 分）

函数功能：阻塞直到指定 pid 的进程不再是前台进程。

参 数：pid\_t pid

处理流程:

进行 while 循环, 每次循环 sleep 1 秒, while 的终止条件是前台程序的 PID 不再是 pid。

要点分析:

调用 fgpip 函数向 jobs 查询当前 state 是 FG 的 job 的 PID。

### 3. 1.5 void sigchld\_handler(int sig) 函数 (10 分)

函数功能: 父进程中接收到 SIGCHLD 信号的处理函数。

参 数: int sig

处理流程:

- 1) 保存 errno
- 2) 处理所有子进程集合中已经停止或终止的子进程。
- 3) 如果该子进程通过调用 exit 或者一个返回正常终止, 则阻塞信号, 删除 job, 恢复信号。
- 4) 如果该子进程当前已经停止, 向屏幕打印信息。
- 5) 如果该当前子进程是因为一个未被捕获的信号终止的, 则向屏幕打印信息, 阻塞信号, 删除 job, 恢复信号。
- 6) 恢复 errno

要点分析:

- 1) 如何处理所有的子进程: 利用 while 循环重复判断, 判断条件是: while ((child\_pid = waitpid(-1, &status, WNOHANG|WUNTRACED)) > 0), 其中 waitpid 中的 option 设置为 WNOHANG|WUNTRACED 的含义是立即返回, 如果等待集合中的子进程都没有停止或终止的则返回为 0, 如果有一个, 则返回该进程的 PID。
- 2) 如何判断子进程不同的返回状态: 通过 waitpid 传入的 status, 分别调用 WIFEXITED, WIFSTOPPED, WIFSIGNALED, 如果为真, 则分别代表的返回情况是: 子进程通过调用 exit 或者一个返回正常终止、该子进程当前已经停止、该当前子进程是因为一个未被捕获的信号终止的。之所以判断是因为我们需要对这三种不同的情况进行不同的处理。
- 3) 因为在信号处理函数中可能会调用修改 errno 的函数, 所以我们需要保存恢复 errno。

### 3.2 程序实现 (tsh.c 的全部内容) (10 分)

重点检查代码风格:

- (1) 用较好的代码注释说明——5 分
- (2) 检查每个系统调用的返回值——5 分

## 第 4 章 TinyShell 测试

总分 15 分

### 4.1 测试方法

针对 tsh 和参考 shell 程序 tshref, 完成测试项目 4.1-4.15 的对比测试, 并将测试结果截图或者通过重定向保存到文本文件(例如: ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" > tshresult01.txt)。

### 4.2 测试结果评价

tsh 与 tshref 的输出在一下两个方面可以不同:


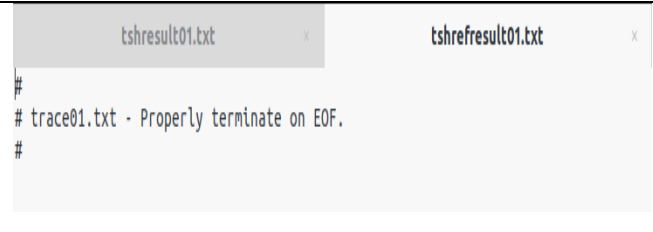
(1) PID

(2)测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令, 每次运行的输出都会不同, 但每个 mysplit 进程的运行状态应该相同。

除了上述两方面允许的差异, tsh 与 tshref 的输出相同则判为正确, 如不同则给出原因分析。

### 4.3 自测试结果

#### 4.3.1 测试用例 trace01.txt 的输出截图 (1 分)

| tsh 测试结果  |    | tshref 测试结果  |  |
|---|----|--|--|
|  |    |  |  |
| 测试结论  | 相同 |  |  |

#### 4.3.2 测试用例 trace02.txt 的输出截图 (1 分)

| tsh 测试结果 | tshref 测试结果 |
|----------|-------------|
|----------|-------------|

|   |    |
|---|----|
| <div> <div>tshresult02.txt</div> <div>tshresult02.txt</div> <pre># # trace02.txt - Process builtin quit command. #</pre> </div> <div> <div>tshresult02.txt</div> <div>tshresult02.txt</div> <pre># # trace02.txt - Process builtin quit command. #</pre> </div> |    |
| 测试结论  | 相同 |

#### 4.3.3 测试用例 trace03.txt 的输出截图（1 分）

| tsh 测试结果   | tshref 测试结果  |
|--|--|
| <div> <div>tshresult03.txt</div> <div>tshresult03.txt</div> <pre># # trace03.txt - Run a foreground job. # tsh&gt; quit</pre> </div> | <div> <div>tshresult03.txt</div> <div>tshresult03.txt</div> <pre># # trace03.txt - Run a foreground job. # tsh&gt; quit</pre> </div> |
| 测试结论   | 相同   |

#### 4.3.4 测试用例 trace04.txt 的输出截图（1 分）

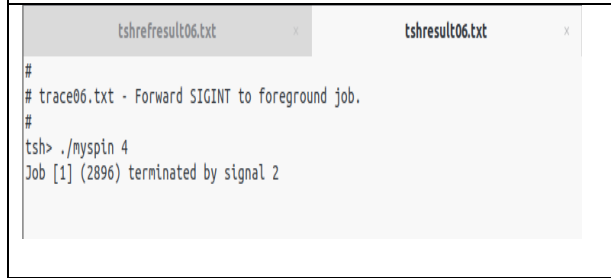
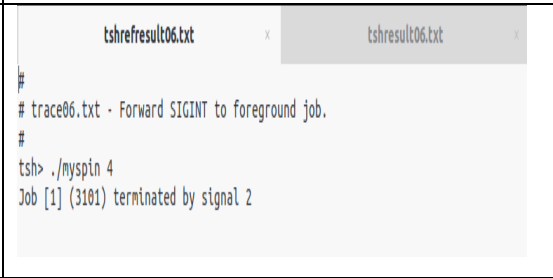
| tsh 测试结果   | tshref 测试结果  |
|--|--|
| <div> <div>tshresult04.txt</div> <div>tshresult04.txt</div> <pre># # trace04.txt - Run a background job. # tsh&gt; ./myspin 1 &amp; [1] (2875) ./myspin 1 &amp;</pre> </div> | <div> <div>tshresult04.txt</div> <div>tshresult04.txt</div> <pre># # trace04.txt - Run a background job. # tsh&gt; ./myspin 1 &amp; [1] (3089) ./myspin 1 &amp;</pre> </div> |
| 测试结论   | 相同   |

#### 4.3.5 测试用例 trace05.txt 的输出截图（1 分）

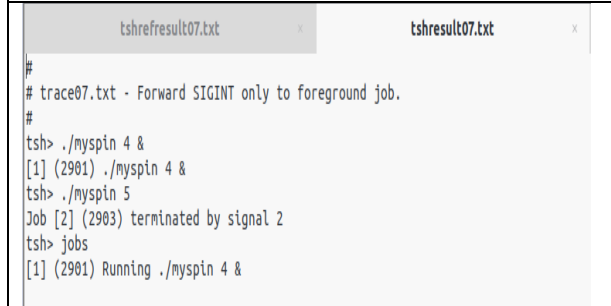
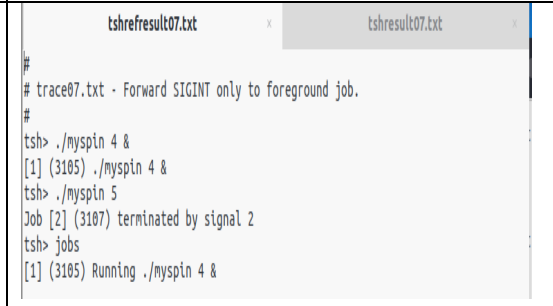
| tsh 测试结果 | tshref 测试结果 |
|----------|-------------|
|----------|-------------|

|   |  |
|---|--|
|  |  |
| 测试结论  | 相同   |

#### 4.3.6 测试用例 trace06.txt 的输出截图（1 分）

| tsh 测试结果   | tshref 测试结果   |
|--|---|
|  |  |
| 测试结论   | 相同  |

#### 4.3.7 测试用例 trace07.txt 的输出截图（1 分）

| tsh 测试结果  | tshref 测试结果  |
|---|--|
|  |  |
| 测试结论  | 相同   |

#### 4.3.8 测试用例 trace08.txt 的输出截图（1 分）

| tsh 测试结果 | tshref 测试结果 |
|----------|-------------|
|----------|-------------|

|   |   |
|---|---|
| <pre> tshresult08.txt # # trace08.txt - Forward SIGISIP only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (3654) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (3656) stopped by signal 20 tsh&gt; jobs [1] (3654) Running ./myspin 4 &amp; [2] (3656) Stopped ./myspin 5 </pre> | <pre> tshresult08.txt # # trace08.txt - Forward SIGISIP only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (3112) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (3114) stopped by signal 20 tsh&gt; jobs [1] (3112) Running ./myspin 4 &amp; [2] (3114) Stopped ./myspin 5 </pre> |
| 测试结论  | 相同  |

## 4.3.9 测试用例 trace09.txt 的输出截图（1 分）

|   |   |
|---|---|
| tsh 测试结果  | tshref 测试结果   |
| <pre> tshresult09.txt # # trace09.txt - Process bg builtin command # tsh&gt; ./myspin 4 &amp; [1] (3662) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (3664) stopped by signal 20 tsh&gt; jobs [1] (3662) Running ./myspin 4 &amp; [2] (3664) Stopped ./myspin 5 tsh&gt; bg %2 [2] (3664) ./myspin 5 tsh&gt; jobs [1] (3662) Running ./myspin 4 &amp; [2] (3664) Running ./myspin 5 </pre> | <pre> tshresult09.txt # # trace09.txt - Process bg builtin command # tsh&gt; ./myspin 4 &amp; [1] (3119) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (3121) stopped by signal 20 tsh&gt; jobs [1] (3119) Running ./myspin 4 &amp; [2] (3121) Stopped ./myspin 5 tsh&gt; bg %2 [2] (3121) ./myspin 5 tsh&gt; jobs [1] (3119) Running ./myspin 4 &amp; [2] (3121) Running ./myspin 5 </pre> |
| 测试结论  | 相同  |

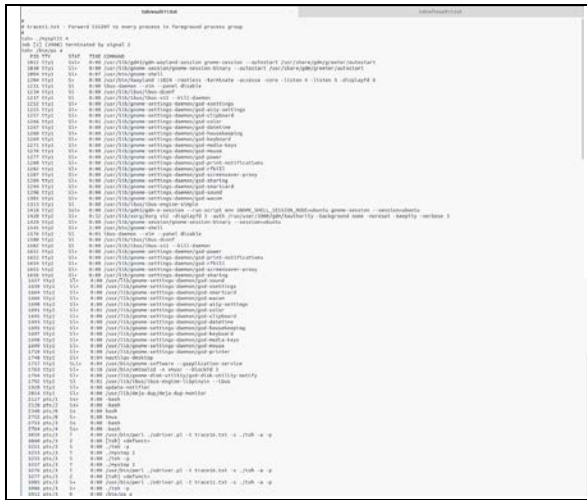
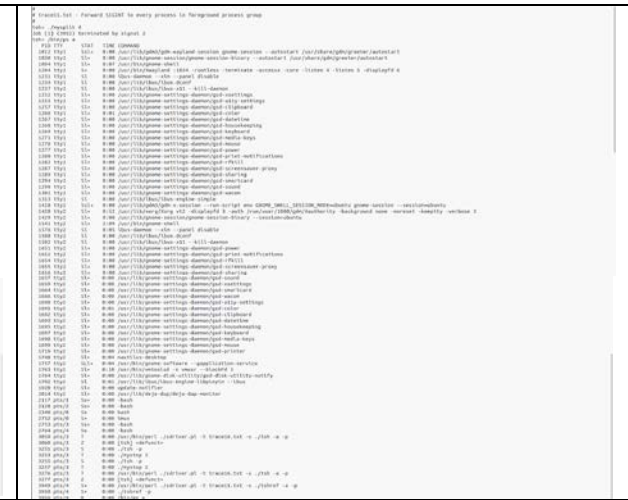
## 4.3.10 测试用例 trace10.txt 的输出截图（1 分）

|   |   |
|---|---|
| tsh 测试结果  | tshref 测试结果   |
| <pre> tshresult10.txt # # trace10.txt - Process fg builtin command. # tsh&gt; ./myspin 4 &amp; [1] (3671) ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (3671) stopped by signal 20 tsh&gt; jobs [1] (3671) Stopped ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; jobs </pre> | <pre> tshresult10.txt # # trace10.txt - Process fg builtin command. # tsh&gt; ./myspin 4 &amp; [1] (3129) ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (3129) stopped by signal 20 tsh&gt; jobs [1] (3129) Stopped ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; jobs </pre> |
| 测试结论  | 相同  |

## 4.3.11 测试用例 trace11.txt 的输出截图（1 分）

|          |             |
|----------|-------------|
| tsh 测试结果 | tshref 测试结果 |
|----------|-------------|



# 计算机系统实验报告

|   |  |
|---|--|
|  |  |
| 测试结论  | 相同   |

## 4.3.12 测试用例 trace12.txt 的输出截图（1 分）

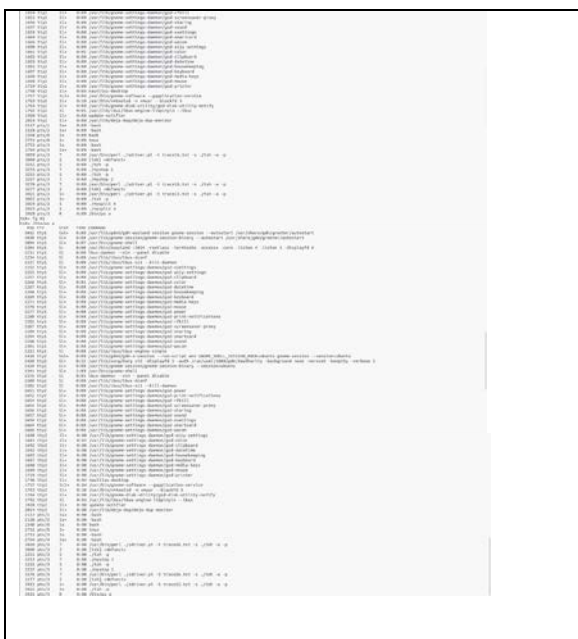

|  |   |
|--|---|
|  |  |
| 测试结论   | 相同  |

## 4.3.13 测试用例 trace13.txt 的输出截图（1 分）



|   |  |
|---|--|
|  |  |
|---|--|



# 计算机系统实验报告

|   |    |  |  |
|---|----|--|--|
|  |    |  |  |
| 测试结论  | 相同 |  |  |

## 4.3.14 测试用例 trace14.txt 的输出截图（1 分）

| tsh 测试结果  | tshref 测试结果  |
|---|--|
|  |  |
| 测试结论  | 相同   |

## 4.3.15 测试用例 trace15.txt 的输出截图（1 分）

| tsh 测试结果  | tshref 测试结果  |
|---|--|
|  |  |
| 测试结论  | 相同   |

#### 4.4 自测试评分

根据节 4.3 的自测试结果，程序的测试评分为：15。

## 第 4 章 总结

### 4.1 请总结本次实验的收获

- \* 了解了一个 shell 应该提供的大致操作
- \* 了解了实现一个 shell 需要的技术要点

### 4.2 请给出对本次实验内容的建议

注：本章为酌情加分项。



## 参考文献

**为完成本次实验你翻阅的书籍与网站等**

- [1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.
- [2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.
- [3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. <http://www.ie.nthu.edu.tw/info/ie.newie.htm>（Big5）.
- [4] 湛颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.
- [5] KANAMORI H. Shaking Without Quaking[J]. Science, 1998, 279 (5359): 2063-2064.
- [6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science , 1998 , 281 : 331-332[1998-09-23]. <http://www.sciencemag.org/cgi/collection/anatmorp>.