哈爾濱工業大學

实验报告

实 验（七）

题　　目 TinyShell

微壳

专　　业 计算学部

学　　号 1190200208

班　　级 1936602

学　　生 李旻翀

指 导 教 师 刘宏伟

实 验 地 点 G709

实 验 日 期 2012.6.3

计算机科学与技术学院

目 录

第 1 章 实验基本信息 - 4 -

1.1 实验目的 - 4 -
1.2 实验环境与工具 - 4 -
1.2.1 硬件环境 - 4 -
1.2.2 软件环境 - 4 -
1.2.3 开发工具 - 4 -
1.3 实验预习 - 4 -

第 2 章 实验预习 - 6 -

2.1 进程的概念、创建和回收方法（5 分） - 6 -
2.2 信号的机制、种类（5 分） - 6 -
2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分） - 7 -
2.4 什么是 SHELL，功能和处理流程（5 分） - 8 -

第 3 章 TINYSHELL 的设计与实现 - 10 -

3.1.1 VOID EVAL(CHAR *CMDLINE)函数（10 分） - 10 -
3. 1.2 INT BUILTIN_CMD(CHAR **ARGV)函数（5 分） - 10 -
3. 1.3 VOID DO_BGFG(CHAR **ARGV) 函数（5 分） - 11 -
3. 1.4 VOID WAITFG(PID_T PID) 函数（5 分） - 11 -
3. 1.5 VOID SIGCHLD_HANDLER(INT SIG) 函数（10 分） - 12 -

第 4 章 TINYSHELL 测试 - 14 -

4.1 测试方法 - 14 -
4.2 测试结果评价 - 14 -
4.3 自测试结果 - 14 -
4.3.1 测试用例 trace01.txt - 14 -
4.3.2 测试用例 trace02.txt - 15 -
4.3.3 测试用例 trace03.txt - 15 -
4.3.4 测试用例 trace04.txt - 15 -
4.3.5 测试用例 trace05.txt - 15 -
4.3.6 测试用例 trace06.txt - 16 -
4.3.7 测试用例 trace07.txt - 16 -
4.3.8 测试用例 trace08.txt - 16 -
4.3.9 测试用例 trace09.txt - 17 -
4.3.10 测试用例 trace10.txt - 17 -
4.3.11 测试用例 trace11.txt - 17 -
4.3.12 测试用例 trace12.txt - 18 -
4.3.13 测试用例 trace13.txt - 18 -

4.3.14 测试用例 trace14.txt ..... *- 19 -*
4.3.15 测试用例 trace15.txt ..... *- 19 -*
4.4 自测试评分 ..... 错误!未定义书签。
第 5 章 总结 ..... **- 22 -**
5.1 请总结本次实验的收获 ..... - 22 -
5.2 请给出对本次实验内容的建议 ..... - 22 -
参考文献 ..... **- 24 -**

第 1 章 实验基本信息

1.1 实验目的

1. 理解现代计算机系统进程与并发的基本知识
2. 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
3. 掌握 shell 的基本原理和实现方法
4. 深入理解 Linux 信号响应可能导致的并发冲突及解决方法
5. 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

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；

1.2.3 开发工具

Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.3 实验预习

1. 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
2. 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。
3. 了解进程、作业、信号的基本概念和原理
4. 了解 shell 的基本原理
5. 熟知进程创建、回收的方法和相关系统函数
6. 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）

进程的概念：

一个特定的执行中程序的实例。

进程的创建：

父进程通过调用 fork()函数创建一个新的运行的子进程。fork()函数的返回值分以下情况：若在子进程中，返回 0；若在父进程中，则返回子进程的 PID。

进程的回收：

可以通过调用 waitpid（）函数来回收进程。除此之外，一个子进程可以通过它的父进程回收也可通过内核安排 init 进程来回收。

2.2 信号的机制、种类（5 分）

信号的机制：

在软件形式的层面上，存在 Linux 信号这一软件形式的异常。它允许进程和内核终端其他进程。信号的全称为软中断信号，简称软中断，一个信号就是一条小信息，它通知进程系统中发生了一个某种类型的事件。每种信号类型都对应于某种系统事件（在头文件<signal.h>中定义了 64 种信号，这些信号的名字都以 SIG 开头，且都被定义为正整数，称为信号编号。可以用“kill -l”查看信号的具体名称）。低层的硬件异常是由内核异常处理程序处理的，正常情况下，对用户进程而言是不可见的。信号提供了一种机制，使用信号来进行进程之间传递消息，通知用户进程发生了这些异常。

信号的种类：

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2

13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT
17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU
25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH
29) SIGIO 30) SIGPWR 31) SIGSYS
34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7
42) SIGRTMIN+8 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11
46) SIGRTMIN+12 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15
50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11
54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3
62) SIGRTMAX-2 63) SIGRTMAX-1 64) SIGRTMAX

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

信号的发送方法：

1. 用 /bin/kill 程序发送信号，/bin/kill 程序可以向另外的进程或进程组发送任意的信号，负的 PID 会导致信号被发送到进程组 PID 中的每个进程；
2. 从键盘发送信号。例如，从键盘上输入 ctrl-c (ctrl-z) 会导致内核发送一个 SIGINT (SIGTSTP)信号到前台进程组中的每个作业 ；
3. 用 kill 函数发送信号。int kill(pid_t pid,int sig);如果 pid 大于零，那么 kill 函数发送信号号码 sig 给进程 pid；如果 pid 等于零，那么 kill 发送信号 sig 给调用进程所在进程组中的每个进程，包括调用进程自己；如果 pid 小于零，kill 发送信号 sig 给进程组|pid|（pid 的绝对值）中的每个进程。
4. 用 alarm 函数发送信号。Unsigned int alarm(unsigned int secs);alarm 函数安排内核在 secs 秒后发送一个 SIGALRM 信号给调用进程。如果 secs 为零，那么不会调度安排新的闹钟（alarm）。在任何情况下，对 alarm 的调用都

将取消任何待处理的闹钟，并且返回任何待处理的闹钟在被发送前还剩下的秒数如果没有任何待处理的闹钟就返回零。

信号的阻塞方法：

1. 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型的待处理的信号。
2. 显式阻塞机制。应用程序可以使用 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。

处理程序的设置方法：

通过把处理程序的地址传递给 signal 函数从而改变默认行为，这叫做设置信号处理程序。

进程可以使用 signal 函数修改和信号 signum 相关联的默认行为

handler_t *signal(int signum, handler_t *handler)

signal 函数可以通过下列三种方法之一来改变和信号 signum 相关联的行为：

1. 如果 handler 是 SIG_IGN，那么忽略类型为 signum 的信号；
2. 如果 handler 是 SIG_DFL，那么类型为 signum 的信号行为恢复为默认行为；
3. 否则，handler 就是用户定义的函数的地址，这个函数被称为信号处理程序，只要接收到一个类型为 signum 的信号，就会调用这个程序。

2.4 什么是 shell，功能和处理流程（5 分）

什么是 shell：

Shell 是一种交互型的应用级程序，它可以代表用户执行程序。

功能：

执行一系列的读/求值步骤然后终止。该步骤读取来自用户的一个命令行。求值步骤读取来自用户的一个命令行，求值步骤解析该命令行，并代表用户执行程序。在解析命令后，如果是内置命令，则解析指令并执行，否则就根据相关文件路径执行可执行目标文件。

处理流程：

命令行是一串 ASCII 字符由空格分隔。字符串的第一个单词是一个可执行程

序，或者是 shell 的内置命令。命令行的其余部分是命令的参数。如果第一个单词是内置命令，shell 会立即在当前进程中执行。否则，shell 会新建一个子进程，然后再子进程中执行程序。新建的子进程又叫做作业。通常，作业可以由 Unix 管道连接的多个子进程组成。如果命令行以 &符号结尾，那么作业将在后台运行，这意味着在打印提示符并等待下一个命令之前，shell 不会等待作业终止。否则，作业在前台运行，这意味着 shell 在作业终止前不会执行下一条命令行。 因此，在任何时候，最多可以在一个作业中运行在前台。 但是，任意数量的作业可以在后台运行。例如，键入命令行：sh> jobs，会让 shell 运行内置命令 jobs。键入命令行 sh> /bin/ls -l -d 会导致 shell 在前台运行 ls 程序。根据约定，shell 会执行程序的 main 函数 int main(int argc, char *argv[])，argc 和 argv 会接收到下面的值：

argc == 3,

argv[0] == “/bin/ls”,

argv[1]== “-l”,

argv[2]== “-d”.

下面以&结尾的命令行会在后台执行 ls 程序

sh> /bin/ls -l -d &

Unix shell 支持作业控制的概念，允许用户在前台和后台之间来回移动作业，并更改进程的状态（运行，停止或终止）。在作业运行时，键入 ctrl-c 会将 SIGINT 信号传递到前台作业中的每个进程。SIGINT 的默认动作是终止进程。类似地，键入 ctrl-z 会导致 SIGTSTP 信号传递给所有前台进程。SIGTSTP 的默认操作是停止进程，直到它被 SIGCONT 信号唤醒为止。Unixshell 还提供支持作业控制的各种内置命令。例如：

jobs：列出运行和停止的后台作业。

bg <job>：将停止的后台作业更改为正在运行的后台作业。

fg <job>：将停止或运行的后台作业更改为在前台运行。

kill <job>：终止作业。

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：解析用户输入

参　　数：char *cmdline

处理流程：

1. 首先调用 parseline 函数，通过这一函数解析用户输入的以空格进行分隔的命令行参数，并构造 argv 向量。parseline 函数最后返回 int 值 bg，用于标志是否在后台运行。
2. 调用 builtin_cmd 函数，传入 argv，判断第一个输入的命令行参数是否为 quit,bg,fg,jobs 这四条内置命令之一，若是则执行内置命令并返回 1，否则返回 0。若 builtin_cmd 返回 0，则 shell 需要 fork 一个子进程并将该任务在子进程的上下文中运行。

要点分析：

1. 每个子进程都需要单独成组，即：每个子进程都要有自己的 PGID。否则在收到键盘输入 ctrl+c 或者 ctrl+z 时，子进程会被系统视为与 tsh 共处于同一进程组中，一起被有关信号终止。具体的实现方式是在 fork 后使用 setpgid(0,0)。
2. 为防止出现子进程与信号处理进程竞争 jobs 的情况，在 fork 子进程前先阻塞了 SIGCHLD、SIGINT、SIGSTP 信号。
3. 在父进程创建子进程且通过 addjob 记录后，再次调用 sigprocmask 解除阻塞。

3.1.2 int builtin_cmd(char **argv)函数（5 分）

函数功能：
检查输入的命令行参数是否为 quit,bg,fg,jobs 这四条内置命令之一，若是则执行内置命令并返回 1，否则返回 0。

参　　数：
char **argv

处理流程：
分别用 if 语句结合 strcmp 函数判断命令行参数是否为 quit,bg,fg,jobs 这四条内置命令之一，若是则立即执行，并返回 1 表示其为内置命令，否则返回 0。

要点分析：
1. 返回 1 表示参数为内置命令，否则返回 0。

2. 针对不同的参数可以方便地拓展出不同的语句，可拓展性较强。

3.1.3 void do_bgfg(char **argv) 函数（5 分）

函数功能：
具体实现内置命令 bg 与 fg

参　　数：
char **argv

处理流程：
1. 先检查命令是否携带着符合要求的参数，需要区分输入的是 pid 还是 jid 以便后续调用不同的函数，最终，将 ID 存于 jobp 中。
2. 若为 bg，则需要恢复后台进程，将 job->state 改为 BG，并发送信号 SIGCONT；若为 fg，则需要恢复前台进程，将 job->state 改为 FG,然后调用 waitfg，等待前台进程运行结束。

要点分析：
1. 检查参数时要考虑两种情况：需要区分输入的是 pid 还是 jid，最终都将 ID 存于 jobp 中。
2. 需要实时修改进程作业状态，向目标进程所在的进程组发送 SIGCONT 信号。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：
等待前台子进程的完成

参　　数：
pid_t pid

处理流程：
当前台进程的 pid 一直等于输入参数时，程序休眠，直到前台进程的 pid 不等于输入参数。

要点分析：
1. 前台进程 pid 等于输入参数，说明此时的前台进程就是当前传入 pid 对应的进程，需要等待
2. 通过 while 循环中的 sleep 函数达到等待的效果

3.1.5 void sigchld_handler(int sig) 函数（10 分）

函数功能：
捕获 SIGCHILD 信号。并进行回收子进程中的僵死进程、改变被终止进程状态标识、改变被挂起进程状态标识的工作。

参　　数：
int sig

处理流程：
1. 设置变量 olderrno 保存原始 errno 值
2. 循环判断，在子进程出现停止/终止时运行具体的处理程序：
   ①若子进程通过调用 exit 或 return 正常终止，则在阻塞信号的情况下 deletejobs，完成后解除信号阻塞。
   ②若子进程因为一个未捕获的信号而终止，则打印提示信息，并在阻塞信号的情况下 deletejobs，完成后解除信号阻塞。
   ③若引起返回的子进程当前是停止的，则打印提示信息，并将该进程状态设置为挂起。
3. 还原 errno 并退出

要点分析：
1. 对原始 errno 值进行保护
2. 循环的判断条件是(pid = waitpid(-1, &status, WUNTRACED | WNOHANG)，若等待集合中的子进程都未停止/终止，则 waitpid 返回 0，不执行 while 循环；while 循环只在子进程出现停止/终止时运行
3. 阻塞信号的方法是 sigprocmask(SIG_BLOCK, &mask, &prev);其中 mask 通过 sigfillset 保存着所有信号；还原信号的方法是 sigprocmask(SIG_SETMASK, &prev, NULL);prev 在之前的操作中保存着 blocked 中的原始信号

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)，并填写完成 4.3 节的相应表格。

4.2 测试结果评价

tsh 与 tshref 的输出在以下两个方面可以不同：

（1）pid

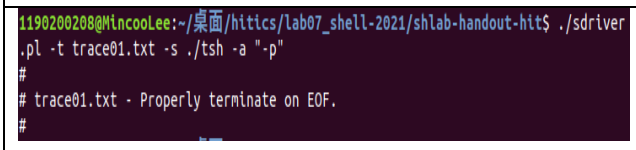
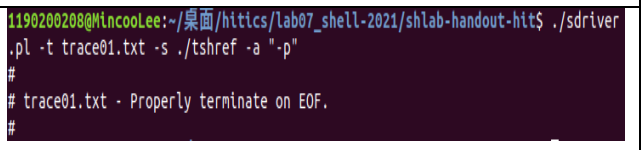
（2)测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令，每次运行的输出都会不同，但每个 mysplit 进程的运行状态应该相同。

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。

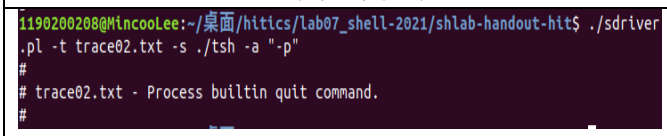
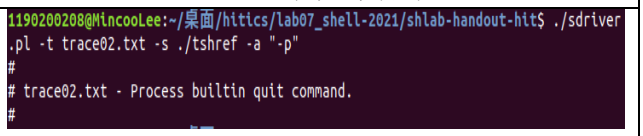
4.3 自测试结果

填写以下各个测试用例的测试结果，每个测试用例 1 分。

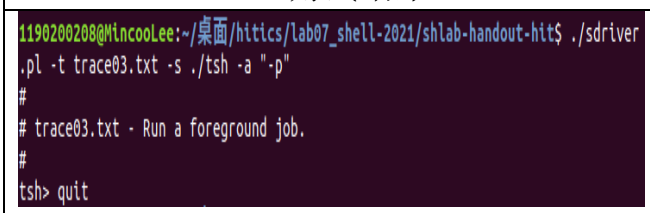
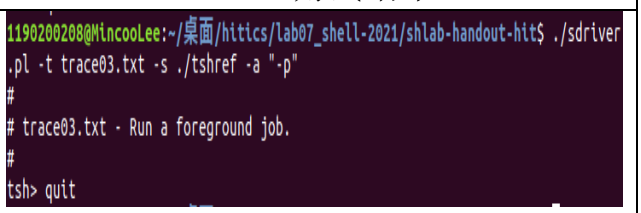
4.3.1 测试用例 trace01.txt

<table>
<tr><td>tsh 测试结果</td><td>tshref 测试结果</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>测试结论</td><td colspan="2">相同</td></tr>
</table>

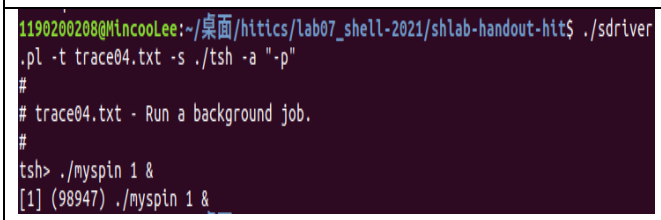
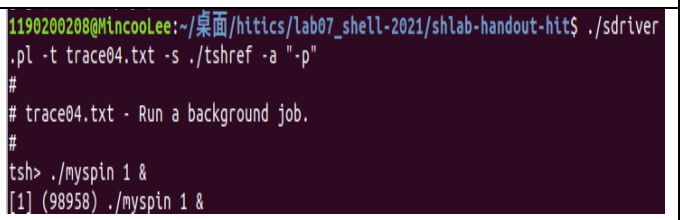
4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.3 测试用例 trace03.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.4 测试用例 trace04.txt

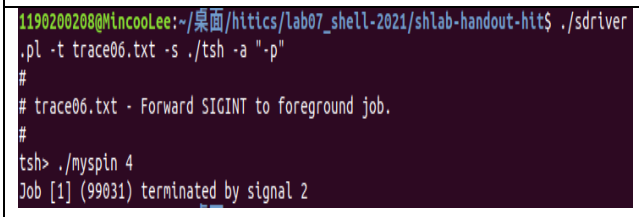
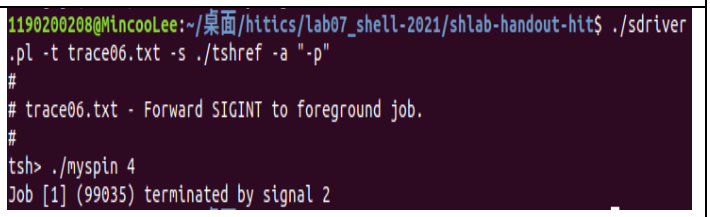
tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.5 测试用例 trace05.txt

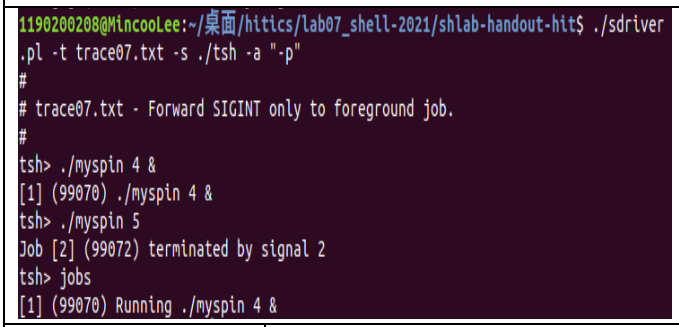
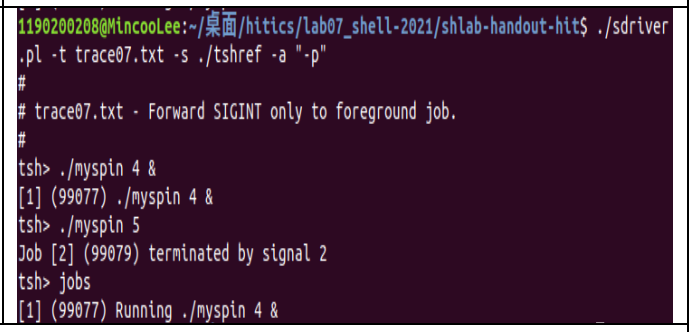
tsh 测试结果	tshref 测试结果

<table>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>测试结论</td><td colspan="2">相同</td></tr>
</table>

4.3.6 测试用例 trace06.txt

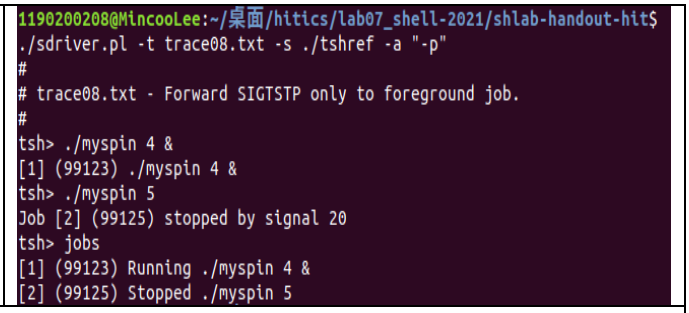
<table>
<tr><td>tsh 测试结果</td><td>tshref 测试结果</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>测试结论</td><td colspan="2">相同</td></tr>
</table>

4.3.7 测试用例 trace07.txt

<table>
<tr><td>tsh 测试结果</td><td>tshref 测试结果</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>测试结论</td><td colspan="2">相同</td></tr>
</table>

4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果

	
测试结论	相同

4.3.9 测试用例 trace09.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

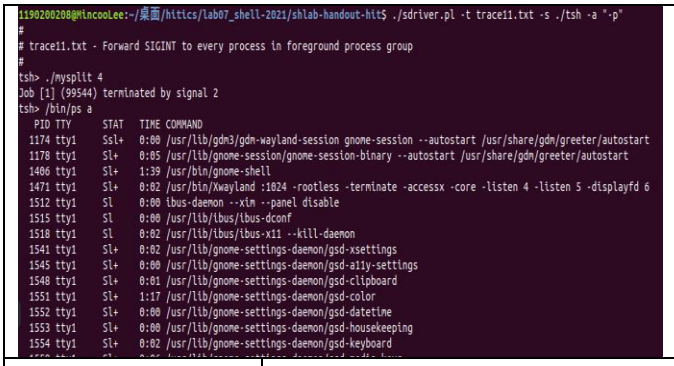
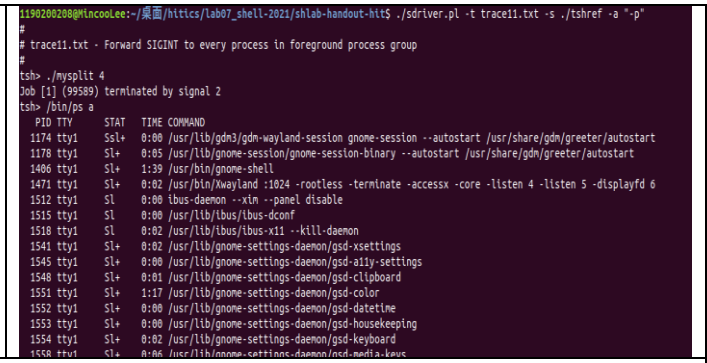
4.3.10 测试用例 trace10.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.11 测试用例 trace11.txt

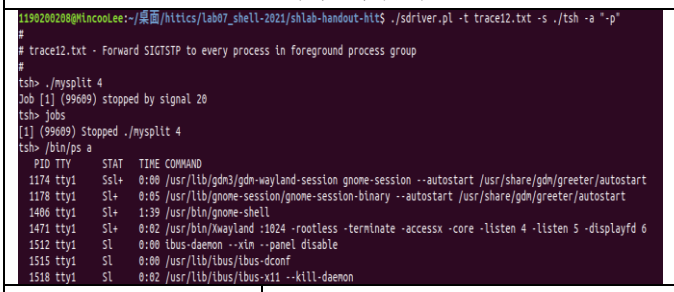
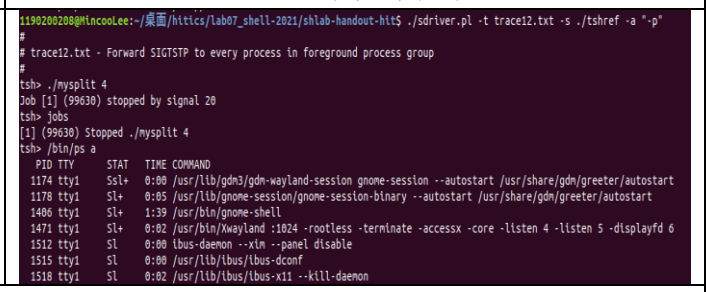
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果

	
测试结论	相同

4.3.12 测试用例 trace12.txt

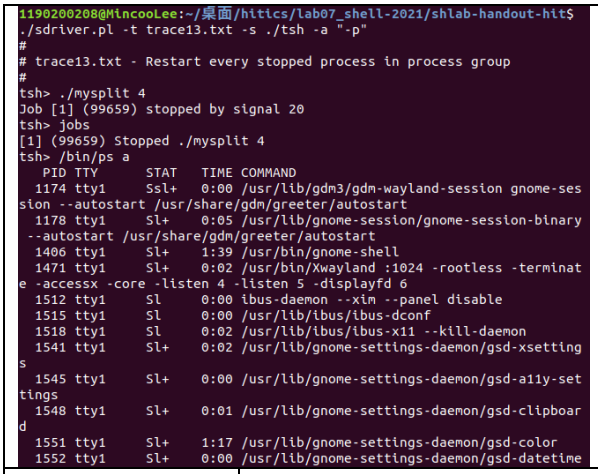
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
	
测试结论	相同

4.3.13 测试用例 trace13.txt

测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果



```
1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (99659) stopped by signal 20
tsh> jobs
[1] (99659) Stopped ./mysplit 4
tsh> /bin/ps a
   PID TTY      STAT   TIME COMMAND
  1174 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-wayland-session gnome-ses
sion --autostart /usr/share/gdm/greeter/autostart
  1178 tty1     Sl+    0:05 /usr/lib/gnome-session/gnome-session-binary
 --autostart /usr/share/gdm/greeter/autostart
  1406 tty1     Sl+    1:39 /usr/bin/gnome-shell
  1471 tty1     Sl+    0:02 /usr/bin/Xwayland :1024 -rootless -terminat
e -accessx -core -listen 4 -listen 5 -displayfd 6
  1512 tty1     Sl     0:00 ibus-daemon --xim --panel disable
  1515 tty1     Sl     0:00 /usr/lib/ibus/ibus-dconf
  1518 tty1     Sl     0:02 /usr/lib/ibus/ibus-x11 --kill-daemon
  1541 tty1     Sl+    0:02 /usr/lib/gnome-settings-daemon/gsd-xsetting
s
  1545 tty1     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-a11y-set
tings
  1548 tty1     Sl+    0:01 /usr/lib/gnome-settings-daemon/gsd-clipboar
d
  1551 tty1     Sl+    1:17 /usr/lib/gnome-settings-daemon/gsd-color
  1552 tty1     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-datetime
```

```
1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$ ./sdriver.pl -t tra
ce13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (99686) stopped by signal 20
tsh> jobs
[1] (99686) Stopped ./mysplit 4
tsh> /bin/ps a
   PID TTY      STAT   TIME COMMAND
  1174 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-wayland-session gnome-session --autostart /u
sr/share/gdm/greeter/autostart
  1178 tty1     Sl+    0:05 /usr/lib/gnome-session/gnome-session-binary --autostart /usr/s
hare/gdm/greeter/autostart
  1406 tty1     Sl+    1:39 /usr/bin/gnome-shell
  1471 tty1     Sl+    0:02 /usr/bin/Xwayland :1024 -rootless -terminate -accessx -core -l
isten 4 -listen 5 -displayfd 6
  1512 tty1     Sl     0:00 ibus-daemon --xim --panel disable
  1515 tty1     Sl     0:00 /usr/lib/ibus/ibus-dconf
  1518 tty1     Sl     0:02 /usr/lib/ibus/ibus-x11 --kill-daemon
  1541 tty1     Sl+    0:02 /usr/lib/gnome-settings-daemon/gsd-xsettings
  1545 tty1     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-a11y-settings
  1548 tty1     Sl+    0:01 /usr/lib/gnome-settings-daemon/gsd-clipboard
  1551 tty1     Sl+    1:17 /usr/lib/gnome-settings-daemon/gsd-color
  1552 tty1     Sl+    0:00 /usr/lib/gnome-settings-daemon/gsd-datetime
```

测试结论	相同

4.3.14 测试用例 trace14.txt

tsh 测试结果	tshref 测试结果

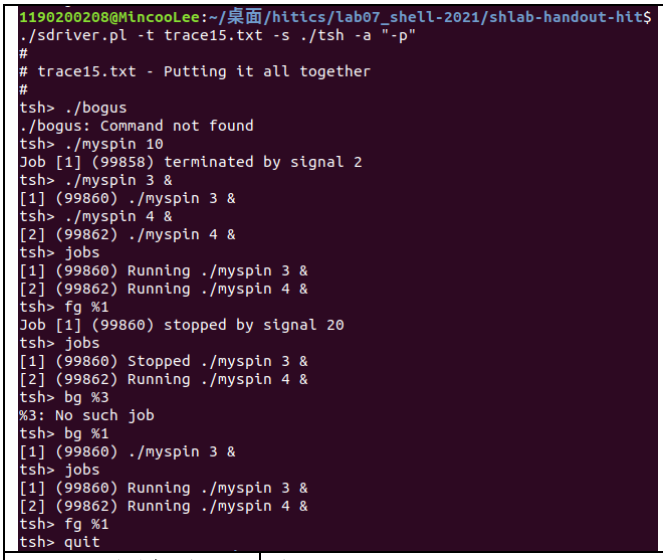
```
1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (99765) ./myspin 4 &
tsh> fg
fg command requires PID or %jobid argument
tsh> bg
bg command requires PID or %jobid argument
tsh> fg a
fg: argument must be a PID or %jobid
tsh> bg a
bg: argument must be a PID or %jobid
tsh> fg 9999999
(9999999): No such process
tsh> bg 9999999
(9999999): No such process
tsh> fg %2
%2: No such job
tsh> fg %1
Job [1] (99765) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (99765) ./myspin 4 &
tsh> jobs
[1] (99765) Running ./myspin 4 &
```

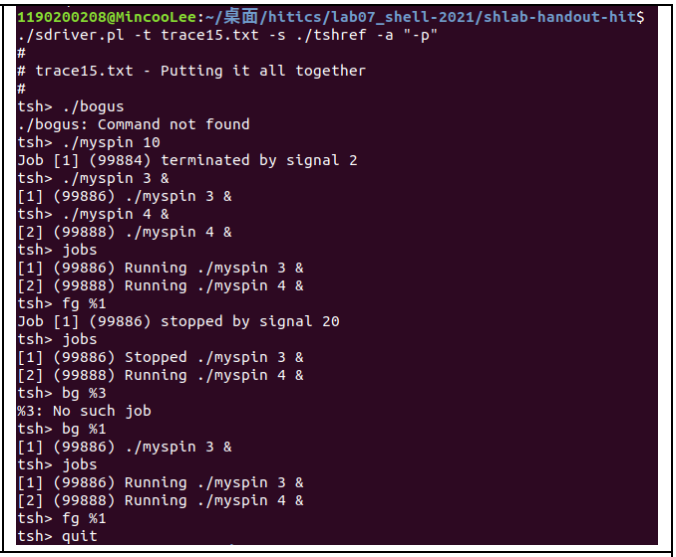
```
1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (99801) ./myspin 4 &
tsh> fg
fg command requires PID or %jobid argument
tsh> bg
bg command requires PID or %jobid argument
tsh> fg a
fg: argument must be a PID or %jobid
tsh> bg a
bg: argument must be a PID or %jobid
tsh> fg 9999999
(9999999): No such process
tsh> bg 9999999
(9999999): No such process
tsh> fg %2
%2: No such job
tsh> fg %1
Job [1] (99801) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (99801) ./myspin 4 &
tsh> jobs
[1] (99801) Running ./myspin 4 &
```

测试结论	相同

4.3.15 测试用例 trace15.txt

tsh 测试结果	tshref 测试结果


1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (99858) terminated by signal 2
tsh> ./myspin 3 &
[1] (99860) ./myspin 3 &
tsh> ./myspin 4 &
[2] (99862) ./myspin 4 &
tsh> jobs
[1] (99860) Running ./myspin 3 &
[2] (99862) Running ./myspin 4 &
tsh> fg %1
Job [1] (99860) stopped by signal 20
tsh> jobs
[1] (99860) Stopped ./myspin 3 &
[2] (99862) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (99860) ./myspin 3 &
tsh> jobs
[1] (99860) Running ./myspin 3 &
[2] (99862) Running ./myspin 4 &
tsh> fg %1
tsh> quit


1190200208@MincooLee:~/桌面/hitics/lab07_shell-2021/shlab-handout-hit$
./sdriver.pl -t trace15.txt -s ./tshref -a "-p"
#
trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (99884) terminated by signal 2
tsh> ./myspin 3 &
[1] (99886) ./myspin 3 &
tsh> ./myspin 4 &
[2] (99888) ./myspin 4 &
tsh> jobs
[1] (99886) Running ./myspin 3 &
[2] (99888) Running ./myspin 4 &
tsh> fg %1
Job [1] (99886) stopped by signal 20
tsh> jobs
[1] (99886) Stopped ./myspin 3 &
[2] (99888) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (99886) ./myspin 3 &
tsh> jobs
[1] (99886) Running ./myspin 3 &
[2] (99888) Running ./myspin 4 &
tsh> fg %1
tsh> quit

测试结论	相同

第 5 章 评测得分

总分 20 分

实验程序统一测试的评分（教师评价）：

（1）正确性得分：_____（满分 10）

（2）性能加权得分：_____（满分 10）

第 6 章 总结

5.1 请总结本次实验的收获

1. 对现代计算机系统进程的概念有了更深刻的了解
2. 掌握了 linux 异常控制流和信号机制的基本原理和相关系统函数
3. 掌握了 shell 的基本原理和简单功能的实现方法
4. 深入了理解 Linux 信号响应可能导致的并发冲突及解决方法

5.2 请给出对本次实验内容的建议

如果条件充分的话，可以对 tsh.c 的基本架构略加讲解，该文件中的一些函数已经编写好，由于时间原因自始至终都没有认真研究，如果能讲解一下应该可以对 Shell 的工作原理理解地更为深刻。

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] 谌颖. 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.