

哈尔滨工业大学

实验报告

实验（四）

题 目 TinyShell

微壳

专 业 计算机类

学 号 120L022115

班 级 2003007

学 生 王炳轩

指 导 教 师 吴锐

实 验 地 点 线上

实 验 日 期 2022-05-02

计算学部

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	- 5 -
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，简述其功能和处理流程（5 分）	- 7 -
第 3 章 TINY SHELL 的设计与实现	- 9 -
3.1.1 VOID EVAL (CHAR *CMDLINE) 函数（10 分）	- 9 -
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）	- 12 -
3.1.5 VOID SIGCHLD_HANDLER (INT SIG) 函数（10 分）	- 13 -
第 4 章 TINY SHELL 测试	- 15 -
4.1 测试方法	- 15 -
4.2 测试结果评价	- 15 -
4.3 自测试结果	- 15 -
4.3.1 测试用例 trace01.txt 的输出截图（1 分）	- 15 -
4.3.2 测试用例 trace02.txt 的输出截图（1 分）	- 15 -
4.3.3 测试用例 trace03.txt 的输出截图（1 分）	- 16 -
4.3.4 测试用例 trace04.txt 的输出截图（1 分）	- 16 -
4.3.5 测试用例 trace05.txt 的输出截图（1 分）	- 16 -
4.3.6 测试用例 trace06.txt 的输出截图（1 分）	- 17 -
4.3.7 测试用例 trace07.txt 的输出截图（1 分）	- 17 -
4.3.8 测试用例 trace08.txt 的输出截图（1 分）	- 17 -
4.3.9 测试用例 trace09.txt 的输出截图（1 分）	- 18 -
4.3.10 测试用例 trace10.txt 的输出截图（1 分）	- 18 -
4.3.11 测试用例 trace11.txt 的输出截图（1 分）	- 18 -
4.3.12 测试用例 trace12.txt 的输出截图（1 分）	- 19 -
4.3.13 测试用例 trace13.txt 的输出截图（1 分）	- 19 -

4.3.14 测试用例 <i>trace14.txt</i> 的输出截图 (1 分)	- 21 -
4.3.15 测试用例 <i>trace15.txt</i> 的输出截图 (1 分)	- 22 -
4.4 自测试评分.....	- 22 -
第 4 章 总结	- 23 -
4.1 请总结本次实验的收获.....	- 23 -
4.2 请给出对本次实验内容的建议.....	- 23 -
参考文献.....	- 24 -

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

Surface Go 3: x64、Pentium G6500Y @ 1.1GHz、16GB RAM、128GB SSD。

1.2.2 软件环境

Windows 11、Windows Subsystem for Linux、Ubuntu 20.04

1.2.3 开发工具

Code::Blocks、gcc、vim、edb、gdb

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：是一个程序在操作系统中的执行实例，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体。

进程的创建方法：在 shell 中，当用户运行新程序时，系统会调用 `fork` 复制与当前 shell 进程相同的一份资源给新程序，然后调用 `execve` 装载程序资源。

进程的回收方法：当子进程走完了自己的生命周期后，它会执行 `exit()` 系统调用，此时原来进程表中的数据会被该进程的退出码(exit code)、执行时所用的 CPU 时间等数据所取代，这些数据会一直保留到系统将它传递给它的父进程（必须使用 `wait` 或 `waitpid`）为止，此段时间，子进程会成为僵尸进程。之后，子进程由父进程（创建它的进程）回收，当父进程比子进程提前结束时，由系统 `init` 进程负责回收。

2.2 信号的机制、种类（5 分）

Linux 信号就是一条小消息，它通知进程系统中发生了一个某种类型的事件，类似于异常和中断。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来标识的：{1-30}。信号中唯一的信息是它的 ID 和它的到达。

信号可以被进程阻塞（延迟接收处理）、忽略、捕获并自定义处理或执行默认处理行为。

内核为每一个进程维护着两个 `n` 位二进制数（`n` 为操作系统的信号数量）：`blocks` 和 `pendings`，这两个二进制数的每一位分别表示对第 `i` 个信号的阻塞/发生，因此，信号不能累计，每一种信号最多只能被发送 1 次直到被接收并清空标志位。

常见的信号种类：

ID	名称	默认行为	相应事件
2	SIGINT	终止	来自键盘的中断
9	SIGKILL	终止	杀死程序(该信号不能被捕获不能被忽略)
11	SIGSEGV	终止	无效的内存引用（段故障）
14	SIGALRM	终止	来自alarm函数的定时器信号
17	SIGCHLD	忽略	一个子进程停止或者终止

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

2.3.1 发送方法

被动发送信号：

系统检测到进程发生除零错误、非法内存访问等异常时。

主动发送信号：

1、在 bash 下通过 kill 命令：

给进程组发送：kill -信号 ID -PGID

给单一进程发送：kill -信号 ID PID（少一个负号-）

2、通过调用系统函数 kill：kill(PID, SIGINT);

3、从键盘发送信号（到前台进程组）：

Ctrl+c:发送 SIGINT（终止）

Ctrl+z:发送 SIGTSTP（挂起停止）

2.3.2 阻塞与解除阻塞信号的方法

隐式阻塞机制

内核默认阻塞与当前正在处理信号类型相同的待处理信号。

显式阻塞和解除阻塞机制

sigprocmask 函数及其辅助函数可以明确地阻塞/解除阻塞选定的信号。

辅助函数：

- sigemptyset——初始化 set 为空集合
- sigfillset——把每个信号都添加到 set 中
- sigaddset——把指定的信号 signum 添加到 set 中
- sigdelset——从 set 中删除指定的信号

2.3.3 处理程序的设置方法

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)`

`handler` 的不同取值:

- `SIG_IGN`: 忽略类型为 `signum` 的信号
- `SIG_DFL`: 类型为 `signum` 的信号行为恢复为默认行为
- 否则, `handler` 就是用户定义的函数的地址, 这个函数称为**信号处理程序**
 - 只要进程接收到类型为 `signum` 的信号就会调用**信号处理程序**
 - 将处理程序的地址传递到 `signal` 函数从而改变默认行为, 这叫作**设置信号处理程序**
 - 调用**信号处理程序**称为**捕获信号**
 - 执行**信号处理程序**称为**处理信号**
 - 当处理程序执行 `return` 时, 控制会传递到控制流中被信号接收所中断的指令处

2.4 什么是 shell, 简述其功能和处理流程 (5 分)

Shell 俗称壳(用来区别于核),是指“为用户提供操作界面”的软件(command interpreter, 命令解析器)。它类似于 DOS 下的 COMMAND.COM 和后来的 cmd.exe。它接收用户命令, 然后调用相应的应用程序。同时它又是一种程序设计语言。作为命令语言, 它交互式解释和执行用户输入的命令或者自动地解释和执行预先设定好的一连串的命令; 作为程序设计语言, 它定义了各种变量和参数, 并提供了许多在高级语言中才具有的控制结构, 包括循环和分支。

基本上 shell 分两大类:

一、图形界面 shell (Graphical User Interface shell 即 GUI shell)

例如: 应用最为广泛的 Windows Explorer (微软的 Windows 系列操作系统), 还有也包括广为人知的 Linux shell, 其中 Linux shell 包括 X Window Manager (BlackBox 和 FluxBox), 以及功能更强大的 CDE、GNOME、KDE、Xfce。

二、命令行式 shell (Command Line Interface shell, 即 CLI shell)

例如: sh (Bourne Shell) /csh/tcsh/bash/ksh/zsh/fish 等 (Unix 及类 unix)

COMMAND.COM (CP/M 系统; MS-DOS、PC-DOS、DR-DOS、FreeDOS 等 DOS; Windows 9x)

cmd.exe/命令提示符 (OS/2、Windows NT、React OS)

Windows PowerShell (支持 .NET Framework 技术的 Windows NT)

shell 的功能就是执行用户发出的命令，常用的命令如下：

常用命令

命令	备注	适用于的命令解释程序
cat [文件名]	输出文件内容到基本输出（屏幕 or 加>fileName 到另一个文件）	Unix、类 unix、Haiku 操作系统
cb	格式化源代码	Unix、类 unix
chmod	改变文件的权限	Unix、类 unix
cp	复制文件	MULTICS、Unix、类 unix、Haiku 操作系统
date	当前的时间和日期	Unix、类 unix、Haiku 操作系统
echo \$abc	在变量赋值之后，只需在变量前面加一个\$去引用。	Unix、类 unix、Haiku 操作系统、Windows Powershell
lint	语法检查程序	Unix、类 unix、Haiku 操作系统
ls	列出文件目录	MULTICS、Unix、类 unix、Haiku 操作系统、Windows Powershell
dir	列出文件目录	Debian GNU/Linux、Cygwin
man	查询命令	Unix、类 unix、Haiku 操作系统
more	查看文本文件内容	Unix、类 unix、Haiku 操作系统
du	查看磁盘空间状况	Unix、类 unix、Haiku 操作系统
uname	查看当前操作系统名称或版本号	Unix、类 unix
who ^[2]	你的用户名和终端类型 定义变量 name	Unix、类 unix
ps	查看当前进程状况 =abc? (bash/pdksh) set name = abc (tcsh)	Unix、类 unix、Haiku 操作系统
mv	改文件名/移动文件	Unix、类 unix、Haiku 操作系统
pwd	显示目录路径命令	Unix、类 unix
exit	登出	Unix、类 unix
logout	登出	Unix、类 unix（FreeBSD 操作系统中仅适用于 root 用户）
rm	删除文件	Unix、类 Unix、Haiku 操作系统
echo [文本]	显示指定文本	Unix、类 unix、Haiku 操作系统、DOS、OS/2、Windows、React OS（Windows NT cmd.exe 中，[文本]不能为 on 或 off）
help [命令]	列出一个命令的详细信息	Cygwin、Windows NT
shutdown	关机	Unix、类 unix、Windows NT、React OS
cd	更改工作目录	Unix、类 unix、Haiku 操作系统、OS/2、Windows、React OS；DOS 从 2.0 版起
chdir	更改工作目录	Unix、类 unix、Haiku 操作系统、OS/2、Windows、React OS；DOS 从 2.0 版起
rmdir	删除目录	Unix、类 unix、Haiku 操作系统
mkdir	创建目录	Unix、类 unix、Haiku 操作系统、OS/2、Windows、React OS；DOS 从 2.0 版起

shell 的处理流程包括：解析指令、执行指令序列（调用系统函数）、打印执行结果信息。

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：执行用户刚刚输入的命令行：如果用户已请求内置命令（退出、作业、bg 或 fg），则立即执行该命令。否则，派生一个子进程，并在该子进程的上下文中运行该作业。如果作业正在前台运行，等待它终止，然后返回。每个子进程有一个唯一的进程组 ID，这样当我们在键盘上键入 ctrl-c (ctrl-z) 时，我们的后台子进程就不会从内核接收 SIGINT (SIGTSTP)。

参 数：char *cmdline 命令行的字符起始指针。

处理流程：初始化、取出指令并判断是否为 builtin-command、设置异常信号阻塞、执行 fork、对于父进程则等待子进程运行结束回收资源，对于子进程则调用 execve 执行程序。

要点分析：

初始化并分析命令：

```
/* $begin handout */
char *argv[MAXARGS]; /* argv for execve() */
int bg;               /* should the job run in bg or fg? */
pid_t pid;            /* process id */
sigset_t mask;        /* signal mask */

/* Parse command line */
bg = parseline(cmdline, argv);
if (argv[0] == NULL)
    return; /* ignore empty lines */
```

设置信号处理：

```
if (sigemptyset(&mask) < 0)
    unix_error("sigemptyset error");
if (sigaddset(&mask, SIGCHLD))
    unix_error("sigaddset error");
if (sigaddset(&mask, SIGINT))
    unix_error("sigaddset error");
if (sigaddset(&mask, SIGTSTP))
    unix_error("sigaddset error");
if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0)
    unix_error("sigprocmask error");
```

fork：

```
/* Create a child process */
if ((pid = fork()) < 0)
    unix_error("fork error");
```

子进程：

```

if (pid == 0) {
    /* Child unblocks signals */
    sigprocmask(SIG_UNBLOCK, &mask, NULL);

    /* Each new job must get a new process group ID
       so that the kernel doesn't send ctrl-c and ctrl-z
       signals to all of the shell's jobs */
    if (setpgid(0, 0) < 0)
        unix_error("setpgid error");

    /* Now load and run the program in the new job */
    if (execve(argv[0], argv, environ) < 0) {
        printf("%s: Command not found\n", argv[0]);
        exit(0);
    }
}

```

父进程:

```

/* Parent adds the job, and then unblocks signals so that
   the signals handlers can run again */
addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
sigprocmask(SIG_UNBLOCK, &mask, NULL);

if (!bg)
    waitfg(pid);
else
    printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
}
/* $end handout */
return;

```

3. 1.2 int builtin_cmd(char **argv) 函数 (5 分)

函数功能: 识别并解释内置命令: quit, fg, bg, 和 jobs.

参 数: char **argv 命令行

处理流程: 读入命令行, 挨个判断调用即可。

要点分析:

```
int builtin_cmd(char** argv)
{
    char* head = argv[0]; //, p = argv[0];
    //int argc = 0;
    if (head == (char*)NULL) return 1;
    /*while(p!=NULL){
        argc++;
        p = argv[argc];
    }*/
    if (strcmp("quit", head) == 0) { //quit
        //printf("Quit Shell! Back Control Terminal\n");
        exit(0);
    }
    else if (strcmp("fg", head) == 0) { // fg
        do_bgfg(argv);
        return 1;
    }
    else if (strcmp("bg", head) == 0) { // bg
        do_bgfg(argv);
        return 1;
    }
    else if (strcmp("jobs", head) == 0) { // jobs
        //printf("jobs\n");
        listjobs(jobs);
        return 1;
    }
}
```

3. 1.3 void do_bgfg(char **argv) 函数 (5 分)

函数功能: 实现内置命令 **bg** 和 **fg**

参 数: char** argv

处理流程: 先判断是否合法, 再执行。

要点分析:

判断合法:

```
/* Ignore command if no argument */
if (argv[1] == NULL) {
    printf("%s command requires PID or %%jobid argument\n", argv[0]);
    return;
}

/* Parse the required PID or %JID arg */
if (isdigit(argv[1][0])) {
    pid_t pid = atoi(argv[1]);
    if (!(jobp = getjobpid(jobs, pid))) {
        printf("(%d): No such process\n", pid);
        return;
    }
}
else if (argv[1][0] == '%') {
    int jid = atoi(&argv[1][1]);
    if (!(jobp = getjobjid(jobs, jid))) {
        printf("%s: No such job\n", argv[1]);
        return;
    }
}
else {
    printf("%s: argument must be a PID or %%jobid\n", argv[0]);
    return;
}
```

执行:

```
/* bg command */
if (!strcmp(argv[0], "bg")) {
    DONTWAIT = 1;
    if (kill(-(jobp->pid), SIGCONT) < 0)
        unix_error("kill (bg) error");
    jobp->state = BG;
    printf("[%d] (%d) %s", jobp->jid, jobp->pid, jobp->cmdline);
}

/* fg command */
else if (!strcmp(argv[0], "fg")) {
    DONTWAIT = 1;
    if (kill(-(jobp->pid), SIGCONT) < 0)
        unix_error("kill (fg) error");
    jobp->state = FG;
    waitfg(jobp->pid);
}
else {
    printf("do_bgfg: Internal error\n");
    exit(0);
}

/* $end handout */
```

3. 1.4 void waitfg(pid_t pid) 函数 (5 分)

函数功能：等待一个前台作业结束

参 数：pid_t pid

处理流程：等待 PID 为 pid 的前台进程作业结束。

要点分析：

```
void waitfg(pid_t pid)
{
    int i;
    //Signal(SIGCHLD, SIG_IGN);
    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid == pid) {
            while (jobs[i].pid == pid && jobs[i].state == FG) {
                sleep(1);
                //printf("WAITFG: %d\n", jobs[i].pid);
            }
            //printf("RETURN %d\n", jobs[i].pid);
            return;
        }
    }

    //printf("WAITING FG %d", pid);
    //Signal(SIGCHLD, sigchld_handler);
    //deletejob(jobs, pid);
    return;
}
```

3. 1.5 void sigchld_handler(int sig) 函数（10 分）

函数功能：捕获 SIGCHLD 信号

参 数：int sig

处理流程：子进程挂起、继续、终止时收到信号，使用 DONTWAIT 判断是否为终止，然后执行回收。

要点分析：

```
void sigchld_handler(int sig)
{
    int olderrno = errno;
    pid_t pid;
    //printf("REAP\n");
    if (!DONTWAIT) {
        if ((pid = wait(NULL)) < 0) {
            //if (errno != 10)
            //printf("%d", pid);
            unix_error("wait error");
        }
        deletejob(jobs, pid);
    }
    DONTWAIT = 0;

    errno = olderrno;
    return;
}
```

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
已注释
- （2）检查每个系统调用的返回值——5 分
已完成

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 mysplrit 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	
<pre>noname:shell>make test01 ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #</pre>		<pre>noname:shell>make rtest01 ./sdriver.pl -t trace01.txt -s ./tshref -a "-p" # # trace01.txt - Properly terminate on EOF. #</pre>	
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
----------	-------------

<pre> noname:shell>make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. # </pre>	<pre> noname:shell>make rtest02 ./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. # </pre>
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
<pre> noname:shell>make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit </pre>	<pre> noname:shell>make rtest03 ./sdriver.pl -t trace03.txt -s ./tshref -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit </pre>
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
<pre> noname:shell>make test04 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (30631) ./myspin 1 & noname:shell> </pre>	<pre> noname:shell>make rtest04 ./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (30625) ./myspin 1 & noname:shell> </pre>
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
----------	-------------

<pre> noname:shell>make test05 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (30648) ./myspin 2 & tsh> ./myspin 3 & [2] (30650) ./myspin 3 & tsh> jobs [1] (30648) Running ./myspin 2 & [2] (30650) Running ./myspin 3 & noname:shell> </pre>	<pre> noname:shell>make rtest05 ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (30639) ./myspin 2 & tsh> ./myspin 3 & [2] (30641) ./myspin 3 & tsh> jobs [1] (30639) Running ./myspin 2 & [2] (30641) Running ./myspin 3 & noname:shell> </pre>
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
<pre> noname:shell>make test06 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (30659) terminated by signal 2 noname:shell> </pre>	<pre> noname:shell>make rtest06 ./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (30665) terminated by signal 2 noname:shell> </pre>
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
<pre> noname:shell>make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (30683) ./myspin 4 & tsh> ./myspin 5 Job [2] (30685) terminated by signal 2 tsh> jobs [1] (30683) Running ./myspin 4 & </pre>	<pre> noname:shell>make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (30683) ./myspin 4 & tsh> ./myspin 5 Job [2] (30685) terminated by signal 2 tsh> jobs [1] (30683) Running ./myspin 4 & </pre>
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> noname:shell>make test08 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (30693) ./myspin 4 & tsh> ./myspin 5 Job [2] (30695) stopped by signal 20 tsh> jobs [1] (30693) Running ./myspin 4 & [2] (30695) Stopped ./myspin 5 noname:shell> </pre>	<pre> noname:shell>make rtest08 ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (30703) ./myspin 4 & tsh> ./myspin 5 Job [2] (30705) stopped by signal 20 tsh> jobs [1] (30703) Running ./myspin 4 & [2] (30705) 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> noname:shell>make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (30724) ./myspin 4 & tsh> ./myspin 5 Job [2] (30726) stopped by signal 20 tsh> jobs [1] (30724) Running ./myspin 4 & [2] (30726) Stopped ./myspin 5 tsh> bg %2 [2] (30726) ./myspin 5 tsh> jobs [1] (30724) Running ./myspin 4 & [2] (30726) Running ./myspin 5 </pre>	<pre> noname:shell>make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (30713) ./myspin 4 & tsh> ./myspin 5 Job [2] (30715) stopped by signal 20 tsh> jobs [1] (30713) Running ./myspin 4 & [2] (30715) Stopped ./myspin 5 tsh> bg %2 [2] (30715) ./myspin 5 tsh> jobs [1] (30713) Running ./myspin 4 & [2] (30715) 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> noname:shell>make test10 ./sdriver.pl -t trace10.txt -s ./tsh -a "-p" # # trace10.txt - Process fg builtin command. # tsh> ./myspin 4 & [1] (30736) ./myspin 4 & tsh> fg %1 Job [1] (30736) stopped by signal 20 tsh> jobs [1] (30736) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs </pre>	<pre> noname:shell>make rtest10 ./Settings ver.pl -t trace10.txt -s ./tshref -a "-p" # # trace10.txt - Process fg builtin command. # ^[atsh> ./myspin 4 & [1] (30746) ./myspin 4 & tsh> fg %1 Job [1] (30746) stopped by signal 20 tsh> jobs [1] (30746) Stopped ./myspin 4 & tsh> fg %1 tsh> 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
----------	-------------

<pre> noname:shell>make test11 ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh> ./mysplit 4 Job [1] (30757) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1990 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu 1992 tty2 Sl+ 1:41 /usr/lib/xorg/Xorg vt2 -displ layfd 3 -auth /run/user/1000/gdm/Xauthority -background n one -noreset -keeppty -verbose 3 2413 tty2 Sl+ 0:00 /usr/libexec/gnome-session-b inary --systemd --systemd --session=ubuntu 30104 pts/0 Ss+ 0:00 bash 30435 pts/1 Ss+ 0:00 bash 30752 pts/0 S+ 0:00 make test11 30753 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t t race11.txt -s ./tsh -a "-p" 30754 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl - t trace11.txt -s ./tsh -a -p 30755 pts/0 S 0:00 ./tsh -p 30760 pts/0 R 0:00 /bin/ps a noname:shell> </pre>	<pre> noname:shell>make rtest11 ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh> ./mysplit 4 Job [1] (30766) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1990 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu 1992 tty2 Sl+ 1:41 /usr/lib/xorg/Xorg vt2 -displ layfd 3 -auth /run/user/1000/gdm/Xauthority -background n c Settings oreset -keeppty -verbose 3 2413 tty2 Sl+ 0:00 /usr/libexec/gnome-session-b inary --systemd --systemd --session=ubuntu 30104 pts/0 Ss+ 0:00 bash 30435 pts/1 Ss 0:00 bash 30761 pts/1 S+ 0:00 make rtest11 30762 pts/1 S+ 0:00 /bin/sh -c ./sdriver.pl -t t race11.txt -s ./tshref -a "-p" 30763 pts/1 S+ 0:00 /usr/bin/perl ./sdriver.pl - t trace11.txt -s ./tshref -a -p 30764 pts/1 S+ 0:00 ./tshref -p 30769 pts/1 R 0:00 /bin/ps a noname:shell> </pre>
测试结论	相同

4.3.12 测试用例 trace12.txt 的输出截图（1 分）

<p style="text-align: center;">tsh 测试结果</p> <pre> # noname:shell>make test12 ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" # # trace12.txt - Forward SIGTSTP to every process in foreground process group # tsh> ./mysplit 4 Job [1] (30589) stopped by signal 20 tsh> jobs [1] (30589) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1990 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu 1992 tty2 Sl+ 1:35 /usr/lib/xorg/Xorg vt2 -displ layfd 3 -auth /run/user/1000/gdm/Xauthority -background n one -noreset -keeppty -verbose 3 2413 tty2 Sl+ 0:00 /usr/libexec/gnome-session-b inary --systemd --systemd --session=ubuntu 30104 pts/0 Ss 0:00 bash 30435 pts/1 Ss+ 0:00 bash 30584 pts/0 S+ 0:00 make test12 30585 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t t race12.txt -s ./tsh -a "-p" 30586 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl - t trace12.txt -s ./tsh -a -p 30587 pts/0 S 0:00 ./tsh -p 30589 pts/0 T 0:00 ./mysplit 4 30590 pts/0 T 0:00 ./mysplit 4 30593 pts/0 R 0:00 /bin/ps a noname:shell> </pre>	<p style="text-align: center;">tshref 测试结果</p> <pre> # noname:shell>make rtest12 ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" # # trace12.txt - Forward SIGTSTP to every process in foreground process group # tsh> ./mysplit 4 Job [1] (30599) stopped by signal 20 tsh> jobs [1] (30599) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1990 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu 1992 tty2 Sl+ 1:35 /usr/lib/xorg/Xorg vt2 -displ layfd 3 -auth /run/user/1000/gdm/Xauthority -background n one -noreset -keeppty -verbose 3 2413 tty2 Sl+ 0:00 /usr/libexec/gnome-session-b inary --systemd --systemd --session=ubuntu 30104 pts/0 Ss+ 0:00 bash 30435 pts/1 Ss 0:00 bash 30594 pts/1 S+ 0:00 make rtest12 30595 pts/1 S+ 0:00 /bin/sh -c ./sdriver.pl -t t race12.txt -s ./tshref -a "-p" 30596 pts/1 S+ 0:00 /usr/bin/perl ./sdriver.pl - t trace12.txt -s ./tshref -a -p 30597 pts/1 S+ 0:00 ./tshref -p 30599 pts/1 T 0:00 ./mysplit 4 30600 pts/1 T 0:00 ./mysplit 4 30603 pts/1 R 0:00 /bin/ps a noname:shell> </pre>
测试结论	相同

4.3.13 测试用例 trace13.txt 的输出截图（1 分）

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

```

noname:shell>make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process
in process group
#
tsh> ./mysplit 4
Job [1] (30822) stopped by signal 20
tsh> jobs
[1] (30822) Stopped ./mysplit 4
tsh> /bin/ps a
      PID TTY          STAT       TIME
COMMAND
    1990 tty2      Ssl+    0:00
/usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --
session=ubuntu
    1992 tty2      Sl+      1:43
/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1000/gdm/Xauthority -background
none -noreset -keeptty -verbose 3
    2413 tty2      Sl+      0:00
/usr/libexec/gnome-session-binary --systemd
--systemd --session=ubuntu
    30104 pts/0    Ss+      0:00 bash
    30435 pts/1    Ss        0:00 bash
    30817 pts/1    S+        0:00 make
rtest13
    30818 pts/1    S+        0:00 /bin/sh -
c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
    30819 pts/1    S+        0:00
/usr/bin/perl ./sdriver.pl -t trace13.txt -
s ./tshref -a -p
    30820 pts/1    S+        0:00 ./tshref -p
    30822 pts/1    T         0:00 ./mysplit 4
    30823 pts/1    T         0:00 ./mysplit 4
    30826 pts/1    R         0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
      PID TTY          STAT       TIME
COMMAND
    1990 tty2      Ssl+    0:00
/usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --
session=ubuntu
    1992 tty2      Sl+      1:43
/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1000/gdm/Xauthority -background
none -noreset -keeptty -verbose 3

```

```

noname:shell>make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process
in process group
#
tsh> ./mysplit 4
Job [1] (30809) stopped by signal 20
tsh> jobs
[1] (30809) Stopped ./mysplit 4
tsh> /bin/ps a
      PID TTY          STAT       TIME
COMMAND
    1990 tty2      Ssl+    0:00
/usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --
session=ubuntu
    1992 tty2      Sl+      1:43
/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1000/gdm/Xauthority -background
none -noreset -keeptty -verbose 3
    2413 tty2      Sl+      0:00
/usr/libexec/gnome-session-binary --systemd
--systemd --session=ubuntu
    30104 pts/0    Ss        0:00 bash
    30435 pts/1    Ss+      0:00 bash
    30804 pts/0    S+        0:00 make test13
    30805 pts/0    S+        0:00 /bin/sh -
c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
    30806 pts/0    S+        0:00
/usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -
a -p
    30807 pts/0    S         0:00 ./tsh -p
    30809 pts/0    T         0:00 ./mysplit 4
    30810 pts/0    T         0:00 ./mysplit 4
    30813 pts/0    R         0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
      PID TTY          STAT       TIME
COMMAND
    1990 tty2      Ssl+    0:00
/usr/lib/gdm3/gdm-x-session --run-script env
GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --
session=ubuntu
    1992 tty2      Sl+      1:43
/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth
/run/user/1000/gdm/Xauthority -background
none -noreset -keeptty -verbose 3
    2413 tty2      Sl+      0:00

```

2413 tty2 SI+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 30104 pts/0 Ss+ 0:00 bash 30435 pts/1 Ss 0:00 bash 30817 pts/1 S+ 0:00 make rtest13 30818 pts/1 S+ 0:00 /bin/sh - c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 30819 pts/1 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt - s ./tshref -a -p 30820 pts/1 S+ 0:00 ./tshref -p 30829 pts/1 R 0:00 /bin/ps a	/usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 30104 pts/0 Ss 0:00 bash 30435 pts/1 Ss+ 0:00 bash 30804 pts/0 S+ 0:00 make test13 30805 pts/0 S+ 0:00 /bin/sh - c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 30806 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh - a -p 30807 pts/0 S 0:00 ./tsh -p 30816 pts/0 R 0:00 /bin/ps a
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
<pre> noname:shell>make test14 ./sdriver.pl -t trace14.txt -s ./tsh -a "-p" # # trace14.txt - Simple error handling # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 4 & [1] (30838) ./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] (30838) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (30838) ./myspin 4 & </pre>	<pre> noname:shell>make rtest14 ./sdriver.pl -t trace14.txt -s ./tshref -a "-p" # # trace14.txt - Simple error handling # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 4 & [1] (30857) ./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] (30857) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (30857) ./myspin 4 & </pre>

tsh> jobs [1] (30838) Running ./myspin 4 &	tsh> jobs [1] (30857) Running ./myspin 4 &
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
<pre> noname:shell>make test15 ./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] (30876) terminated by signal 2 tsh> ./myspin 3 & [1] (30878) ./myspin 3 & tsh> ./myspin 4 & [2] (30880) ./myspin 4 & tsh> jobs [1] (30878) Running ./myspin 3 & [2] (30880) Running ./myspin 4 & tsh> fg %1 Job [1] (30878) stopped by signal 20 tsh> jobs [1] (30878) Stopped ./myspin 3 & [2] (30880) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (30878) ./myspin 3 & tsh> jobs [1] (30878) Running ./myspin 3 & [2] (30880) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>	<pre> noname:shell>make rtest15 ./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] (30890) terminated by signal 2 tsh> ./myspin 3 & [1] (30897) ./myspin 3 & tsh> ./myspin 4 & [2] (30899) ./myspin 4 & tsh> jobs [1] (30897) Running ./myspin 3 & [2] (30899) Running ./myspin 4 & tsh> fg %1 Job [1] (30897) stopped by signal 20 tsh> jobs [1] (30897) Stopped ./myspin 3 & [2] (30899) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (30897) ./myspin 3 & tsh> jobs [1] (30897) Running ./myspin 3 & [2] (30899) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>
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：满分 15 分。

第 4 章 总结

4.1 请总结本次实验的收获

学会了使用系统调用 `fork`、`execve` 等。
学会了一些信号与信号处理、捕获信号的方法。
学会使用 `driver` 构建一个 `tinyshe`l。

4.2 请给出对本次实验内容的建议



无

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

-  (2条消息) SIGTSTP和SIGSTOP的区别_LevinLin的...
-  linux – BASH: 发送SIGTSTP信号(ctrl z) - 编程之家
-  (2条消息) linux挂起与恢复进程,Linux进程的优先...
-  setpgid () 函数 Unix/Linux - Unix/Linux系统调...
-  setpgid_百度百科
-  在用SIGTSTP暂停子进程之后, shell没有响应 - Vo..
-  处理SIGTSTP-jackywgw-ChinaUnix博客
-  (2条消息) SIGTSTP和SIGSTOP的区别_LevinLin的...
-  Linux 进程--父进程查询子进程的退出状态_12567...
-  (2条消息) 查看系统进程状态命令 (一) ——atop_..
-  (2条消息) linux清除信号处理函数,linux信号处理函..
-  (2条消息) 信号量SIGCHLD的使用, 如何让父进程...
-  虚拟机 (VMware) 如何设置共享文件夹-百度经验
-  百度翻译-200种语言互译、沟通全世界!
-  shell (计算机壳层) _百度百科
-  command.com_百度百科
-  进程 (一段程序的执行过程) _百度百科

-  [tinyshe ll sigtstp_百度搜索](#)
-  (2条消息) [ICS lab9 TinyShell 的10条建议_布客飞..](#)
-  (2条消息) [CSAPP-TinyShell 微壳_你踩你也麻的博..](#)
-  [kill, sigtstp中的sigtstp是什么意思_百度知道](#)
-  [第8章 异常控制流II-信号.pdf 和另外 7 个标签页](#)
-  [第8章 异常控制流II-信号.pdf](#)
-  [捕获SIGTSTP信号_百度搜索](#)
-  [CSAPP-shell lab实验记录 - 知乎](#)
-  [\[Linux Shell学习系列十三\]捕获-1.信号 - working..](#)
-  (2条消息) [scanf 捕获SIGTSTP - CSDN](#)
-  [处理SIGTSTP-jackywgw-ChinaUnix博客](#)
-  [如何等待所有的子进程结束? - Pig's home - CSD..](#)
-  (2条消息) [等待子进程结束wait\(\)和waitpid\(\)_cany..](#)