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 1190202107

班　　级 1936602

学　　生 姚舜宇

指 导 教 师 刘宏伟

实 验 地 点 G709

实 验 日 期 2021 6 3

计算机科学与技术学院

目 录

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

- 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；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

- Vmware 11；Ubuntu 18

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

- **Kill 命令**
- **进程状态**
- **ps t /ps aux /ps**
- 作业 ： jobs、 fg %n 、 bg%n

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）

概念：进程就是一个执行中程序的实例。是计算机中的程序关于某数据集合上的一次运行活动，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。

创建：父进程通过调用 fork 函数创建一个新的运行的子进程。新创建的子进程几乎但不完全与父进程相同。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，包括代码和数据段、堆、共享库以及用户栈。子进程还获得与父进程任何打开文件描述符相同的副本。父进程和新创建的子进程之间最大的区别在于它们有不同的 PID。

回收：当一个进程由于某种原因终止时，内核并不是立即把它从系统中清除。相反，进程被保持在一种已终止的状态中，知道 被它的父进程回收。一个进程可以通过调用 waitpid 函数来等待它的子进程终止或者停止。

2.2 信号的机制、种类（5 分）

信号的机制：一个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。没种信号都对应于某种系统事件，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，堆用户进程是不可见的。信号提供了一种机制，通知用户进程发生了这些异常。

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。发送信号可能的原因有两种，1，讷河检测到一个系统事件，比如除零错误或者子进程终止。2，一个进程调用了 kill 函数，显式地要求内核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做出反应时，它就接收了信号。进程可以忽略这个信号，终止或者通过执行一个称为信号处理程序的用户层函数捕获这个信号。

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	跟踪陷阱
6	SIGABRT	终止并转储内存	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	浮点异常
9	SIGKILL	终止	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	无效的内存引用
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	停止到下一个 SIGCONT	不是来自终端的停止信号
20	SIGTSTP	停止到下一个 SIGCONT	来自终端的停止信号
21	SIGTTIN	停止到下一个 SIGCONT	后台进程从终端读
22	SIGTTOU	停止到下一个 SIGCONT	后台进程从终端写

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	刨析定时器期满
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。默认情况下是终止前台作业。输入 Ctrl+Z 会发送一个 SIGTSTP 信号到前台进程组中的每个进程。默认情况下，结果是停止（挂起）前台作业。

3. 用 kill 函数发送信号

   进程可以通过调用 kill 函数发送信号给其他的进程（包括它们自己）。

4. 用 alarm 函数发送信号

   进程可以通过调用 alarm 函数向它自己发送 SIGALRM 信号。

阻塞方法：

显示阻塞机制：

内核默认阻塞任何当前处理程序终正在处理信号类型的待处理的信号。

隐式阻塞机制：

应用程序可以使用 sigprocmask 函数和它的辅助函数，明确地阻塞和接触阻塞选定的信号。sigprocmask 函数改变当前阻塞的信号集合。sigemptyset 初始化 set 为空集合。sigfillset 函数把每个信号都添加到 set 中。sigaddset 函数把 signum 添加到 set，sigdelset 从 set 中删除 signum。

处理程序的设置方法：

1. 安全的信号处理

   处理程序要尽可能简单。

   在处理程序中只调用以部信号安全的函数。

   保存和恢复 errno。

   阻塞所有的信号，保护对共享全局数据结构的访问。

   用 volatile 声明全局变量。

   用 sig_atomic_t 声明标志。

2. 正确的信号处理

   未处理的信号是不排队的，不可以用信号来对其他进程中发生的事件计数。

3. 可移植的信号处理

   signal 函数的语义各有不同，系统调用可以被中断。可以使用 sigaction 函数，它允许用户在设置信号处理时，明确指定它们想要的信号处理语义。

2.4 什么是 shell，功能和处理流程（5 分）

shell 的定义：shell 是一个交互型应用级程序，它代表用户运行其他程序。

功能：shell 最重要的功能是命令解释。shell 是一个命令解释器。用户提交了一个命令后，shell 首先判断它是否为内置命令，如果是就通过 shell 内部的解释器将其解释为系统功能调用并转交给内核执行；若是外部命令或使用程序就试图在硬盘中查找该命令并将其调入内存，再将其解释为系统功能调用并转交给内核执行。

shell 的处理流程：

shell 打印一个命令行提示符，等待用户在 stdin 上输入命令行，然后对这个命令行求值。命令行求值的首要任务是调用 parseline 函数，这个函数解析了以空格分割的命令行参数，并构造最终会传递给 execve 的 argv 向量。第一个参数被假设为要么是一个内置的 shell 命令名，马上就会解释这个命令，要么是一个可执行目标文件，会在一个新的子进程的上下文中加载并运行这个文件。

在解析了命令行之后，eval 函数调用 builtin_command 函数，该函数检查第一个命令行参数是否是一个内置的 shell 命令。如果是，它就会易理解释这个命令，并返回值 1。否则返回 0，shell 创建一个子进程，并在子进程中执行所请求的程序。如果用户要求在后台印象该程序，那么 shell 返回到循环的顶部，等待下一个命令行。否则，shell 使用 waitpid 函数等待作业终止。当作业终止时，shell 就回收子进程，并开始下一轮迭代。

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：执行用户输入的命令行。如果用户请求的是一个内置命令，则立即执行它。否则，fork 出一个子进程，并且在这个子进程的上下文中运行该作业。如果作业在前台运行，则等待其终止后返回。
参　　数：cmdline，用户输入的命令
处理流程：

1. 调用函数 parseline，解析以空格分隔的命令行参数，构造最终会传递给 execve 的 argv 向量，并返回 bg，是否在后台运行的标记。

```
bg = parseline(cmdline, argv);
```

2. 判断第一个命令行的第一个参数。如果为 null，则直接返回并退出。如果不是内置的命令，就继续执行，否则由其他函数负责执行内置命令。

```
if (argv[0] == NULL)
    return;     /* ignore empty lines */
```

3. 设置阻塞集合，并将信号 SIGCHLD,SIGINT,SIGTSTP 加入到集合内。
4. fork 一个子进程。在子进程内，解除信号阻塞，创建一个进程组，并通过 execve(argv[0],argv,environ)执行文件。
5. 将该作业添加到作业集合，解除阻塞。
6. 如果是前台作业，则等待其运行结束后返回。否则输出当前进程信息。

```
if (!bg)
    waitfg(pid);
else
    printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
```

要点分析：
阻塞信号的原因是防止在执行函数 addjob 时把不存在的子进程加入到作业集合中。
在调用各个系统函数时需要判断参数是否合法，检查函数是否被正常调用，并根据返回情况予以提示。

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

函数功能：检查第一个命令行参数是否是一个内置的 shell 命令。如果是，就立即解释这个命令，并返回 1.否则返回 0。
参　　数：argv，命令行字符串指针。

处理流程：

1. 判断是否为 quit 退出命令，如果是则直接 exit(0)，否则进入下一个判断。
2. 判断是否为&符号，忽略单独是符号&。
3. 判断是否为 fg 前台运行命令。如果是则将命令行指针传入函数 do_dgfg 进行处理，并且返回 1。
4. 判断是否为 bg 后台运行命令。如果是则将命令行指针传入函数 do_dgfg 进行处理，并且返回 1。
5. 判断是否为 jobs 列出作业命令。如果是则调用函数 listjobs 列出作业，并且返回 1。

要点分析：

如果第一个命令行参数为“fg”或“bg”，都是调用函数 do_bgfg，可以将这两个判断分支合并。

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

函数功能：对第一个命令行参数为“fg”或“bg”的命令行参数进行处理。

参　　数：argv，命令行字符串指针。

处理流程：

1. 判断第二个命令行参数。如果为 null，则输出提示信息并返回。否则继续执行。

```
if (argv[1] == NULL) {
    printf("%s command requires PID or %%jobid argument\n", argv[0]);
    return;
}
```

2. 如果第二个参数的第一个字符是数字，则以此为 pid 寻找作业。如果未找到则输出提示信息。

```
if (isdigit(argv[1][0])) {
    pid_t pid = atoi(argv[1]);
    if (!(jobp = getjobpid(jobs, pid))) {
        printf("(%d): No such process\n", pid);
        return;
    }
}
```

3. 如果第二个参数的第一个字符是“%”，则以第二个字符为 pid 寻找作业。如果未找到则输出提示信息。

```
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;
     }
}
```

4. 如果第二个参数既不是数字也不是“%”，则提示参数必须是一个 pid 或者%加作业 id，然后返回。

5. 判断是后台作业还是前台作业。如果是后台作业则发送信号 SIGCONT 给进程组 pid 的每一个进程，将作业的状态修改为后台，并输出作业信息。如果是前台作业则发送信号 SIGCONT 给进程组 pid 的每一个进程，将作业的状态修改为前台，并等待作业完成。如果参数既不是“bg”也不是“fg”，则输出错误提示信息后退出。

要点分析：

需要增强程序的健壮性，对于系统函数的返回值和参数进行检查，如有不合法处则进行提示。

获取进程 pid 或者作业 jid 的方式是根据第二个参数的第一个字符是数字还是“%”，然后再在获取的 id 上进行操作。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：等待前台进程 pid 结束。

参　　数：前台进程号 pid。

处理流程：

通过循环判断进程 pid 是否为前台进程，用 mask 替换当前的阻塞集合并挂起进程知道进程 pid 运行结束即不是前台进程。

要点分析：

在循环内部，有多种实现方式。可以使用 pause()函数，但这样如果收到一个或多个 SIGINT 信号，pause 会被中断。而且会有很严重的竞争条件，如果在 while 测试后和 pause 之前收到信号 SIGCHID，pause 会永远睡眠。也可以使用 sleep(1)函数，但这样做运行的速度会非常慢，如果在 while 之后 sleep 之前收到信号，程序必须等相当一段时间才会再次检查循环的终止条件。间隔太小，循环会太浪费。间隔太大，程序又会太慢。所以使用了 sigsuspend。sigsuspend 函数暂时用 mask 替换当前的阻塞集合，然后挂起该进程，直到收到一个信号，其行为要么是运行一个处理程序，要么是终止该进程。如果它的行为是终止，那么该进程不从 sigsuspend 返回就直接终止。如果它的行为是运行一个处理程序，那么 sigsuspend 从处理程序返回，恢复调用 sigsuspend 时原有的阻塞集合。

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

函数功能：当子进程终止或因接收到信号 SIGSTOP 或 SIGTSTP 时而停止时，内核会向 shell 发送信号 SIGCHLD。这个函数需要回收所有可以回收的僵死子进程。

参　　数：sig，即子进程发出的 SIGCHLD 信号。

处理流程：

1. 保存错误状态，并把每个信号都添加到阻塞集合里。

```
int olderrno=errno;   /* 保存错误状态 */
int status;
sigset_t mask,prev_mask;
if (sigfillset(&mask)<0) {     /* 把每个信号都添加到集合中 */
     unix_error("sigfillset error");
```

```
}
```

2. 循环判断获取子进程的 pid。
3. 在循环体内，临时阻塞接受信号，由子进程的 pid 获取当前的作业指针。

```
if (sigprocmask(SIG_BLOCK,&mask,&prev_mask)<0) {    /* 临时阻塞接受信号 */
    unix_error("sigprocmask error");
}
struct job_t * thisjob=getjobpid(jobs,pid);
```

4. 判断，如果子进程通过调用 exit 或者一个返回正常终止，则回收终止的进程。
5. 判断，如果引起返回的子进程当前是停止的，则将当前作业的状态修改为停止，并且输出作业停止的提示信息。
6. 判断，如果子进程是因为一个未被捕获的信号终止，则输出作业终止的提示信息，并且回收终止的进程。
7. 解除阻塞接收信号，继续下一轮循环。
8. 循环退出后，恢复错误状态。

要点分析：

子进程停止或终止的原因可能有多种，每种原因的处理方式都可能有一些差别，所以需要分为几类来处理。一，子进程是通过调用 exit 或者一个返回正常终止。二，引起返回的子进程当前是停止的。三，子进程是因为一个未被捕获的信号终止。

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

```
/*
 * tsh - A tiny shell program with job control
 * 姚舜宇 1190202107
 * <Put your name and login ID here>
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

/* Misc manifest constants */
#define MAXLINE    1024   /* max line size */
#define MAXARGS     128   /* max args on a command line */
#define MAXJOBS      16   /* max jobs at any point in time */
```

```
#define MAXJID      1<<16   /* max job ID */

/* Job states */
#define UNDEF 0 /* undefined */
#define FG 1    /* running in foreground */
#define BG 2    /* running in background */
#define ST 3    /* stopped */

/*
 * Jobs states: FG (foreground), BG (background), ST (stopped)
 * Job state transitions and enabling actions:
 *      FG -> ST  : ctrl-z
 *      ST -> FG  : fg command
 *      ST -> BG  : bg command
 *      BG -> FG  : fg command
 * At most 1 job can be in the FG state.
 */

/* Global variables */
extern char **environ;          /* defined in libc */
char prompt[] = "tsh> ";        /* command line prompt (DO NOT CHANGE) */
int verbose = 0;                /* if true, print additional output */
int nextjid = 1;                /* next job ID to allocate */
char sbuf[MAXLINE];             /* for composing sprintf messages */

struct job_t {                  /* The job struct */
    pid_t pid;                  /* job PID */
    int jid;                    /* job ID [1, 2, ...] */
    int state;                  /* UNDEF, BG, FG, or ST */
    char cmdline[MAXLINE];      /* command line */
};
struct job_t jobs[MAXJOBS]; /* The job list */
/* End global variables */


/* Function prototypes */

/* Here are the functions that you will implement */
void eval(char *cmdline);
int builtin_cmd(char **argv);
void do_bgfg(char **argv);
void waitfg(pid_t pid);

void sigchld_handler(int sig);
void sigtstp_handler(int sig);
void sigint_handler(int sig);
```

```
/* Here are helper routines that we've provided for you */
int parseline(const char *cmdline, char **argv);
void sigquit_handler(int sig);

void clearjob(struct job_t *job);
void initjobs(struct job_t *jobs);
int maxjid(struct job_t *jobs);
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);
int deletejob(struct job_t *jobs, pid_t pid);
pid_t fgpid(struct job_t *jobs);
struct job_t *getjobpid(struct job_t *jobs, pid_t pid);
struct job_t *getjobjid(struct job_t *jobs, int jid);
int pid2jid(pid_t pid);
void listjobs(struct job_t *jobs);

void usage(void);
void unix_error(char *msg);
void app_error(char *msg);
typedef void handler_t(int);
handler_t *Signal(int signum, handler_t *handler);

/*
 * main - The shell's main routine
 */
int main(int argc, char **argv)
{
    char c;
    char cmdline[MAXLINE];
    int emit_prompt = 1; /* emit prompt (default) */

    /* Redirect stderr to stdout (so that driver will get all output
     * on the pipe connected to stdout) */
    dup2(1, 2);

    /* Parse the command line */
    while ((c = getopt(argc, argv, "hvp")) != EOF) {
        switch (c) {
        case 'h':             /* print help message */
            usage();
            break;
        case 'v':             /* emit additional diagnostic info */
            verbose = 1;
            break;
        case 'p':             /* don't print a prompt */
            emit_prompt = 0;  /* handy for automatic testing */
            break;
    default:
```

```
                usage();
        }
        }

        /* Install the signal handlers */

        /* These are the ones you will need to implement */
        Signal(SIGINT,  sigint_handler);   /* ctrl-c */
        Signal(SIGTSTP, sigtstp_handler);  /* ctrl-z */
        Signal(SIGCHLD, sigchld_handler);  /* Terminated or stopped child */

        /* This one provides a clean way to kill the shell */
        Signal(SIGQUIT, sigquit_handler);

        /* Initialize the job list */
        initjobs(jobs);

        /* Execute the shell's read/eval loop */
        while (1) {

       /* Read command line */
       if (emit_prompt) {
              printf("%s", prompt);
              fflush(stdout);
       }
       if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
              app_error("fgets error");
       if (feof(stdin)) { /* End of file (ctrl-d) */
              fflush(stdout);
              exit(0);
       }

       /* Evaluate the command line */
       eval(cmdline);
       fflush(stdout);
       fflush(stdout);
        }

        exit(0); /* control never reaches here */
}

/*
  * eval - Evaluate the command line that the user has just typed in
  *
  * If the user has requested a built-in command (quit, jobs, bg or fg)
  * then execute it immediately. Otherwise, fork a child process and
  * run the job in the context of the child. If the job is running in
```

```
 * the foreground, wait for it to terminate and then return.    Note:
 * each child process must have a unique process group ID so that our
 * background children don't receive SIGINT (SIGTSTP) from the kernel
 * when we type ctrl-c (ctrl-z) at the keyboard.
*/
void eval(char *cmdline)
{
      /* $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 */

      if (!builtin_cmd(argv)) {

            /*
       * This is a little tricky. Block SIGCHLD, SIGINT, and SIGTSTP
       * signals until we can add the job to the job list. This
       * eliminates some nasty races between adding a job to the job
       * list and the arrival of SIGCHLD, SIGINT, and SIGTSTP signals.
       */

          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");

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

          /*
           * Child    process
           */

          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);
            }
        }

        /*
         * Parent process
         */

        /* 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;
}

/*
 * parseline - Parse the command line and build the argv array.
 *
 * Characters enclosed in single quotes are treated as a single
 * argument.  Return true if the user has requested a BG job, false if
 * the user has requested a FG job.
 */
int parseline(const char *cmdline, char **argv)
{
    static char array[MAXLINE]; /* holds local copy of command line */
    char *buf = array;          /* ptr that traverses command line */
    char *delim;                /* points to first space delimiter */
    int argc;                   /* number of args */
```

```
    int bg;                                     /* background job? */

    strcpy(buf, cmdline);
    buf[strlen(buf)-1] = ' ';    /* replace trailing '\n' with space */
    while (*buf && (*buf == ' ')) /* ignore leading spaces */
    buf++;

    /* Build the argv list */
    argc = 0;
    if (*buf == '\'') {
    buf++;
    delim = strchr(buf, '\'');
    }
    else {
    delim = strchr(buf, ' ');
    }

    while (delim) {
    argv[argc++] = buf;
    *delim = '\0';
    buf = delim + 1;
    while (*buf && (*buf == ' ')) /* ignore spaces */
            buf++;

    if (*buf == '\'') {
        buf++;
        delim = strchr(buf, '\'');
    }
    else {
        delim = strchr(buf, ' ');
    }
    }
    argv[argc] = NULL;

    if (argc == 0)    /* ignore blank line */
    return 1;

    /* should the job run in the background? */
    if ((bg = (*argv[argc-1] == '&')) != 0) {
    argv[--argc] = NULL;
    }
    return bg;
}

/*
 * builtin_cmd - If the user has typed a built-in command then execute
 *       it immediately.
```

```
 */
int builtin_cmd(char **argv)
{
    if (!strcmp(argv[0],"quit"))    /* quit command */
        exit(0);
    if (!strcmp(argv[0],"&"))    /* Ignore singleton & */
        return 1;
    if (!strcmp(argv[0],"fg")) {    /* foreground process command */
        do_bgfg(argv);                    /* call function do_bgfg */
        return 1;
    }
    if (!strcmp(argv[0],"bg")) {    /* background process command */
        do_bgfg(argv);                    /* call function do_bgfg */
        return 1;
    }
    if (!strcmp(argv[0],"jobs")) {    /* list jobs command */
        listjobs(jobs);                   /* call function listjobs to list the jobs */
        return 1;
    }
     return 0;         /* not a builtin command */
}

/*
 * do_bgfg - Execute the builtin bg and fg commands
 */
void do_bgfg(char **argv)
{
    /* $begin handout */
    struct job_t *jobp=NULL;

    /* 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;
        }

        /* bg command */
        if (!strcmp(argv[0], "bg")) {
            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")) {
            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 */
        return;
}

/*
 * waitfg - Block until process pid is no longer the foreground process
 */
void waitfg(pid_t pid)
{
    sigset_t mask;
    if (sigemptyset(&mask)<0) {       /* 清空 block */
        unix_error("sigemptyset error");
    }
    while(pid == fgpid(jobs)) {
        sigsuspend(&mask);   /* 用 mask 替换当前的阻塞集合，然后挂起该进程 */
    }
    return;
}
```

```
/*****************
 * Signal handlers
 *****************/

/*
 * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
 *         a child job terminates (becomes a zombie), or stops because it
 *         received a SIGSTOP or SIGTSTP signal. The handler reaps all
 *         available zombie children, but doesn't wait for any other
 *         currently running children to terminate.
 */
void sigchld_handler(int sig)
{
    int olderrno=errno;   /* 保存错误状态 */
    int status;
    sigset_t mask,prev_mask;
    if (sigfillset(&mask)<0) {     /* 把每个信号都添加到集合中 */
        unix_error("sigfillset error");
    }
    pid_t pid;
    while ((pid=waitpid(-1,&status,WNOHANG | WUNTRACED))>0) {   /* 循环回收子进程 */
        if (sigprocmask(SIG_BLOCK,&mask,&prev_mask)<0) {   /* 临时阻塞接受信号 */
            unix_error("sigprocmask error");
        }
        struct job_t * thisjob=getjobpid(jobs,pid);
        if (WIFEXITED(status)) {   /* 子进程通过调用 exit 或者一个返回正常终止 */
            deletejob(jobs,pid);   /* 回收终止的进程 */
        }
        if (WIFSTOPPED(status)) {   /* 引起返回的子进程当前是停止的 */
            thisjob->state=ST;       /* 将当前作业的状态修改为停止 */
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, WSTOPSIG(status));
        }
        if (WIFSIGNALED(status)) {   /* 子进程是因为一个未被捕获的信号终止 */
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(status));
            deletejob(jobs,pid);   /* 回收终止的进程 */
        }
        if (sigprocmask(SIG_SETMASK,&prev_mask,NULL)<0) {   /* 解除阻塞接受信号 */
```

```
            unix_error("sigprocmask error");
        }
    }
    errno=olderrno;   /* 恢复错误状态 */
     return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *      user types ctrl-c at the keyboard.    Catch it and send it along
 *      to the foreground job.
 */
void sigint_handler(int sig)
{
    int olderrno=errno;   /* 保存错误状态 */
    sigset_t mask,prev_mask;
    if (sigfillset(&mask)<0) {     /* 把每个信号都添加到集合中 */
        unix_error("sigfillset error");
    }
    if (sigaddset(&mask,sig)<0) {   /* 将信号 SIGTSTP 加入集合 */
        unix_error("sigaddset error");
    }
    if (sigprocmask(SIG_BLOCK,&mask,&prev_mask)<0) {   /* 临时阻塞接受信号
*/
        unix_error("sigprocmask error");
    }
    pid_t fg_pid=fgpid(jobs);   /* 获取前台作业 pid */
    if (sigprocmask(SIG_SETMASK,&prev_mask,NULL)<0) {   /* 解除阻塞接受信
号 */
        unix_error("sigprocmask error");
    }
    if (fg_pid!=0) {
        kill(-fg_pid,SIGINT);   /* 将信号 SIGINT 传递给子进程 */
    }
    errno=olderrno;   /* 恢复错误状态 */
     return;
}

/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *      the user types ctrl-z at the keyboard. Catch it and suspend the
 *      foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig)
{
```

```
    int olderrno=errno;   /* 保存错误状态 */
    sigset_t mask,prev_mask;
    if (sigfillset(&mask)<0) {    /* 把每个信号都添加到集合中 */
        unix_error("sigfillset error");
    }
    if (sigaddset(&mask,sig)<0) {   /* 将信号 SIGTSTP 加入集合 */
        unix_error("sigaddset error");
    }
    if (sigprocmask(SIG_BLOCK,&mask,&prev_mask)<0) {   /* 临时阻塞接受信号
*/
        unix_error("sigprocmask error");
    }
    pid_t fg_pid=fgpid(jobs);   /* 获取前台作业 pid */
    if (sigprocmask(SIG_SETMASK,&prev_mask,NULL)<0) {   /* 解除阻塞接受信
号 */
        unix_error("sigprocmask error");
    }
    if (fg_pid!=0) {
        kill(-fg_pid,SIGTSTP);   /* 将信号 SIGTSTP 传递给子进程 */
    }
    errno=olderrno;   /* 恢复错误状态 */
     return;
}

/*********************
 * End signal handlers
 *********************/

/***********************************************
 * Helper routines that manipulate the job list
 **********************************************/

/* clearjob - Clear the entries in a job struct */
void clearjob(struct job_t *job) {
     job->pid = 0;
     job->jid = 0;
     job->state = UNDEF;
     job->cmdline[0] = '\0';
}

/* initjobs - Initialize the job list */
void initjobs(struct job_t *jobs) {
     int i;

     for (i = 0; i < MAXJOBS; i++)
    clearjob(&jobs[i]);
```

```
}

/* maxjid - Returns largest allocated job ID */
int maxjid(struct job_t *jobs)
{
       int i, max=0;

       for (i = 0; i < MAXJOBS; i++)
      if (jobs[i].jid > max)
            max = jobs[i].jid;
       return max;
}

/* addjob - Add a job to the job list */
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline)
{
       int i;

       if (pid < 1)
      return 0;

       for (i = 0; i < MAXJOBS; i++) {
      if (jobs[i].pid == 0) {
            jobs[i].pid = pid;
            jobs[i].state = state;
            jobs[i].jid = nextjid++;
            if (nextjid > MAXJOBS)
           nextjid = 1;
            strcpy(jobs[i].cmdline, cmdline);
            if(verbose){
                  printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].pid,
jobs[i].cmdline);
                   }
                   return 1;
      }
       }
       printf("Tried to create too many jobs\n");
       return 0;
}

/* deletejob - Delete a job whose PID=pid from the job list */
int deletejob(struct job_t *jobs, pid_t pid)
{
       int i;

       if (pid < 1)
      return 0;
```

```
    for (i = 0; i < MAXJOBS; i++) {
    if (jobs[i].pid == pid) {
        clearjob(&jobs[i]);
        nextjid = maxjid(jobs)+1;
        return 1;
    }
    }
    return 0;
}

/* fgpid - Return PID of current foreground job, 0 if no such job */
pid_t fgpid(struct job_t *jobs) {
    int i;

    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].state == FG)
        return jobs[i].pid;
    return 0;
}

/* getjobpid    - Find a job (by PID) on the job list */
struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
    int i;

    if (pid < 1)
    return NULL;
    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].pid == pid)
        return &jobs[i];
    return NULL;
}

/* getjobjid    - Find a job (by JID) on the job list */
struct job_t *getjobjid(struct job_t *jobs, int jid)
{
    int i;

    if (jid < 1)
    return NULL;
    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].jid == jid)
        return &jobs[i];
    return NULL;
}

/* pid2jid - Map process ID to job ID */
```

```
int pid2jid(pid_t pid)
{
       int i;

       if (pid < 1)
      return 0;
       for (i = 0; i < MAXJOBS; i++)
      if (jobs[i].pid == pid) {
                    return jobs[i].jid;
             }
       return 0;
}

/* listjobs - Print the job list */
void listjobs(struct job_t *jobs)
{
       int i;

       for (i = 0; i < MAXJOBS; i++) {
      if (jobs[i].pid != 0) {
             printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
             switch (jobs[i].state) {
            case BG:
                   printf("Running ");
                   break;
            case FG:
                   printf("Foreground ");
                   break;
            case ST:
                   printf("Stopped ");
                   break;
             default:
                   printf("listjobs: Internal error: job[%d].state=%d ",
                        i, jobs[i].state);
             }
             printf("%s", jobs[i].cmdline);
      }
       }
}
/******************************
 * end job list helper routines
 ******************************/


/***********************
 * Other helper routines
 ***********************/
```

```
/*
 * usage - print a help message
 */
void usage(void)
{
    printf("Usage: shell [-hvp]\n");
    printf("   -h   print this message\n");
    printf("   -v   print additional diagnostic information\n");
    printf("   -p   do not emit a command prompt\n");
    exit(1);
}

/*
 * unix_error - unix-style error routine
 */
void unix_error(char *msg)
{
    fprintf(stdout, "%s: %s\n", msg, strerror(errno));
    exit(1);
}

/*
 * app_error - application-style error routine
 */
void app_error(char *msg)
{
    fprintf(stdout, "%s\n", msg);
    exit(1);
}

/*
 * Signal - wrapper for the sigaction function
 */
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);
}
```

```
/*
 * sigquit_handler - The driver program can gracefully terminate the
 *       child shell by sending it a SIGQUIT signal.
 */
void sigquit_handler(int sig)
{
    printf("Terminating after receipt of SIGQUIT signal\n");
    exit(1);
}
```

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

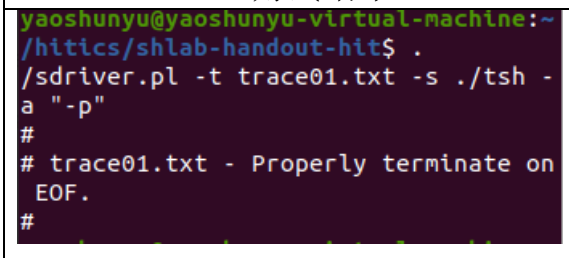
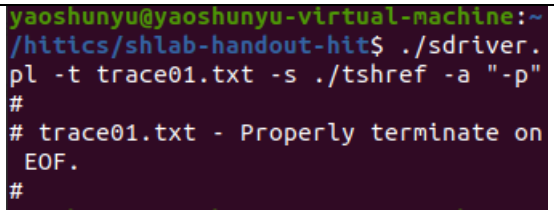
（2)测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令，每次运行的输出都会不同，但每个 mysplit 进程的运行状态应该相同。

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。

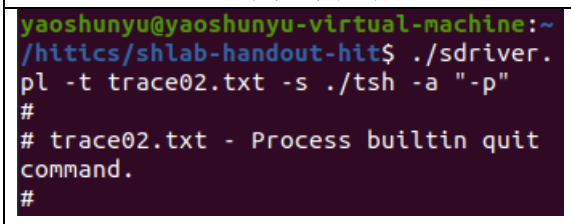
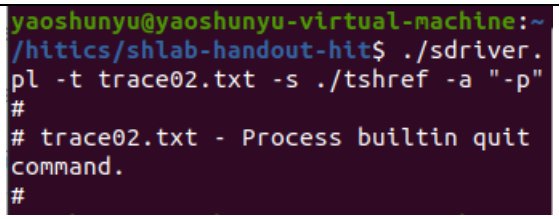
4.3 自测试结果

填写以下各个测试用例的测试结果，每个测试用例 1 分。

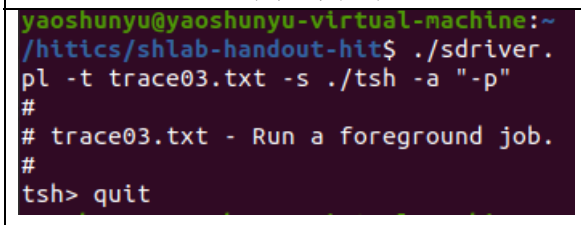
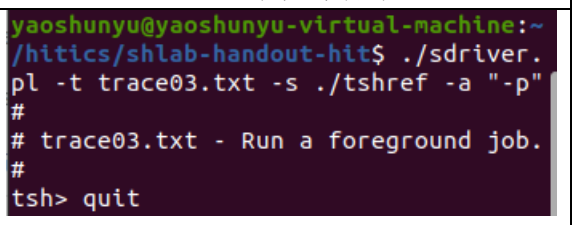
4.3.1 测试用例 trace01.txt

<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/不同，原因分析如下：相同</td></tr>
</table>

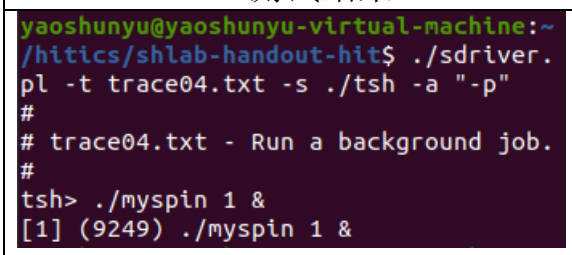
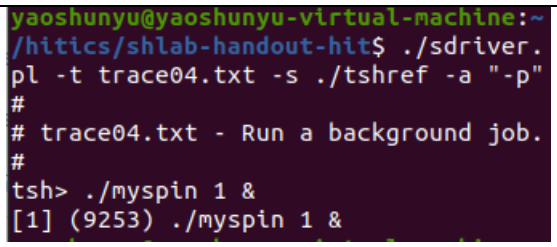
4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
	
测试结论	相同/不同，原因分析如下：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/不同，原因分析如下：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/不同，原因分析如下：相同

4.3.5 测试用例 trace05.txt

tsh 测试结果	tshref 测试结果

yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (9257) ./myspin 2 & tsh> ./myspin 3 & [2] (9259) ./myspin 3 & tsh> jobs [1] (9257) Running ./myspin 2 & [2] (9259) Running ./myspin 3 &	yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (9264) ./myspin 2 & tsh> ./myspin 3 & [2] (9266) ./myspin 3 & tsh> jobs [1] (9264) Running ./myspin 2 & [2] (9266) Running ./myspin 3 &
测试结论	相同/不同，原因分析如下：相同

4.3.6 测试用例 trace06.txt

tsh 测试结果	tshref 测试结果
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (13109) terminated by signal 2	yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (13105) terminated by signal 2
测试结论	相同/不同，原因分析如下：相同

4.3.7 测试用例 trace07.txt

tsh 测试结果	tshref 测试结果
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (9287) ./myspin 4 & tsh> ./myspin 5 Job [0] (9289) terminated by signal 2 tsh> jobs [1] (9287) Running ./myspin 4 &	yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (9294) ./myspin 4 & tsh> ./myspin 5 Job [2] (9296) terminated by signal 2 tsh> jobs [1] (9294) Running ./myspin 4 &
测试结论	相同/不同，原因分析如下：相同

4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果

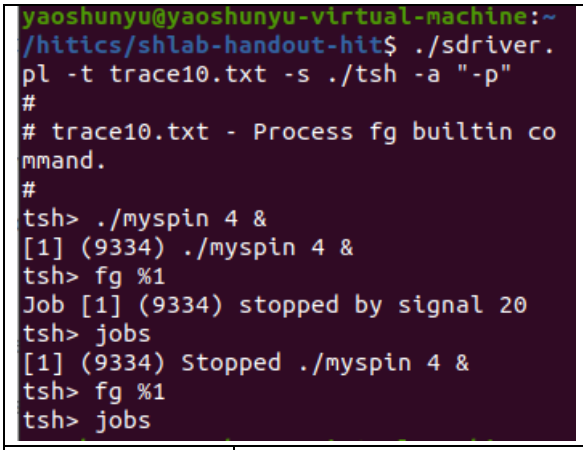
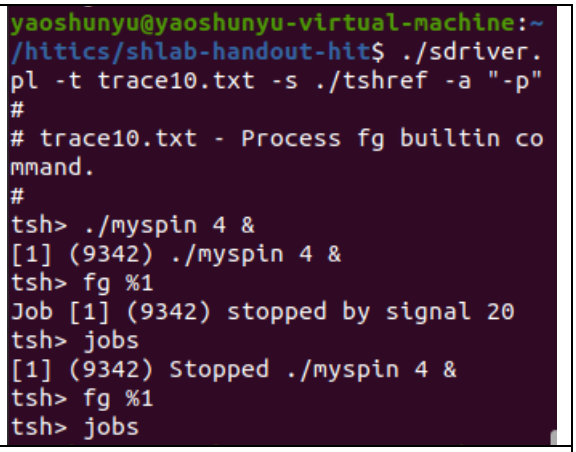
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (9302) ./myspin 4 & tsh> ./myspin 5 Job [2] (9304) stopped by signal 20 tsh> jobs [1] (9302) Running ./myspin 4 & [2] (9304) Stopped ./myspin 5	yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (9309) ./myspin 4 & tsh> ./myspin 5 Job [2] (9311) stopped by signal 20 tsh> jobs [1] (9309) Running ./myspin 4 & [2] (9311) Stopped ./myspin 5
测试结论	相同/不同，原因分析如下：相同

4.3.9 测试用例 trace09.txt

tsh 测试结果	tshref 测试结果
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (9316) ./myspin 4 & tsh> ./myspin 5 Job [2] (9318) stopped by signal 20 tsh> jobs [1] (9316) Running ./myspin 4 & [2] (9318) Stopped ./myspin 5 tsh> bg %2 [2] (9318) ./myspin 5 tsh> jobs [1] (9316) Running ./myspin 4 & [2] (9318) Running ./myspin 5	yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (9325) ./myspin 4 & tsh> ./myspin 5 Job [2] (9327) stopped by signal 20 tsh> jobs [1] (9325) Running ./myspin 4 & [2] (9327) Stopped ./myspin 5 tsh> bg %2 [2] (9327) ./myspin 5 tsh> jobs [1] (9325) Running ./myspin 4 & [2] (9327) Running ./myspin 5
测试结论	相同/不同，原因分析如下：相同

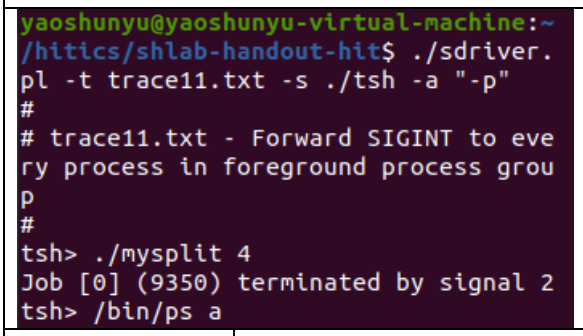
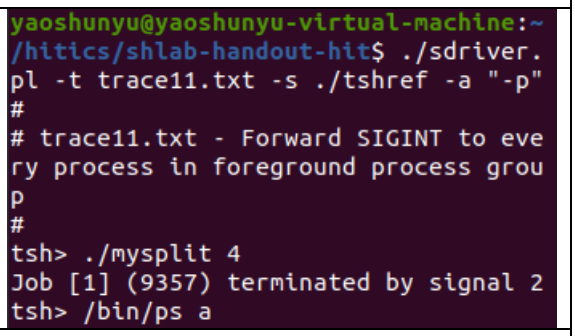
4.3.10 测试用例 trace10.txt

tsh 测试结果	tshref 测试结果

	
测试结论	相同/不同，原因分析如下：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/不同，原因分析如下：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

```
yaoshunyu@yaoshunyu-virtual-machine:~
/hitics/shlab-handout-hit$ ./sdriver.
pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to ev
ery process in foreground process gro
up
#
tsh> ./mysplit 4
Job [1] (9368) stopped by signal 20
tsh> jobs
[1] (9368) Stopped ./mysplit 4
tsh> /bin/ps a
```

```
yaoshunyu@yaoshunyu-virtual-machine:~
/hitics/shlab-handout-hit$ ./sdriver.
pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to ev
ery process in foreground process gro
up
#
tsh> ./mysplit 4
Job [1] (9376) stopped by signal 20
tsh> jobs
[1] (9376) Stopped ./mysplit 4
tsh> /bin/ps a
```

测试结论	相同/不同，原因分析如下：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

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/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] (9385) stopped by signal 20
tsh> jobs
[1] (9385) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1668 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
   1672 tty2     Sl+    0:40 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /ru
n/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1716 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd -
-systemd --session=ubuntu
   1791 tty2     Z+     0:00 [fcitx] <defunct>
   9209 pts/0    Ss     0:00 bash
   9382 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tsh -a -p
   9383 pts/0    S+     0:00 ./tsh -p
   9385 pts/0    T      0:00 ./mysplit 4
   9386 pts/0    T      0:00 ./mysplit 4
   9389 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1668 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
   1672 tty2     Sl+    0:41 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /ru
n/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1716 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd -
-systemd --session=ubuntu
   1791 tty2     Z+     0:00 [fcitx] <defunct>
   9209 pts/0    Ss     0:00 bash
   9382 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tsh -a -p
   9383 pts/0    S+     0:00 ./tsh -p
   9392 pts/0    R      0:00 /bin/ps a
```

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/shlab-handout-hit$ ./sdriver.
pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (9397) stopped by signal 20
tsh> jobs
[1] (9397) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1668 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
   1672 tty2     Sl+    0:42 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /ru
n/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1716 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd -
-systemd --session=ubuntu
   1791 tty2     Z+     0:00 [fcitx] <defunct>
   9209 pts/0    Ss     0:00 bash
   9394 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
   9395 pts/0    S+     0:00 ./tshref -p
   9397 pts/0    T      0:00 ./mysplit 4
   9398 pts/0    T      0:00 ./mysplit 4
   9401 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1668 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
   1672 tty2     Sl+    0:42 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /ru
n/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1716 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd -
-systemd --session=ubuntu
   1791 tty2     Z+     0:00 [fcitx] <defunct>
   9209 pts/0    Ss     0:00 bash
   9394 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s
./tshref -a -p
   9395 pts/0    S+     0:00 ./tshref -p
   9404 pts/0    R      0:00 /bin/ps a
```

测试结论	相同/不同，原因分析如下：相同

4.3.14 测试用例 trace14.txt

tsh 测试结果	tshref 测试结果

<table>
<tr>
<td>

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/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] (9418) ./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] (9418) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (9418) ./myspin 4 &
tsh> jobs
[1] (9418) Running ./myspin 4 &
```

</td>
<td>

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/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] (9435) ./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] (9435) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (9435) ./myspin 4 &
tsh> jobs
[1] (9435) Running ./myspin 4 &
```

</td>
</tr>
<tr><td>测试结论</td><td colspan="2">相同/不同，原因分析如下：相同</td></tr>
</table>

4.3.15 测试用例 trace15.txt

tsh 测试结果	tshref 测试结果

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/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 [0] (9453) terminated by signal 2
tsh> ./myspin 3 &
[1] (9455) ./myspin 3 &
tsh> ./myspin 4 &
[2] (9457) ./myspin 4 &
tsh> jobs
[1] (9455) Running ./myspin 3 &
[2] (9457) Running ./myspin 4 &
tsh> fg %1
Job [1] (9455) stopped by signal 20
tsh> jobs
[1] (9455) Stopped ./myspin 3 &
[2] (9457) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (9455) ./myspin 3 &
tsh> jobs
[1] (9455) Running ./myspin 3 &
[2] (9457) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
yaoshunyu@yaoshunyu-virtual-machine:~/hitics/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] (9471) terminated by signal 2
tsh> ./myspin 3 &
[1] (9473) ./myspin 3 &
tsh> ./myspin 4 &
[2] (9475) ./myspin 4 &
tsh> jobs
[1] (9473) Running ./myspin 3 &
[2] (9475) Running ./myspin 4 &
tsh> fg %1
Job [1] (9473) stopped by signal 20
tsh> jobs
[1] (9473) Stopped ./myspin 3 &
[2] (9475) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (9473) ./myspin 3 &
tsh> jobs
[1] (9473) Running ./myspin 3 &
[2] (9475) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

测试结论	相同/不同，原因分析如下：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

了解了 shell 的工作原理和流程。对信号这一概念以及信号处理函数更加了解了。

5.2 请给出对本次实验内容的建议

阻塞那一部分希望能够多给一些材料。

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.