哈爾濱工業大學

实验报告

实 验（七）

题　　目 TinyShell

微壳

专　　业 计算机类

学　　号 1190200523

班　　级 1903002

学　　生 石翔宇

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.6.4

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

- Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
- 16GB RAM
- 1TB HDD + 512G SSD

1.2.2 软件环境

- Windows 10 21H1
- Ubuntu 20.04 LTS

1.2.3 开发工具

- VSCode，CodeBlocks，gcc+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）

概念：

进程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。程序是指令、数据及其组织形式的描述，进程是程序的实体。

创建方法：

父进程通过调用 fork 函数创建一个新的运行的子进程。子进程中，fork 返回 0；父进程中，返回子进程的 PID。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虚拟地址。空间相同的但是独立的一份副本，子进程获得与父进程任何打开文件描述符相同的副本，最大区别是子进程有不同于父进程的 PID。

回收方法：

父进程通过 wait 函数回收子进程，wait 函数挂起当前进程的执行直到它的一个子进程终止，返回已终止子进程的 PID。

2.2 信号的机制、种类（5 分）

机制：

信号是用来通知进程发生了异步事件，是在软件层次上是对中断机制的一种模拟，在原理上，一个进程收到一个信号与处理器收到一个中断请求可以说是一样的。信号是进程间通信机制中唯一的异步通信机制，一个进程不必通过任何操作来等待信号的到达，事实上，进程也不知道信号到底什么时候到达。进程之间可以互相通过系统调用 kill 发送软中断信号，内核也可以因为内部事件而给进程发送信号，通知进程发生了某个事件。

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①	跟踪陷阱
6	SIGABRT	终止并转储内存①	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①	浮点异常
9	SIGKILL	终止②	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①	无效的内存引用（段故障）
12	SIGUSR2	终止	用户定义的信号 2
13	SIGPIPE	终止	向一个没有读用户的管道做写操作
14	SIGALRM	终止	来自 alarm 函数的定时器信号
15	SIGTERM	终止	软件终止信号
16	SIGSTKFLT	终止	协处理器上的栈故障
17	SIGCHLD	忽略	一个子进程停止或者终止
18	SIGCONT	忽略	继续进程如果该进程停止
19	SIGSTOP	停止直到下一个 SIGCONT②	不是来自终端的停止信号
20	SIGTSTP	停止直到下一个 SIGCONT	来自终端的停止信号
21	SIGTTIN	停止直到下一个 SIGCONT	后台进程从终端读
22	SIGTTOU	停止直到下一个 SIGCONT	后台进程向终端写
23	SIGURG	忽略	套接字上的紧急情况
24	SIGXCPU	终止	CPU 时间限制超出
25	SIGXFSZ	终止	文件大小限制超出
26	SIGVTALRM	终止	虚拟定时器期满
27	SIGPROF	终止	剖析定时器期满
28	SIGWINCH	忽略	窗口大小变化
29	SIGIO	终止	在某个描述符上可执行 I/O 操作
30	SIGPWR	终止	电源故障

图 8-26　Linux 信号

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 程序可以向另外的进程或进程组发送任意的信号。
2. 从键盘发送信号输入 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. 还可以调用系统函数来发送信号，主要有 kill()，raise()，alarm()，pause()。

阻塞方法：

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如：一个 SIGINT 信号处理程序不能被另一个 SIGINT 信号中断 （此时另一个 SIGINT 信号被阻塞）

显式阻塞机制：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。

设置信号处理程序：

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)

2.4 什么是 shell，功能和处理流程（5 分）

概念：

Shell 是一种命令行解释器，其读取用户输入的字符串命令，解释并且执行命令。 它是一种特殊的应用程序，介于系统调用/库与应用程序之间，其提供了运行其他程序的的接口。它可以是交互式的，即读取用户输入的字符串;也可以是非交互式的，即读取脚本文件并解释执行，直至文件结束。 无论是在类 UNIX，Linux 系统，还是 Windows，有很多不同种类的 Shell: 如类 UNIX，Linux 系统上的 Bash，Zsh 等; Windows 系统上的 cmd，PowerShell 等。

功能：

Shell 为用户提供了启动程序、管理文件系统中的文件和运行在 Linux 系统上的进程。

处理流程：

1. Shell 首先从命令行中找出特殊字符（元字符），在将元字符翻译成间隔符号。
2. 程序块 tokens 被处理，检查看他们是否是 shell 中所引用到的关键字。
3. 当程序块 tokens 被确定以后，shell 根据 aliases 文件中的列表来检查命令的第一个单词。如果这个单词出现在 aliases 表中，执行替换操作并且处理过程回到第一步重新分割程序块 tokens。
4. Shell 对~符号进行替换。
5. Shell 对所有前面带有$符号的变量进行替换。
6. Shell 将命令行中的内嵌命令表达式替换成命令；他们一般都采用$(command)标记法。
7. Shell 计算采用$(expression)标记的算术表达式。
8. Shell 将命令字符串重新划分为新的块 tokens。
9. Shell 执行通配符* ? [ ]的替换。
10. Shell 把所有处理的结果中用到的注释删除，并且按照下面的顺序实行命令的检查：
    (1) 内建的命令
    (2) shell 函数（由用户自己定义的）
    (3) 可执行的脚本文件（需要寻找文件和 PATH 路径）
11. 初始化所有的输入输出重定向。

12. 执行命令。

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：

解析传入的命令行命令。

参　　数：

char *cmdline

处理流程：

1. 调用 parseline 函数进行命令参数的解析。
2. 判断命令是否为内置命令
   a) 若是内置命令则在 builtin_cmd 处理。
   b) 若不是内置命令，则：
      i. 阻塞 SIGCHLD, SIGINT 和 SIGTSTP 信号。
      ii. fork 一个新的进程。
      iii. 在父进程中将子进程添加到 jobs 里面，解除阻塞。若该命令是在后台运行的，则输出提示信息；否则调用 waitfg 函数等待命令完成。
      iv. 在子进程中解除阻塞，获取一个新的组 ID。调用 execve 函数执行命令。

要点分析：

1. 在流程第二步开始前判断命令是否存在，若不存在则返回。
2. 若不是内置命令，则在 addjob 前要阻塞 SIGCHLD, SIGINT 和 SIGTSTP 信号，防止还没有将新进程添加到jobs 列表里面就收到上述信号，出现错误。

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

函数功能：

处理内置命令。

参　　数：

char **argv

处理流程：

1. 定义 char 指针指向 4 个内置命令。
2. 对 4 个内置命令依次判断，若相等则执行相应的命令。
   a) 若是 quit 则调用 exit 直接结束。
   b) 若是 fg 或者 bg 则调用 do_bgfg。
   c) 若是 jobs 则调用 listjobs。

要点分析：

1. 注意若输入的命令能够与 4 个内置命令匹配，则除了 quit 命令，其他命令执行完之后要 return 1 以告诉 eval 函数输入的命令是内置命令。

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：

1. 判断是否存在第一个参数，若不存在则输出提示信息并返回。
2. 判断第一个参数的第一位是数字还是‘%’：
   a) 若是数字，则说明输入的可能是 PID，查找 PID 对应的 JobID，若查找失败则输出提示信息并返回。
   b) 若是‘%’，则说明输入的可能是 JobID，查找 Job 列表中是否有当前的 JobID，若查找失败则输出提示信息并返回。
   c) 若上述都不是，则输出提示信息并返回。
3. 判断当前命令是 bg 还是 fg：
   a) 若是 bg，则调用 kill 函数向该进程发送 SIGCONT 信号，将 job 的状态设置为 BG 并输出提示信息。
   b) 若是 fg，则调用 kill 函数向该进程发送 SIGCONT 信号，将 job 的状态设置为 FG 并调用 waitfg 等待该进程运行完毕。

要点分析：

1. 要考虑到输入参数的各种情况，若处理不完善则可能会发生不可预知的错误。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：

等待一个前台命令结束运行。

参　　数：

pid_t pid

处理流程：

1. 使用 while 循环，每次循环调用 sleep 函数休眠 1 秒，循环条件为前台运行的进程的 pid 等于 pid。

要点分析：

1. 我们可以调用 fgpid 函数获取前台运行的进程的 pid。

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

函数功能：

SIGCHILD 的信号处理程序。

参　　数：

int sig

处理流程：

1. 使用 while 循环，循环条件是有被终止或暂停的子进程，判断导致返回的进程的状态：
   a) 若进程正常结束了，则先阻塞所有信号，再在 job 列表中删除被终止的进程，最后将信号阻塞列表还原。
   b) 若进程被暂停了，则调用 getjobpid 函数找到进程在 job 列表中的 jobID，将该 job 的状态设置为 ST，输出提示信息。
   c) 若进程被异常终止了，则调用 getjobpid 函数找到进程在 job 列表中的 jobID，输出提示信息。再阻塞所有信号，在 job 列表中删除被终止的进程，将信号阻塞列表还原。

要点分析：

1. 要根据进程的返回状态来分开处理，使用不同的处理方法。
2. 调用 deletejob 之前要阻塞所有信号的输入。

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
 *
 * 1190200523 Xiangyu Shi
 */
#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 */
        if (sigprocmask(SIG_UNBLOCK, &mask, NULL) < 0)
            unix_error("sigprocmask error");

        /* 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);
    if (sigprocmask(SIG_UNBLOCK, &mask, NULL) < 0)
        unix_error("sigprocmask error");

    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)
{
    char *cmd_quit = "quit";
    char *cmd_fg = "fg";
    char *cmd_bg = "bg";
    char *cmd_jobs = "jobs";

    /* if the command is "quit", just exit. */
    if (!strcmp(argv[0], cmd_quit)) {
        exit(0);
    }
    /* if the command is "fg" or "bg", call do_bgfg. */
    else if (!strcmp(argv[0], cmd_fg) || !strcmp(argv[0], cmd_bg)) {
        do_bgfg(argv);
        return 1;
    }
    /* if the command is "jobs", call listjobs to print job list. */
    else if (!strcmp(argv[0], cmd_jobs)) {
        listjobs(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)
{
    /* sleep until process pid is no longer the foreground process */
    while (fgpid(jobs) == pid)
        sleep(1);
}

/*****************
 * 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 status;
    sigset_t mask_all, prev_all;
    pid_t pid;

    if (sigfillset(&mask_all) < 0)
```

```
        unix_error("sigfillset error");
    while ((pid = waitpid(-1, &status, WNOHANG|WUNTRACED)) > 0) {
        if (WIFEXITED(status)) {
            /* killblock all signals. */
            if (sigprocmask(SIG_BLOCK, &mask_all, &prev_all) < 0)
                unix_error("sigprocmask error");
            /* delete process from job list */
            deletejob(jobs, pid);
            /* unblock signals. */
            if (sigprocmask(SIG_SETMASK, &prev_all, NULL) < 0)
                unix_error("sigprocmask error");
        }
        else if (WIFSTOPPED(status)) {
            struct job_t *job = getjobpid(jobs, pid);
            /* set status. */
            job->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n", job->jid, job->pid
, WSTOPSIG(status));
        }
        else if (WIFSIGNALED(status))
        {
            struct job_t *job = getjobpid(jobs, pid);
            printf("Job [%d] (%d) terminated by signal %d\n", job->jid, job->
pid, WTERMSIG(status));
            /* killblock all signals. */
            if (sigprocmask(SIG_BLOCK, &mask_all, &prev_all))
                unix_error("sigprocmask error");
            /* delete process from job list */
            deletejob(jobs, pid);
            /* unblock signals. */
            if (sigprocmask(SIG_SETMASK, &prev_all, NULL))
                unix_error("sigprocmask error");
        }
    }

}

/*
 * 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)
{
    sigset_t mask_all, prev_all;

    sigfillset(&mask_all);
    pid_t fg_pid = fgpid(jobs);
    if (fg_pid != 0) {
        /* killblock all signals. */
        if (sigprocmask(SIG_BLOCK, &mask_all, &prev_all))
            unix_error("sigprocmask error");
        /* send signal SIGINT to the foreground job. */
        if (kill(-fg_pid, SIGINT) < 0)
            unix_error("kill error");
```

```
        /* unblock signals. */
        if (sigprocmask(SIG_SETMASK, &prev_all, NULL))
            unix_error("sigprocmask error");
    }

}

/*
 * 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)
{
    sigset_t mask_all, prev_all;

    sigfillset(&mask_all);
    pid_t fg_pid = fgpid(jobs);
    if (fg_pid != 0) {
        /* killblock all signals. */
        if (sigprocmask(SIG_BLOCK, &mask_all, &prev_all))
            unix_error("sigprocmask error");
        /* send signal SIGTSTP to the foreground job. */
        if (kill(-fg_pid, SIGTSTP) < 0)
            unix_error("kill error");
        /* unblock signals. */
        if (sigprocmask(SIG_SETMASK, &prev_all, NULL))
            unix_error("sigprocmask error");
    }

}

/*********************
 * 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

tsh 测试结果	tshref 测试结果
./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #	./sdriver.pl -t trace01.txt -s ./tshref -a "-p" # # trace01.txt - Properly terminate on EOF. #
测试结论	相同

4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. #	./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. #
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
./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit	./sdriver.pl -t trace03.txt -s ./tshref -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit
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
./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (3840) ./myspin 1 &	./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (3846) ./myspin 1 &
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
./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (3852) ./myspin 2 & tsh> ./myspin 3 & [2] (3854) ./myspin 3 & tsh> jobs [1] (3852) Running ./myspin 2 & [2] (3854) Running ./myspin 3 &	./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (3861) ./myspin 2 & tsh> ./myspin 3 & [2] (3863) ./myspin 3 & tsh> jobs [1] (3861) Running ./myspin 2 & [2] (3863) Running ./myspin 3 &
测试结论	相同

4.3.6 测试用例 trace06.txt

tsh 测试结果	tshref 测试结果
./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (3871) terminated by signal 2	./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (3877) terminated by signal 2
测试结论	相同

4.3.7 测试用例 trace07.txt

tsh 测试结果	tshref 测试结果
./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (3901) ./myspin 4 & tsh> ./myspin 5 Job [2] (3903) terminated by signal 2 tsh> jobs [1] (3901) Running ./myspin 4 &	./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (3910) ./myspin 4 & tsh> ./myspin 5 Job [2] (3912) terminated by signal 2 tsh> jobs [1] (3910) Running ./myspin 4 &
测试结论	相同

4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果
./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (3919) ./myspin 4 & tsh> ./myspin 5 Job [2] (3921) stopped by signal 20 tsh> jobs [1] (3919) Running ./myspin 4 & [2] (3921) Stopped ./myspin 5	./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (3928) ./myspin 4 & tsh> ./myspin 5 Job [2] (3930) stopped by signal 20 tsh> jobs [1] (3928) Running ./myspin 4 & [2] (3930) Stopped ./myspin 5
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

```
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (3938) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3940) stopped by signal 20
tsh> jobs
[1] (3938) Running ./myspin 4 &
[2] (3940) Stopped ./myspin 5
tsh> bg %2
[2] (3940) ./myspin 5
tsh> jobs
[1] (3938) Running ./myspin 4 &
[2] (3940) Running ./myspin 5
```

```
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (3949) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3951) stopped by signal 20
tsh> jobs
[1] (3949) Running ./myspin 4 &
[2] (3951) Stopped ./myspin 5
tsh> bg %2
[2] (3951) ./myspin 5
tsh> jobs
[1] (3949) Running ./myspin 4 &
[2] (3951) Running ./myspin 5
```

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

```
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (3965) ./myspin 4 &
tsh> fg %1
Job [1] (3965) stopped by signal 20
tsh> jobs
[1] (3965) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

```
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (3977) ./myspin 4 &
tsh> fg %1
Job [1] (3977) stopped by signal 20
tsh> jobs
[1] (3977) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

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

```
./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] (3987) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1517 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GN
   1519 tty2     Sl+    0:48 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/
   1574 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --s
   2182 pts/0    Ss     0:00 bash
   3982 pts/0    S+     0:00 make test11
   3983 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh
   3984 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./
   3985 pts/0    S+     0:00 ./tsh -p
   3991 pts/0    R      0:00 /bin/ps a
```

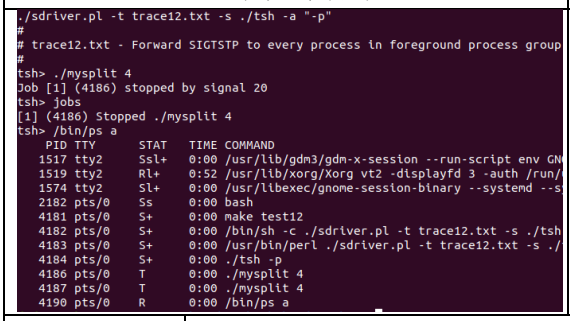
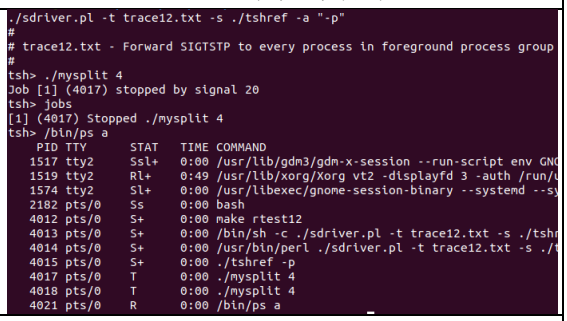
```
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (3997) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1517 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GN
   1519 tty2     Sl+    0:48 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/
   1574 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --s
   2182 pts/0    Ss     0:00 bash
   3992 pts/0    S+     0:00 make rtest11
   3993 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh
   3994 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./
   3995 pts/0    S+     0:00 ./tshref -p
   4000 pts/0    R      0:00 /bin/ps a
```

测试结论	相同

4.3.12 测试用例 trace12.txt

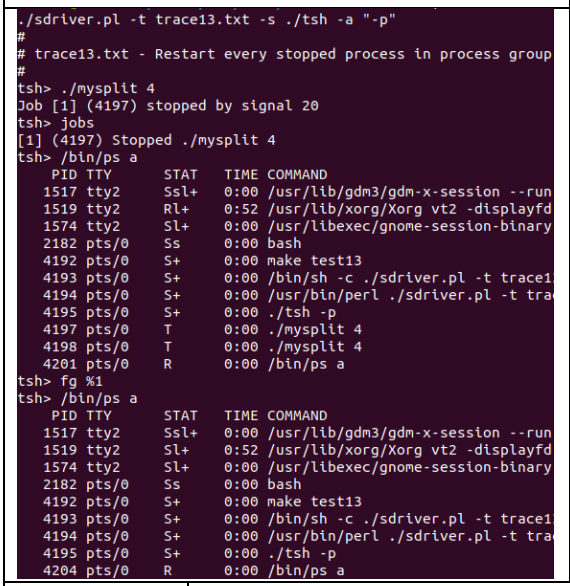
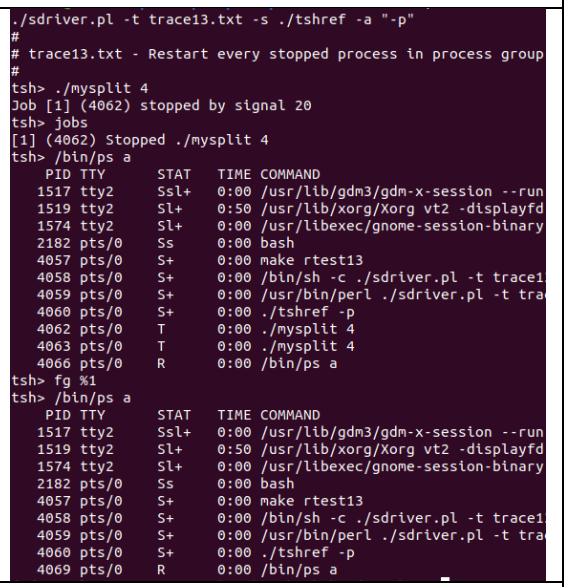
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、

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
	
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

```
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (4217) ./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] (4217) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4217) ./myspin 4 &
tsh> jobs
[1] (4217) Running ./myspin 4 &
```

```
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (4236) ./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] (4236) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4236) ./myspin 4 &
tsh> jobs
[1] (4236) 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

```
./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] (6260) terminated by signal 2
tsh> ./myspin 3 &
[1] (6262) ./myspin 3 &
tsh> ./myspin 4 &
[2] (6264) ./myspin 4 &
tsh> jobs
[1] (6262) Running ./myspin 3 &
[2] (6264) Running ./myspin 4 &
tsh> fg %1
Job [1] (6262) stopped by signal 20
tsh> jobs
[1] (6262) Stopped ./myspin 3 &
[2] (6264) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (6262) ./myspin 3 &
tsh> jobs
[1] (6262) Running ./myspin 3 &
[2] (6264) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
./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] (6280) terminated by signal 2
tsh> ./myspin 3 &
[1] (6282) ./myspin 3 &
tsh> ./myspin 4 &
[2] (6284) ./myspin 4 &
tsh> jobs
[1] (6282) Running ./myspin 3 &
[2] (6284) Running ./myspin 4 &
tsh> fg %1
Job [1] (6282) stopped by signal 20
tsh> jobs
[1] (6282) Stopped ./myspin 3 &
[2] (6284) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (6282) ./myspin 3 &
tsh> jobs
[1] (6282) Running ./myspin 3 &
[2] (6284) 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

- 对信号的发出、接收、阻塞等机制有了更深的理解。
- 更加深入地了解了 shell 的工作流程。

5.2 请给出对本次实验内容的建议

- 希望提供的代码能够少一些。

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.