哈爾濱工業大學

实验报告

实 验（七）

题　　目 TinyShell

微壳

专　　业 计算机

学　　号 1190202105

班　　级 1903002

学　　生 傅浩东

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.06.04

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；

1 TB SSD

1.2.2 软件环境

Windows 10 21H1；VirtualBox；Ubuntu 20.04 LTS

1.2.3 开发工具

EDB；GDB；CodeBlocks；vi/vim/gpedit+gcc；VSCode

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）

1. **进程概念：**程序是指令、数据及其组织形式的描述，进程是程序的实体。在现代系统上运行一个程序时，我们会得到一个假象，就好像我们的程序是系统中当 前运行的唯一的程序一样。我们的程序好像是独占地使用处理器和内存。处理器就好像是 无间断地一条接一条地执行我们程序中的指令。最后，我们程序中的代码和数据好像是系 统内存中唯一的对象。这些假象都是通过进程的概念提供给我们的。进程的经典定义就是一个执行中程序的实例。
2. **创建方法：**UNIX 系统父进程调用 fork 函数创建一个新的运行的子进程，子进程得到和父进程用户及虚拟地址空间完全相同的一个副本。只不过子进程中 fork 返回 0，父进程 fork 返回子进程的 PID。
3. **回收方法：**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。默认情况下（当 options=0 时）， waitpid 挂起调用进程的执行，直到它的 wait set 中的一个子进程终止，waitpid 返回导致 waitpid 返回的已终止子进程的 PID。或者是调用 waitpid 函数的简单版本 wait 函数来回收进程，相当于于调用 waitpid(- l, &status, 0)。

2.2 信号的机制、种类（5 分）

1. **信号机制：**信号提供了一种机制，通知用户进程发生的异常。每种信号类型都对应于某种系统事件，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。低层的硬件异常是由内核异常处理程序处理的，正常情况下，对用户进程而言是不可见的。
2. **信号种类**

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）

1. **信号发送方法：**Unix 系统提供了大量向进程发送信号的机制。所有这些机制都是基于进程组（process group）概念的。

   (1) 用/bin/kill 程序向另外的进程发送任意的信号。

   (2) 从键盘发送信号。

   (3) 进程通过调用 kill 函数发送信号给其他进程（包括它们自己）。

   (4) 进程用 alarm 函数向它自己发送 SIGALRM 信号。

2. **信号阻塞方法**

   隐式阻塞机制：内核默认阻塞任何当前处理程序正在处理信号类和待处理信号。

显示阻塞进制：应用程序可以调用 sigprocmask 函数和它的辅助函数，明确地阻塞和解除阻塞选定的信号。

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

2.4 什么是 shell，功能和处理流程（5 分）

1. **Shell 概念：**Shell 是系统的用户界面，提供了用户与内核进行交互操作的一种接口，Shell 是一种命令行解释器，其读取用户输入的字符串命令，解释并把它们送到内核。它是一种特殊的应用程序，介于系统调用/库和应用程序之间，提供了运行其他程序的接口。
2. **Shell 功能：**Shell 这个应用程序提供了一个界面，用户通过这个界面访问操作系统内核的服务。
3. **Shell 处理流程**

(1) 从终端读入输入的命令。

(2) 将输入字符串切分获得所有的参数。

(3) 如果是内置命令则立即执行。

(4) 否则调用相应的程序执行。

(5) shell 应该接受键盘输入信号，并对这些信号进行相应处理。

第 3 章 TinyShell 的设计与实现

总分 45 分

3.1 设计

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

函数功能：检测并执行内置命令
参　　数：char *cmdline
处理流程：
1. 调用 parseline 函数解析命令行参数，判断是否为内置命令。
2. 调用函数 build_cmd，判断是否为内置命令 quit、jobs、bg、fg。
3. 是则继续执行函数 build_cmd，不是则返回 0。首先阻塞 SIGCHLD、SIGINT、SIGTSTP 信号，然后子进程创建一个新的进程，调用 execve 函数。将子进程添加到 jobs 里，解除阻塞。命令后台运行则提示，前台则调用 waitfg 函数。

要点分析：
1. 判断出不是内置信号，一定要先阻塞信号 SIGCHLD、SIGINT、SIGTSTP，再添加到 jobs 列表。
2. 将子进程添加到一个进程组中（设置唯一进程组 ID），避免接收到其它的 SIGINT 信号等。

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

函数功能：处置内置命令 quit、jobs、bg、fg
参　　数：char **argv
处理流程：判断输入是否为 quit、jobs、bg、fg 命令之一，并进行后续处置
1. 内置命令 quit，调用 exit(0)直接退出；
2. 内置命令 fg/bg，调用函数 do_bgfg，并返回 1；
3. 内置命令 jobs，调用函数 listjobs，并返回 1.
4. 若不是内置命令则返回 0.

要点分析：
1. 对于这四种内置命令，除了 quit 直接退出，其他命令都要返回 1（都不是则返回 0），tinyshell 会根据此来继续处理。

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

函数功能：实现内置命令 bg/fg
参　　数：char **argv
处理流程：

1. 判断参数是否存在，不存在直接提示退出，否则继续执行下面程序。
2. 判断参数是 PID（参数第一位是数组）还是%JID（第一位是%），根据 PID/%JID 查找对应的 jobid，查找列表中是否有当前 jobid，都没找到则返回。
3. 命令为 bg：调用 kill 函数向进程组发送 SIGCONT 信号，将 job 状态设置为 BG。
4. 命令为 fg：调用 kill 函数向进程组发送 SIGCONT 信号，将 job 状态设置为 FG，调用 waitfg 函数等待当前进程运行。

要点分析：

1. 考虑处理多种参数输入情况，以及 fg 两个对应情况 stopped/进行状态。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：等待前台 pid 进程结束运行
参　　数：pid_t pid
处理流程：

1. While 循环，每次循环 sleep 一秒，直到前台进程的 PID 不是 pid。

要点分析：

1. 调用 fgpid 函数查询前台运行进程的 PID。

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

函数功能：处理 SIGCHID 信号
参　　数：int sig
处理流程：

1. 保存原始 errno。
2. While 循环，直到有被终止或暂停的子进程，处理该子进程
   (1) 调用 exit 或正常终止：首先阻塞所以信号，然后删除 job 列表中该进程，还原阻塞信号列表。
   (2) 未捕获的信号而异常终止：首先输出异常信息，接着阻塞所以信号，然后删除 job 列表中该进程，还原阻塞信号列表。
   (3) 被暂停：输出信息，调用函数 getjobpid 获取 pid，将 job 状态设置为 ST。
3. 恢复 errno

要点分析：

1. 信号处理可能会改变 errno，所以需要先保存后恢复；
2. 注意删除进程之前需要先阻塞信号。

3.2 程序实现（tsh.c 的全部内容）（10 分）

重点检查代码风格：

（1）用较好的代码注释说明——5 分

（2）检查每个系统调用的返回值——5 分

```c
/*
 * tsh - A tiny shell program with job control
 *
 * <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)) // SIGINT: Ctrl + C
        unix_error("sigaddset error");
    if (sigaddset(&mask, SIGTSTP)) // SIGTSTP: Ctrl + Z
        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)
{
    /* built-in command is quit, then quit directly */
    if (!strcmp(argv[0], "quit")) {
        exit(0);
    }
    /* built-in command is fg or bg, then execute do_bgfg, and return 1 *
/
    if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) {
        do_bgfg(argv);
        return 1;
    }
    /* built-in command is jobs, then execute listjobs, and return 1 */
    if (!strcmp(argv[0], "jobs")) {
        listjobs(jobs);
        return 1;
    }
    return 0; /* not a builtin command, just return 0 */
}

/*
 * 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)
{
    /* just sleep as blocked */
    while (pid == fgpid(jobs)) {
        sleep(1);
    }
    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 status;
    int olderrno = errno; /* save original errno */
    sigset_t mask, prev;
    pid_t pid;

    sigfillset(&mask); /* add all signals to mask */

    /* whenever a child job terminates or stops */
    while ((pid = waitpid(-1, &status, WUNTRACED | WNOHANG)) > 0)
    {
        if (WIFEXITED(status)) { // 子进程通过调用 exit 或正常终止
            /* killblock all signals, and save them in prev */
            sigprocmask(SIG_BLOCK, &mask, &prev);
            /* delete jobs from the list */
            deletejob(jobs, pid);
            /* unblock signals, restore original situation of blockedd */
            sigprocmask(SIG_SETMASK, &prev, NULL);
        }
        else if (WIFSIGNALED(status)) { // 子进程因未捕获的信号而终止
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid
), pid, WTERMSIG(status));
            /* killblock all signals, and save them in prev */
            sigprocmask(SIG_BLOCK, &mask, &prev);
            /* delete jobs from the list */
            deletejob(jobs, pid);
            /* unblock signals, restore original situation of blockedd */
            sigprocmask(SIG_SETMASK, &prev, NULL);
        }
        else if (WIFSTOPPED(status)) { // 子进程被暂停
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid),
pid, WSTOPSIG(status));
            /* set status */
```

```
            getjobpid(jobs, pid)->state = ST;
        }
    }
    errno = olderrno;
    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;
    sigset_t mask, prev;

    /* add all signals to mask */
    sigfillset(&mask);
    /* killblock all signals, and save them in prev */
    sigprocmask(SIG_BLOCK, &mask, &prev);

    pid_t pid = fgpid(jobs);
    if (pid) {
        /* send SIGNAL to foreground job */
        if (kill(-pid, SIGINT) < 0)
            unix_error("kill error\n");
    }
    /* unblock signals, restore original situation of blockedd */
    sigprocmask(SIG_SETMASK, &prev, NULL);
    errno = olderrno;
    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;
    sigset_t mask, prev;

    /* add all signals to mask */
```

```
    sigfillset(&mask);
    /* killblock all signals, and save them in prev */
    sigprocmask(SIG_BLOCK, &mask, &prev);

    pid_t pid = fgpid(jobs);
    if (pid) {
        /* send SIGNAL to foreground job */
        if (kill(-pid, SIGTSTP) < 0)
            unix_error("kill error\n");
    }
    /* unblock signals, restore original situation of blockedd */
    sigprocmask(SIG_SETMASK, &prev, NULL);
    errno = olderrno;
    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><th>tsh 测试结果</th><th>tshref 测试结果</th></tr>
<tr><td>ifu@ifu-VirtualBox:shlab-handout-hit2$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
trace01.txt - Properly terminate on EOF.
#</td><td>ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
trace01.txt - Properly terminate on EOF.
#</td></tr>
<tr><td>测试结论</td><td colspan="2">相同</td></tr>
</table>

4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果

ifu@ifu-VirtualBox:shlab-handout-hit2$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. #	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest02 ./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
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest03 ./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
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test04 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (3930) ./myspin 1 &	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest04 ./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (3936) ./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
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test05 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (3942) ./myspin 2 & tsh> ./myspin 3 & [2] (3944) ./myspin 3 & tsh> jobs [1] (3942) Running ./myspin 2 & [2] (3944) Running ./myspin 3 &	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest05 ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (3951) ./myspin 2 & tsh> ./myspin 3 & [2] (3953) ./myspin 3 & tsh> jobs [1] (3951) Running ./myspin 2 & [2] (3953) 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

ifu@ifu-VirtualBox:shlab-handout-hit2$ make test06 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (3965) terminated by signal 2	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest06 ./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (3971) 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
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test07 ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (3977) ./myspin 4 & tsh> ./myspin 5 Job [2] (3979) terminated by signal 2 tsh> jobs [1] (3977) Running ./myspin 4 &	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (3986) ./myspin 4 & tsh> ./myspin 5 Job [2] (3988) terminated by signal 2 tsh> jobs [1] (3986) Running ./myspin 4 &
测试结论	相同

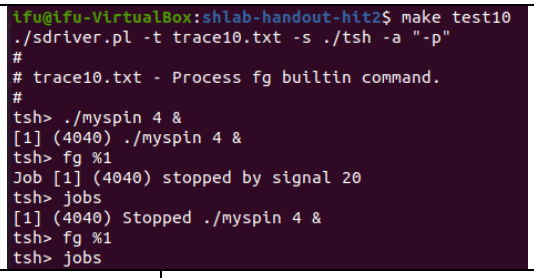
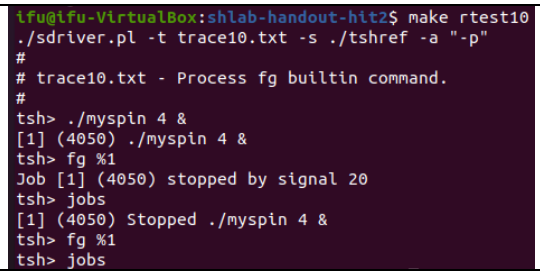
4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test08 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (3995) ./myspin 4 & tsh> ./myspin 5 Job [2] (3997) stopped by signal 20 tsh> jobs [1] (3995) Running ./myspin 4 & [2] (3997) Stopped ./myspin 5	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest08 ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (4004) ./myspin 4 & tsh> ./myspin 5 Job [2] (4006) stopped by signal 20 tsh> jobs [1] (4004) Running ./myspin 4 & [2] (4006) Stopped ./myspin 5
测试结论	相同

4.3.9 测试用例 trace09.txt

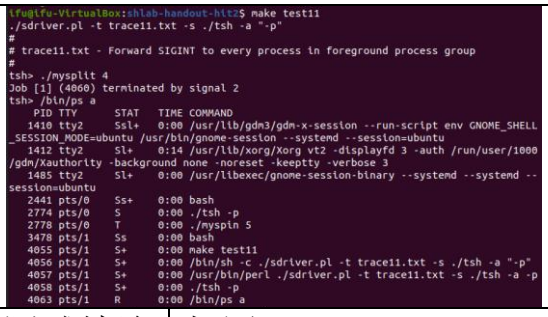
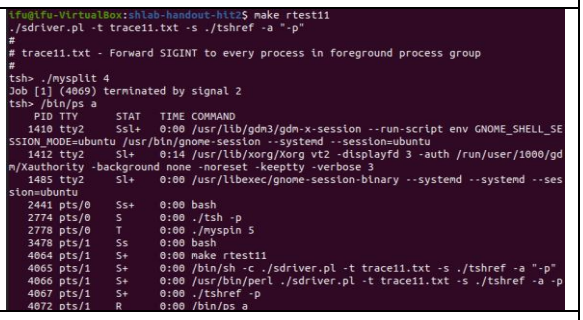
tsh 测试结果	tshref 测试结果
ifu@ifu-VirtualBox:shlab-handout-hit2$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (4013) ./myspin 4 & tsh> ./myspin 5 Job [2] (4015) stopped by signal 20 tsh> jobs [1] (4013) Running ./myspin 4 & [2] (4015) Stopped ./myspin 5 tsh> bg %2 [2] (4015) ./myspin 5 tsh> jobs [1] (4013) Running ./myspin 4 & [2] (4015) Running ./myspin 5	ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (4024) ./myspin 4 & tsh> ./myspin 5 Job [2] (4026) stopped by signal 20 tsh> jobs [1] (4024) Running ./myspin 4 & [2] (4026) Stopped ./myspin 5 tsh> bg %2 [2] (4026) ./myspin 5 tsh> jobs [1] (4024) Running ./myspin 4 & [2] (4026) 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
	
测试结论	相同

4.3.11 测试用例 trace11.txt

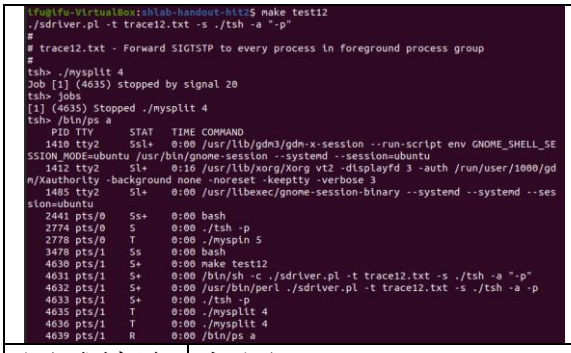
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
	
测试结论	相同

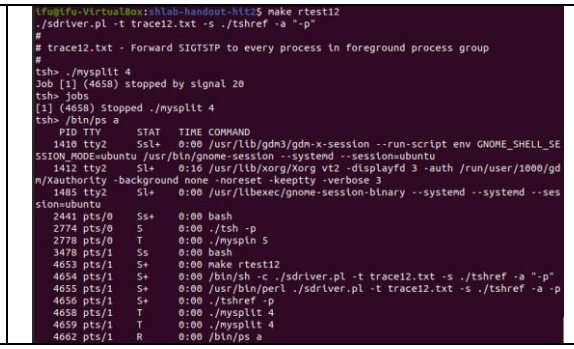
4.3.12 测试用例 trace12.txt

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

tsh 测试结果	tshref 测试结果



```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4635) stopped by signal 20
tsh> jobs
[1] (4635) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspin 5
   3478 pts/1    Ss     0:00 bash
   4630 pts/1    S+     0:00 make test12
   4631 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
   4632 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
   4633 pts/1    S+     0:00 ./tsh -p
   4635 pts/1    T      0:00 ./mysplit 4
   4636 pts/1    T      0:00 ./mysplit 4
   4639 pts/1    R      0:00 /bin/ps a
```



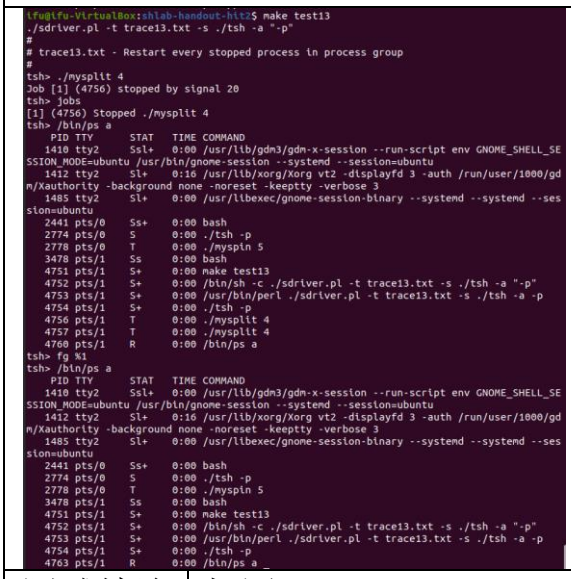
```
ifu@ifu-VirtualBox:shlab-handout-hit2$ make rtest12
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (4658) stopped by signal 20
tsh> jobs
[1] (4658) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspln 5
   3478 pts/1    Ss     0:00 bash
   4653 pts/1    S+     0:00 make rtest12
   4654 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
   4655 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
   4656 pts/1    S+     0:00 ./tshref -p
   4658 pts/1    T      0:00 ./mysplit 4
   4659 pts/1    T      0:00 ./mysplit 4
   4662 pts/1    R      0:00 /bin/ps a
```

测试结论	相同

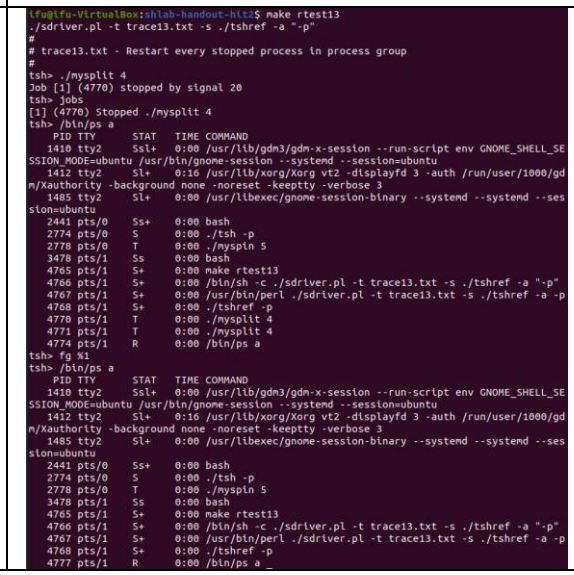
4.3.13 测试用例 trace13.txt

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



```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4756) stopped by signal 20
tsh> jobs
[1] (4756) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspin 5
   3478 pts/1    Ss     0:00 bash
   4751 pts/1    S+     0:00 make test13
   4752 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   4753 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   4754 pts/1    S+     0:00 ./tsh -p
   4756 pts/1    T      0:00 ./mysplit 4
   4757 pts/1    T      0:00 ./mysplit 4
   4760 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspin 5
   3478 pts/1    Ss     0:00 bash
   4751 pts/1    S+     0:00 make test13
   4752 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   4753 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   4754 pts/1    S+     0:00 ./tsh -p
   4763 pts/1    R      0:00 /bin/ps a
```



```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4770) stopped by signal 20
tsh> jobs
[1] (4770) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspin 5
   3478 pts/1    Ss     0:00 bash
   4765 pts/1    S+     0:00 make rtest13
   4766 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
   4767 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
   4768 pts/1    S+     0:00 ./tshref -p
   4771 pts/1    T      0:00 ./mysplit 4
   4774 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1410 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE
SSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1412 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd
m/Xauthority -background none -noreset -keeptty -verbose 3
   1485 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses
sion=ubuntu
   2441 pts/0    Ss+    0:00 bash
   2774 pts/0    S      0:00 ./tsh -p
   2778 pts/0    T      0:00 ./myspin 5
   3478 pts/1    Ss     0:00 bash
   4765 pts/1    S+     0:00 make rtest13
   4766 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
   4767 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
   4768 pts/1    S+     0:00 ./tshref -p
   4777 pts/1    R      0:00 /bin/ps a
```

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

```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4786) ./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] (4786) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4786) ./myspin 4 &
tsh> jobs
[1] (4786) Running ./myspin 4 &
```

```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4805) ./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] (4805) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4805) ./myspin 4 &
tsh> jobs
[1] (4805) 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

```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4830) terminated by signal 2
tsh> ./myspin 3 &
[1] (4832) ./myspin 3 &
tsh> ./myspin 4 &
[2] (4834) ./myspin 4 &
tsh> jobs
[1] (4832) Running ./myspin 3 &
[2] (4834) Running ./myspin 4 &
tsh> fg %1
Job [1] (4832) stopped by signal 20
tsh> jobs
[1] (4832) Stopped ./myspin 3 &
[2] (4834) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (4832) ./myspin 3 &
tsh> jobs
[1] (4832) Running ./myspin 3 &
[2] (4834) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
ifu@ifu-VirtualBox:shlab-handout-hit2$ 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] (4850) terminated by signal 2
tsh> ./myspin 3 &
[1] (4852) ./myspin 3 &
tsh> ./myspin 4 &
[2] (4854) ./myspin 4 &
tsh> jobs
[1] (4852) Running ./myspin 3 &
[2] (4854) Running ./myspin 4 &
tsh> fg %1
Job [1] (4852) stopped by signal 20
tsh> jobs
[1] (4852) Stopped ./myspin 3 &
[2] (4854) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (4852) ./myspin 3 &
tsh> jobs
[1] (4852) Running ./myspin 3 &
[2] (4854) 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

了解了 linux 信号机制的基本原理和相关系统函数

了解了 shell 的基本原理和实现方法

理解 shell 对命令的处理机制

5.2 请给出对本次实验内容的建议

希望有更多的时间来理解该实验

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.