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 1190200717

班　　级 1903008

学　　生 梁浩

指 导 教 师 吴锐

实 验 地 点 G709

实 验 日 期 2021/06/12

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

Ubuntu, gcc

1.2.1 硬件环境

处理器：Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 2.40GHz
已安装的内存(RAM)：8.00GB(7.81GB 可用)
系统类型：64 位操作系统，基于 x64 的处理器

1.2.2 软件环境

Windows 10 家庭中文版; VirtualBox 6.1; Ubuntu 20.04

1.2.3 开发工具

visual studio

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

kill －l：列出信号

kill －SIGKILL 17130： 杀死 pid 为 17130 的进程

kill -9 17130 ：杀死 pid 为 17130 的进程，或者：

kill -9 -17130：杀死进程组 17130 中的每个进程

killall -9 pname： 杀死名字为 pname 的进程

进程状态

D 不可中断睡眠 (通常是在 IO 操作) 收到信号不唤醒和不可运行, 进程必须等待直到有中断发生

R 正在运行或可运行（在运行队列排队中）

S 可中断睡眠 (休眠中, 受阻, 在等待某个条件的形成或接受到信号)

T 已停止的 进程收到 SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU 信号后停止运行

W 正在换页(2.6.内核之前有效)

X 死进程 (未开启)

Z 僵尸进程 a defunct (”zombie”) process

< 高优先级(not nice to other users)

N 低优先级(nice to other users)

L 页面锁定在内存（实时和定制的 IO）

s 一个信息头

l 多线程（使用 CLONE_THREAD，像 NPTL 的 pthreads 的那样）

+ 在前台进程组

ps t /ps aux /ps

t <终端机编号 n> 列终端 n 的程序的状况。

a 显示现行终端机下的所有程序，包括其他用户的程序。

u 以用户为主的格式来显示程序状况。

x 显示所有程序，不以终端来区分。

Linux>sleep 2000 |more|sort|grep hit &

Linux>ps -f a

Linux>ps aj

作业 ：jobs、 fg %n 、 bg%n

jobs 显示当前暂停的进程

bg %n 使第 n 个任务在后台运行(%前有空格)

fg %n 使第 n 个任务在前台运行

bg, fg 不带%n 表示对最后一个进程操作

ctrl+c: 终止前台作业(进程组的每个进程)

ctrl+z: 停止前台作业(进程组的每个进程)，随后可用 bg 恢复后台运行， fg 恢复前台运行。

第 2 章 实验预习

总分 20 分

2.1 进程的概念、创建和回收方法（5 分）

进程的概念:
进程的经典定义就是一个执行中程序的实例。程序本身只是指令、数据及其组织形式的描述，相当于一个名词，进程才是程序的真正运行实例。同一个程序处理不同的数据就是不同的进程。系统中的每个程序都运行在某个进程的上下文中。上下文是由程序正确运行所需的状态组成的。这个状态包括存放在内存中的程序的代码和数据，它的栈、通用目的寄存器的内容、程序计数器、环境变量以及打开文件描述符的集合。

进程的创建:
父进程通过调用 fork 函数创建一个新的运行的子进程。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（但是独立的）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，这就意味着当父进程调用 fork 时，子进程可以读写父进程中打开的任何文件。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。fork 函数被调用一次，会返回两次:一次是在调用进程（父进程）中，一次是在新创建的子进程中。在父进程中，fork 返回子进程的 PID。在子进程中，fork 返回 0。因为子进程的 PID 总是为非零，返回值就提供一个明确的方法来分辨程序是在父进程还是在子进程中执行。

进程的回收:
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，直到被它的父进程回收。当父进程回收已终止的子进程时，内核将子进程的退出状态传递给父进程，然后抛弃已终止的进程，从此时开始，该进程就不存在了。一个终止了但还未被回收的进程称为僵死进程。但是当父进程终止时，子进程会被交给 init 进程作为“养子”，在终止后被 init 回收。

2.2 信号的机制、种类（5 分）

信号的机制:

一个信号就是一条消息，它通知进程系统中发生了一个某种类型的事件。每种信号类型都对应某种系统的事件。信号类型是用小整数 ID(1-30)来标识，每一个信号对应唯一的 ID。信号中唯一的信息是它的 ID 和它的到达。

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

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

信号的发送方法：

1)用 /bin/kill 程序发送信号

/bin/kill 程序可以向另外的进程发送任意的信号

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，结果是终止前台作业。类似的，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

进程通过调用 kill 函数发送信号给其他进程（包括它们自己）。如果 pid 大于零，那么 kill 函数发送信号号码 sig 给进程 pid。如果 pid 等于零，那么 kill 发送信号 sig 给调用进程所在进程组中的每个进程，包括调用进程自己。如果 pid 小于零，kill 发送信号 sig 给进程组|pid|(pid 的绝对值)中的每个进程。

4)用 alarm 函数发送信号

进程可以通过调用 alarm 函数向它自己发送 SIGALRM 信号。alarm 函数安排内核在 secs 秒后发送一个 SIGALRM 信号给调用进程。如果 secs 是零,那么不会调度安排新的闹钟(alarm)。在任何情况下,对 alarm 的调用都将取消任何待处理的(pending)闹钟，并且返回任何待处理的闹钟在被发送前还剩下的秒数（如果这次对 alarm 的调用没有取消它的话）；如果没有任何待处理的闹钟，就返回零。

信号的阻塞方法:

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。

信号处理程序的设置方法：

G0. 处理程序尽可能简单

G1. 在处理程序中只调用异步信号安全的函数

G2. 保存和恢复 errno

G3. 阻塞所有信号保护对共享全局数据结构的访问

G4. 用 volatile 声明全局变量

G5. 强迫编译器从内存中读取引用的值
G6. 用 sig_atomic_t 声明标志

2.4 什么是 shell，功能和处理流程（5 分）

定义：
shell 是一个交互型应用级程序，代表用户运行其他程序，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。它接收用户输入的命令并把它送入内核去执行。

功能：
Shell 是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如此，Shell 有自己的编程语言用于对命令的编辑，它允许用户编写由 shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果。

处理流程：
当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；若是外部命令或实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。在查找该命令时分为两种情况：（1）用户给出了命令的路径，Shell 就沿着用户给出的路径进行查找，若找到则调入内存，若没找到则输出提示信息；（2）用户没有给出命令的路径，Shell 就在环境变量 PATH 所制定的路径中依次进行查找，若找到则调入内存，若没找到则输出提示信息。

第 3 章 TinyShell 的设计与实现

总分 45 分

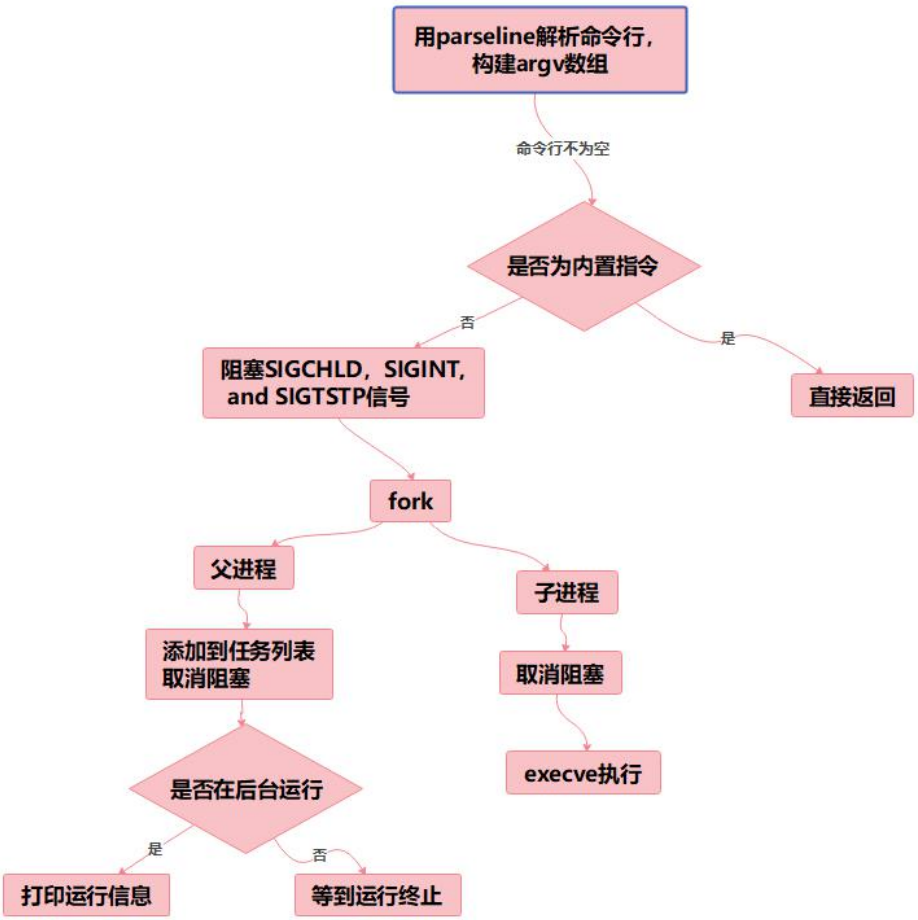
3.1 设计

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

函数功能：
解析用户输入的命令。如果用户输入的是内置的指令(quit, jobs, bg, fg)，立即执行。如果不是内置指令，fork 一个子进程，在子进程中运行指令，如果进程是在前台运行，就等到它结束。

参　　数：char *cmdline

处理流程：



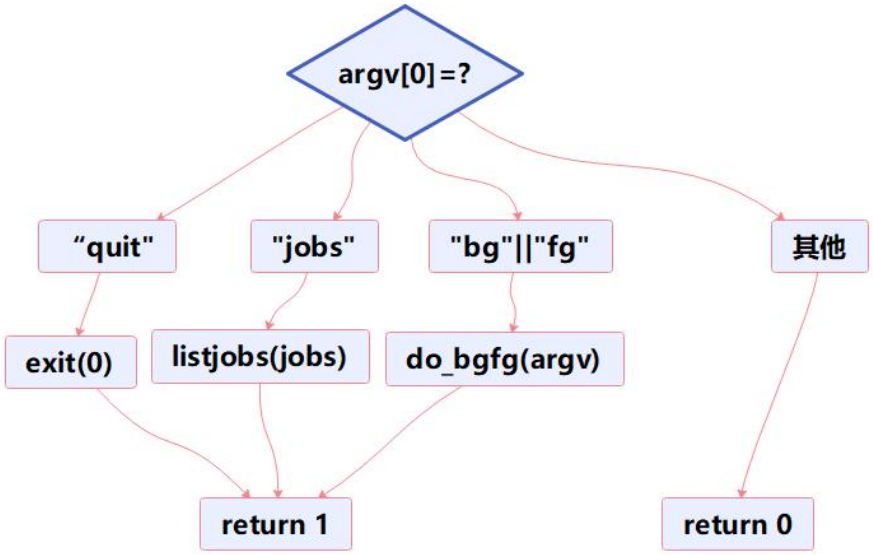
要点分析：
1、在 fork 子进程之前，应该先阻塞 SIGCHLD, SIGINT, SIGTSTP 信号，直到可以将任务添加进任务列表中。这是为了避免 SIGCHLD, SIGINT, SIGTSTP 信号的到来和添加任务进列表冲突。
2、每个子进程必须有新的进程组 ID，以防止发送的 Ctrl + C 和 Ctrl +Z 被 tsh 创建的所有子进程接收，通过使用 setpgid(0, 0)，将子进程的 PGID 修改为自己的 PID。

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

函数功能：
检查用户传入的命令行是否是 tsh 内置的命令。如果是，立即执行命令，并返回 1，如果不是，则返回 0。

参　　数：char **argv

处理流程：



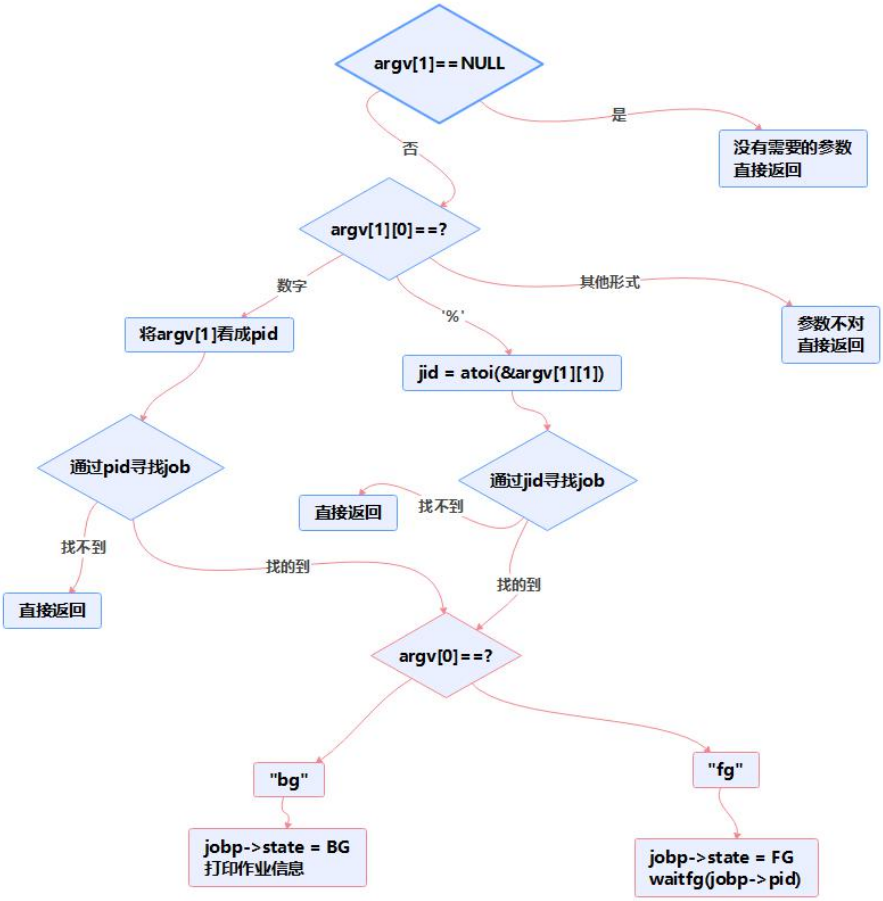
要点分析：
注意覆盖所有可能输入的情况，不同情况下返回不同的值。

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：



要点分析：
首先要判断命令后面跟的参数是数字还是%，然后通过进程号或工作组号来获得job，然后在前台或后台执行相关操作。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：等待一个前台作业结束

参　　数：pid_t pid

处理流程：
调用 fgpid()函数来判断该进程是否为前台运行，若是则循环，并用 sleep(1)使进程休眠。

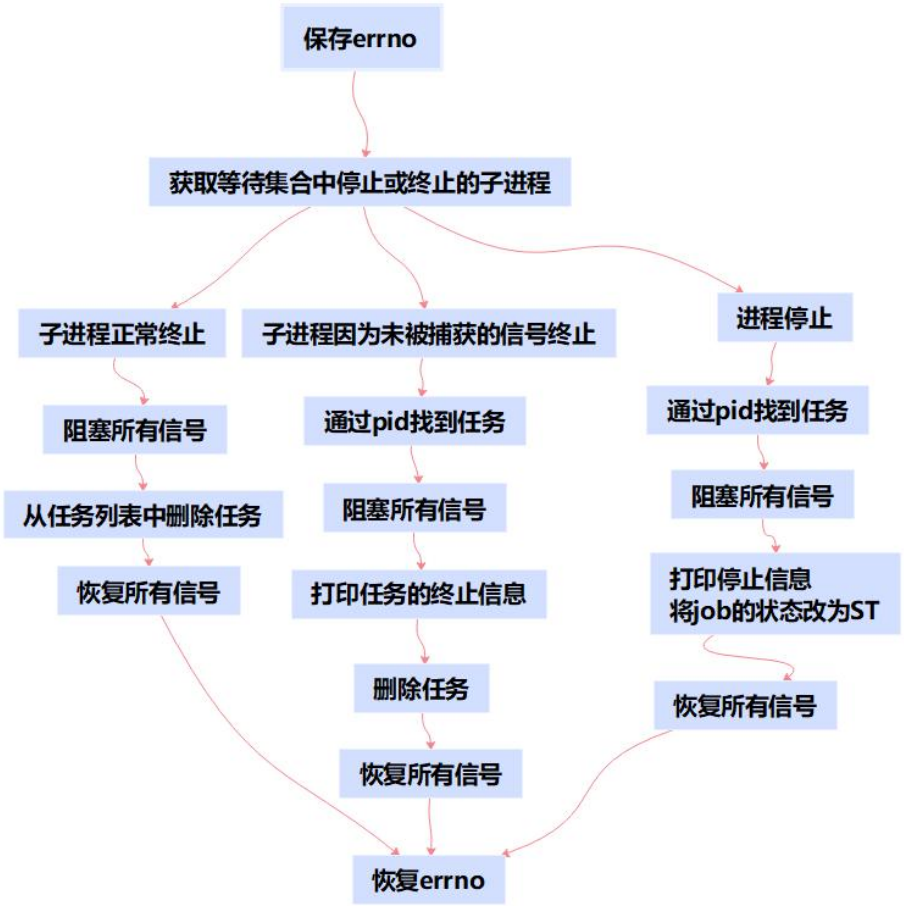
要点分析：
在 waitfg 函数中，在 sleep 函数附近使用 busy loop，例如：while (xxxxx) sleep(1);

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

函数功能：
当内核因为某个子进程终止或停止而发出一个 SIGCHLD 信号时，本函数将捕获 SIGCHILD 信号，回收所有的僵死子进程，不会等待别的子进程终止或停止。

参　　数：
int sig

处理流程：



要点分析：
1、WHOHANG | WUNTRACED：立即返回，如果等待中的子进程都没有被停止或终止，则返回值为 0 ；如果有一个停止或终止，则返回值为该子进程的 PID。

2、最后要恢复 errno

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
 *
 * <Liang Hao 1190200717>
 */
#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);//解析和解释命令行的主例程
int builtin_cmd(char **argv);//识别并解释内置命令
void do_bgfg(char **argv);//实现内置命令 bg 和 fg
void waitfg(pid_t pid);//等待一个前台作业结束

void sigchld_handler(int sig);//捕获 SIGCHILD 信号
void sigtstp_handler(int sig);//捕获 SIGINT (ctrl-c) 信号
void sigint_handler(int sig);//捕获 SIGTSTP (ctrl-z) 信号

/* 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) //识别并解释内置命令: quit, fg, bg, 和 jobs.
{
    if (!strcmp(argv[0], "quit")) {//如果是 quit 指令
        exit(0);
    }
    else if (!strcmp(argv[0], "jobs")) {//如果为 jobs，用 listjobs 输出作业列表的所有作业信息
        listjobs(jobs);
        return 1;
    }
    else if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) {//如果为 bg 或 fg，调用 do_bgfg
        do_bgfg(argv);
        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)//等待一个前台作业结束
{

    /* fgpid - Return PID of current foreground job, 0 if no such job */
    while (pid == fgpid(jobs))
    {
        sleep(1);  //暂时休眠挂起
    }
    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;
    int status;
    pid_t pid;
    sigset_t mask, prev_mask;
    struct job_t* job = NULL;//新建 job_t 的结构体指针 job
    //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 */
    //};
    sigfillset(&mask);//将每个信号都添加到 mask 中

    //WNOHANG | WUNTRACED:立即返回，如果等待集合中的子进程都没有被停止或终止，则返回值为 0
    //如果有一个停止或终止，则返回值为该子进程的 PID。
    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)
    {
        if (WIFEXITED(status)) {//如果子进程是正常终止的
            sigprocmask(SIG_BLOCK, &mask, &prev_mask);//阻塞所有信号
            deletejob(jobs, pid);/* deletejob - Delete a job whose PID=pid from the job list */
            sigprocmask(SIG_SETMASK, &prev_mask, NULL);//该进程新的信号屏蔽字将被 pre_mask 的信号集的值代替
        }
        else if (WIFSIGNALED(status)) {//通过一个未被捕获的信号终止
            job = getjobpid(jobs, pid);/* getjobpid  - Find a job (by PID) on the job list */
            sigprocmask(SIG_BLOCK, &mask, &prev_mask);//阻塞所有信号
            printf("Job [%d] (%d) terminated by signal %d\n",job->jid, job->pid, WTERMSIG(status));
            deletejob(jobs, pid);
            sigprocmask(SIG_SETMASK, &prev_mask, NULL);
        }
        else if (WIFSTOPPED(status)) {//进程停止
            job = getjobpid(jobs, pid);
            sigprocmask(SIG_BLOCK, &mask, &prev_mask);
            printf("Job [%d] (%d) stopped by signal %d\n",job->jid, job->pid, WSTOPSIG(status));
            job->state = ST;
            sigprocmask(SIG_SETMASK, &prev_mask, NULL);
```

```
        }
    }
    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)
{

    //SIGINT:程序终止(interrupt)信号, 在用户键入 INTR 字符(通常是 Ctrl-C)时发出，用于通知前台进程组终止进程。
    pid_t pid;
    sigset_t mask, prev_mask;
    int olderrno = errno;

    sigfillset(&mask);

    sigprocmask(SIG_BLOCK, &mask, &prev_mask);  //阻塞所有信号
    pid = fgpid(jobs);/* fgpid - Return PID of current foreground job, 0 if no such job */
    sigprocmask(SIG_SETMASK, &prev_mask, NULL);

    if (pid != 0) { //杀死前台进程
        kill(-pid, SIGINT);
    }
    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)
{
```

```
    //SIGTSTP:停止进程的运行，但该信号可以被处理和忽略. 用户键入 SUSP 字符时(通常是 Ctrl-Z)发出这个信号

    pid_t pid;
    sigset_t mask, prev_mask;
    int olderrno = errno;

    sigfillset(&mask);
    sigprocmask(SIG_BLOCK, &mask, &prev_mask);  //阻塞所有信号

    pid = fgpid(jobs);/* fgpid - Return PID of current foreground job, 0 if no such job */
    sigprocmask(SIG_SETMASK, &prev_mask, NULL);

    if (pid != 0) {//向前台进程发送 SIGTSTP
        kill(-pid, SIGTSTP);
    }

    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><td colspan="2">tsh 测试结果</td></tr>
<tr><td colspan="2">1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
trace01.txt - Properly terminate on EOF.
#</td></tr>
<tr><td colspan="2">tshref 测试结果</td></tr>
<tr><td colspan="2">1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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>相同</td></tr>
</table>

4.3.2 测试用例 trace02.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (3673) ./myspin 1 &
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (3679) ./myspin 1 &
```

测试结论	相同

4.3.5 测试用例 trace05.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (5511) ./myspin 2 &
tsh> ./myspin 3 &
[2] (5513) ./myspin 3 &
tsh> jobs
[1] (5511) Running ./myspin 2 &
[2] (5513) Running ./myspin 3 &
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (5664) ./myspin 2 &
tsh> ./myspin 3 &
[2] (5666) ./myspin 3 &
tsh> jobs
[1] (5664) Running ./myspin 2 &
[2] (5666) Running ./myspin 3 &
```

测试结论	相同

4.3.6 测试用例 trace06.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (12050) terminated by signal 2
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (12056) terminated by signal 2
```

测试结论	相同

4.3.7 测试用例 trace07.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (12286) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12288) terminated by signal 2
tsh> jobs
[1] (12286) Running ./myspin 4 &
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (12277) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12279) terminated by signal 2
tsh> jobs
[1] (12277) Running ./myspin 4 &
```

测试结论	相同

4.3.8 测试用例 trace08.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (12324) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12326) stopped by signal 20
tsh> jobs
[1] (12324) Running ./myspin 4 &
[2] (12326) Stopped ./myspin 5
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (12315) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12317) stopped by signal 20
tsh> jobs
[1] (12315) Running ./myspin 4 &
[2] (12317) Stopped ./myspin 5
```

测试结论	相同

4.3.9 测试用例 trace09.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (12336) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12338) stopped by signal 20
tsh> jobs
[1] (12336) Running ./myspin 4 &
[2] (12338) Stopped ./myspin 5
tsh> bg %2
[2] (12338) ./myspin 5
tsh> jobs
[1] (12336) Running ./myspin 4 &
[2] (12338) Running ./myspin 5
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (12347) ./myspin 4 &
tsh> ./myspin 5
Job [2] (12349) stopped by signal 20
tsh> jobs
[1] (12347) Running ./myspin 4 &
[2] (12349) Stopped ./myspin 5
tsh> bg %2
[2] (12349) ./myspin 5
tsh> jobs
[1] (12347) Running ./myspin 4 &
[2] (12349) Running ./myspin 5
```

测试结论	相同

4.3.10 测试用例 trace10.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (12359) ./myspin 4 &
tsh> fg %1
Job [1] (12359) stopped by signal 20
tsh> jobs
[1] (12359) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (12369) ./myspin 4 &
tsh> fg %1
Job [1] (12369) stopped by signal 20
tsh> jobs
[1] (12369) 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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make test11
./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (12403) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:13 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -bac
kground none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss     0:00 bash
  12398 pts/0    S+     0:00 make test11
  12399 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
  12400 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
  12401 pts/0    S+     0:00 ./tsh -p
  12406 pts/0    R      0:00 /bin/ps a
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ make rtest11
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (12412) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:13 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -bac
kground none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss     0:00 bash
  12407 pts/0    S+     0:00 make rtest11
  12408 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
  12409 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p
  12410 pts/0    S+     0:00 ./tshref -p
  12415 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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12426) stopped by signal 20
tsh> jobs
[1] (12426) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -bac
kground none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss     0:00 bash
  12421 pts/0    S+     0:00 make test12
  12422 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
  12423 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
  12424 pts/0    S+     0:00 ./tsh -p
  12426 pts/0    T      0:00 ./mysplit 4
  12427 pts/0    T      0:00 ./mysplit 4
  12430 pts/0    R      0:00 /bin/ps a
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12436) stopped by signal 20
tsh> jobs
[1] (12436) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:16 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -bac
kground none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss     0:00 bash
  12431 pts/0    S+     0:00 make rtest12
  12432 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
  12433 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
  12434 pts/0    S+     0:00 ./tshref -p
  12436 pts/0    T      0:00 ./mysplit 4
  12437 pts/0    T      0:00 ./mysplit 4
  12440 pts/0    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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12499) stopped by signal 20
tsh> jobs
[1] (12499) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:18 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss+    0:00 bash
  12488 pts/1    Ss     0:00 bash
  12494 pts/1    S+     0:00 make test13
  12495 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
  12496 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
  12497 pts/1    S+     0:00 ./tsh -p
  12499 pts/1    T      0:00 ./mysplit 4
  12500 pts/1    T      0:00 ./mysplit 4
  12503 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:18 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss+    0:00 bash
  12488 pts/1    Ss     0:00 bash
  12494 pts/1    S+     0:00 make test13
  12495 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
  12496 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
  12497 pts/1    S+     0:00 ./tsh -p
  12506 pts/1    R      0:00 /bin/ps a
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12512) stopped by signal 20
tsh> jobs
[1] (12512) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:18 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss+    0:00 bash
  12488 pts/1    Ss     0:00 bash
  12507 pts/1    S+     0:00 make rtest13
  12508 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
  12509 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
  12510 pts/1    S+     0:00 ./tshref -p
  12512 pts/1    T      0:00 ./mysplit 4
  12513 pts/1    T      0:00 ./mysplit 4
  12516 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1370 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
   1372 tty2     Sl+    0:18 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1418 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
  12392 pts/0    Ss+    0:00 bash
  12488 pts/1    Ss     0:00 bash
  12507 pts/1    S+     0:00 make rtest13
  12508 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
  12509 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
  12510 pts/1    S+     0:00 ./tshref -p
  12519 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

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12537) ./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] (12537) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (12537) ./myspin 4 &
tsh> jobs
[1] (12537) Running ./myspin 4 &
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12556) ./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] (12556) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (12556) ./myspin 4 &
tsh> jobs
[1] (12556) Running ./myspin 4 &
```

测试结论	相同

4.3.15 测试用例 trace15.txt

tsh 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12613) terminated by signal 2
tsh> ./myspin 3 &
[1] (12615) ./myspin 3 &
tsh> ./myspin 4 &
[2] (12617) ./myspin 4 &
tsh> jobs
[1] (12615) Running ./myspin 3 &
[2] (12617) Running ./myspin 4 &
tsh> fg %1
Job [1] (12615) stopped by signal 20
tsh> jobs
[1] (12615) Stopped ./myspin 3 &
[2] (12617) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (12615) ./myspin 3 &
tsh> jobs
[1] (12615) Running ./myspin 3 &
[2] (12617) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

tshref 测试结果

```
1190200717lh@Graham:~/lab7/shlab-handout-hit$ 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] (12633) terminated by signal 2
tsh> ./myspin 3 &
[1] (12635) ./myspin 3 &
tsh> ./myspin 4 &
[2] (12637) ./myspin 4 &
tsh> jobs
[1] (12635) Running ./myspin 3 &
[2] (12637) Running ./myspin 4 &
tsh> fg %1
Job [1] (12635) stopped by signal 20
tsh> jobs
[1] (12635) Stopped ./myspin 3 &
[2] (12637) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (12635) ./myspin 3 &
tsh> jobs
[1] (12635) Running ./myspin 3 &
[2] (12637) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

测试结论	相同

第 5 章 评测得分

总分 20 分

实验程序统一测试的评分（教师评价）：

（1）正确性得分：_____（满分 10）

（2）性能加权得分：_____（满分 10）

第 6 章 总结

5.1 请总结本次实验的收获

1)更深入地了解了 shell 的工作原理
2)理解了信号的处理过程，了解信号机制和信号处理关系的系统函数
3)初步尝试使用和信号有关的函数，感觉很新奇

5.2 请给出对本次实验内容的建议

无

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 深入理解计算机系统