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 1190202128

班　　级 1903002

学　　生 林搏海

指 导 教 师 郑贵滨

实 验 地 点 G704

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

I7-9750H CPU　8g 内存　983.58G 硬盘　GTX1650 显卡

1.2.2 软件环境

VirtualBox 6.1.18　Ubuntu20.04.2LTS

1.2.3 开发工具

VIM8.1.2269　Visual Studio Code1.54.3

1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

了解进程、作业、信号的基本概念和原理

了解 shell 的基本原理

熟知进程创建、回收的方法和相关系统函数

熟知信号机制和信号处理相关的系统函数

第 2 章 实验预习

总分 20 分

2.1 进程的概念、创建和回收方法（5 分）

进程的概念：

进程是计算机科学中的一个抽象概念其定义是一个执行中的程序的实例，是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体。

进程的创建方法：

进程的创建工作在 UNIX 系统下由 fork()函数实现，在 WINDOWS 系统下由 CreateProcess()函数进行创建

进程的回收方法：

对于已经结束的子进程，父进程需要调用 wait()或 waitpid()等函数对进程进行回收，否则终止的进程不被回收就会成为僵死进程(Zombie)过多僵死进程的存在会影响系统性能。

2.2 信号的机制、种类（5 分）

信号的机制：

信号是一种消息，通知进程系统中发生了一个某种类型的事件，它提供了一种机制，使得用户进程对于系统中的一些异常有着了解。

信号的种类：

系统中收到不同的信号，都会有着对应的默认行为，一般分为四种：1、进程终止；2、进程终止并转储内存；3、进程停止（挂起）直到被 SIGCONT 信号重启：4、进程忽略该信号。

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

信号的发送方法：

信号的发送方法主要由以下几种：1、使用 bin/kill/程序发送信号，该程序可以向一个制定的进程或者是进程组发送任意信号；2、从键盘发送信号。例如：在键盘上同时输入 Ctrl+C 会导致内核发送一个 SIGINT 信号到前台进程组的每个进程中；3、可以通过 kill 函数向指定进程组或者是指定进程发送任意信号；4、可以使用 alarm 函数向调用进程自己发送 SUGALRM 信号。

信号的阻塞方法：

信号的阻塞方式包括了隐式阻塞机制和显式阻塞机制。隐式阻塞机制主要是指内核默认阻塞任何当前处理信号程序正在处理的信号类型的待处理信号。显式阻塞机制主要是指通过 sigprocmask 函数改变当前的阻塞信号集合。

处理信号的方法：

我们可以通过 signal 函数设置对指定信号的处理方法，可以设置为忽略改信号，恢复默认处理方法或者是跳转向指定的处理函数位置。

2.4 什么是 shell，功能和处理流程（5 分）

Shell 的定义：

Shell 俗称壳（用来区别于核），是指“为使用者提供操作界面”的软件（command interpreter，命令解析器）。

Shell 的功能：

它接收用户命令，然后解释并执行命令，或者调用相应的应用程序。

Shell 的处理流程：

Shell 会先等待命令行的输入，然后进行命令解析，判断是否是内置命令，如果是则立即执行，如果不是则调用相应的程序运行，然后再重复该流程。

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：

解析命令行并执行

参　　数：

char *cmdline（输入的命令行）

处理流程：

首先，调用 parse 函数解析命令行，判断其是否为后台命令，其次，调用 bulitin_cmd 函数判断是否是内置命令，如果是则立即执行，如果不是就创建子进程，并且使子进程加载目标程序运行，然后在父进程中将子进程加入作业（job）并且根据是否是前台命令决定是否等待子进程的执行。

要点分析：

在实现程序的时候需要充分考虑父子进程的竞争冲突问题，在父进程改变 job 列表前需要阻塞 SIGCHLD，SIGINT 和 SIGTSTP 信号并且在之后解除阻塞。并且注意子进程会继承父进程的信号阻塞集，所以在调用 fork 函数之后还需要在子进程中解除信号阻塞。

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

函数功能：

判断是否是内置命令，若是内置命令则直接执行。

参　　数：

char **argv（根据输入的命令行 cmdline 解析拆分得到的字符串列表）

处理流程：

由于内置命令较少，因此使用 if 分支分别判断即可（如下图所示）

```c
int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0], "quit"))
        exit(0);
    else if(!strcmp(argv[0], "&"))
        return 1;

    else if(!strcmp(argv[0], "jobs")){
        listjobs(jobs);
        return 1;
    }
    else if(!strcmp(argv[0],"bg") || !strcmp(argv[0],"fg")){
        do_bgfg(argv);
        return 1;
    }
    return 0;     /* not a builtin command */
}
```

要点分析：

可以囊括一个特殊情况”&”，表示只输入了一个”&”时，我们也可以将其看做一个特殊的内置命令符号直接返回 1，增加程序的健壮性。

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

函数功能：

执行内置指令”bg”或者”fg”

参　　数：

char **argv（根据输入的命令行 cmdline 解析拆分得到的字符串列表）

处理流程：

首先我们解析传入的命令，判断是对第几个进程进行操作，然后根据对于前台和后台的判断

要点分析：

需要 stopped 的状态作出特殊检查。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：

等待直到前台进程不是 pid。

参　　数：pid_t pid（当前进程的 pid）

处理流程：

每次间隔一秒钟判断当前进程是否为前台进程即可（如下图）

```c
void waitfg(pid_t pid)
{
    while(fgpid(jobs) == pid)
        sleep(1);
    return;
}
```

要点分析：注意直接使用 sleep 函数即可，使用 suspend 函数难以对阻塞集作出判断。

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

函数功能：

处理接收到的信号 SIGCHLD

参　　数：

int sig（接收到的信号代码）

处理流程：

我们需要尽可能的回收更多的子进程，因此我们需要使用 while 循环进行判断，此外，我们需要区分造成信号发出的原因是子进程终止还是子进程停止，子进程终止的情况还需要继续判断是因为调用了函数 exit 或者 return 从而终止还是因为收到了一个信号导致终止，从而输出相关的信息并且修改 job 列表的状态。函数整体设计如下图：

```c
void sigchld_handler(int sig)
{
    int olderrno = errno;
    int status;
    pid_t pid;
    sigset_t mask, prev_mask;
    sigfillset(&mask);
    while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0){
        sigprocmask(SIG_BLOCK, &mask, &prev_mask);
        if(WIFSIGNALED(status) || WIFEXITED(status)){
            if(WIFSIGNALED(status))
                printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(status));
            deletejob(jobs, pid);
        }
        else if(WIFSTOPPED(status)){
            getjobpid(jobs, pid)->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, WSTOPSIG(status));
        }

        sigprocmask(SIG_SETMASK, &prev_mask, NULL);
    }
    errno = olderrno;
    return;
}
```

要点分析：

我们需要在针对 job 列表进行删除时需要屏蔽调所有的信号并且在之后及时进行恢复。此外，针对子进程停止的情况我们还需要将对应的 job 成员状态设为 ST。

3.2 程序实现（tsh.c 的全部内容）（10 分）

重点检查代码风格：

```
/*
 * tsh - A tiny shell program with job control
 *
 * <Linbohai 1190202128>
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>

/* Misc manifest constants */
#define MAXLINE    1024   /* max line size */
#define MAXARGS     128   /* max args on a command line */
#define MAXJOBS      16   /* max jobs at any point in time */
#define MAXJID    1<<16   /* max job ID */

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

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

/* Global variables */
extern char **environ;      /* defined in libc */
char prompt[] = "tsh> ";    /* command line prompt (DO NOT
```

```
CHANGE) */
int verbose = 0;                /* if true, print additional output */
int nextjid = 1;                /* next job ID to allocate */
char sbuf[MAXLINE];             /* for composing sprintf messages */

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


/* Function prototypes */

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

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

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

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

```
void listjobs(struct job_t *jobs);

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

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

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

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

    /* Install the signal handlers */
```

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

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

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

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

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

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

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

/*
 * eval - Evaluate the command line that the user has just typed in
 *
 * If the user has requested a built-in command (quit, jobs, bg or fg)
```

```
 * then execute it immediately. Otherwise, fork a child process and
 * run the job in the context of the child. If the job is running in
 * the foreground, wait for it to terminate and then return.  Note:
 * each child process must have a unique process group ID so that our
 * background children don't receive SIGINT (SIGTSTP) from the kernel
 * when we type ctrl-c (ctrl-z) at the keyboard.
*/
void eval(char *cmdline)
{
    /* $begin handout */
    char *argv[MAXARGS]; /* argv for execve() */
    int bg;              /* should the job run in bg or fg? */
    pid_t pid;           /* process id */
    sigset_t mask;       /* signal mask */

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

    if (!builtin_cmd(argv)) {

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

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

```
        unix_error("sigprocmask error");

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

/*
 * Child    process
 */

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

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

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

/*
 * Parent process
 */

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

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

```
    }
    /* $end handout */
    return;
}

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

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

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

    while (delim) {
  argv[argc++] = buf;
  *delim = '\0';
```

```
    buf = delim + 1;
    while (*buf && (*buf == ' ')) /* ignore spaces */
            buf++;

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

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

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

/*
 * builtin_cmd - If the user has typed a built-in command then execute
 *       it immediately.
 */
int builtin_cmd(char **argv)
{
     if(!strcmp(argv[0], "quit"))
        exit(0);
    else if(!strcmp(argv[0], "&"))
        return 1;

    else if(!strcmp(argv[0], "jobs")){
        listjobs(jobs);
        return 1;
    }
    else if(!strcmp(argv[0],"bg") || !strcmp(argv[0],"fg")){
```

```
        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)
{
    while(fgpid(jobs) == pid)
        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 olderrno = errno;
    int status;
    pid_t pid;
    sigset_t mask, prev_mask;
    sigfillset(&mask);
    while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) >
0){
        sigprocmask(SIG_BLOCK, &mask, &prev_mask);
        /* if process is terminated*/
        if(WIFSIGNALED(status) || WIFEXITED(status)){
            if(WIFSIGNALED(status))
                printf("Job [%d] (%d) terminated by signal %d\n",
pid2jid(pid), pid, WTERMSIG(status));
            deletejob(jobs, pid);
        }
        /* if process is stopped*/
        else if(WIFSTOPPED(status)){
            getjobpid(jobs, pid)->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n",
pid2jid(pid), pid, WSTOPSIG(status));
        }

        sigprocmask(SIG_SETMASK, &prev_mask, NULL);
    }
    /*save the original errno*/
    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;
    pid_t pid = fgpid(jobs);
    if(pid)
        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)
{
     int olderrno = errno;
    pid_t pid = fgpid(jobs);
    if(pid)
        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);
}
```

（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

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。

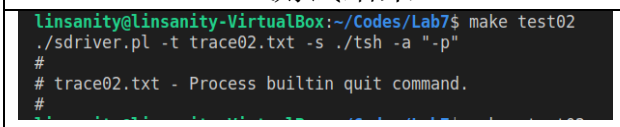
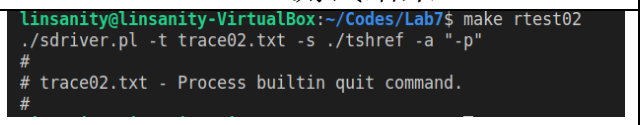
4.3 自测试结果

填写以下各个测试用例的测试结果，每个测试用例 1 分。

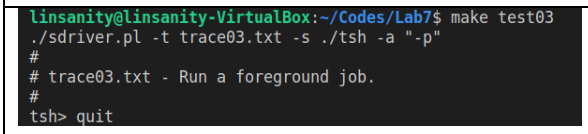
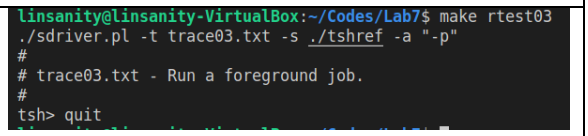
4.3.1 测试用例 trace01.txt

tsh 测试结果	tshref 测试结果
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test01 ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #	linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest01 ./sdriver.pl -t trace01.txt -s ./tshref -a "-p" # # trace01.txt - Properly terminate on EOF. #
测试结论	相同

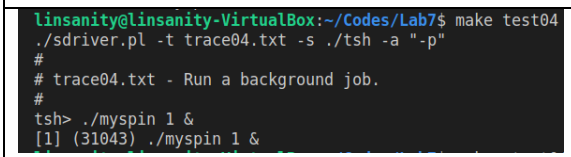
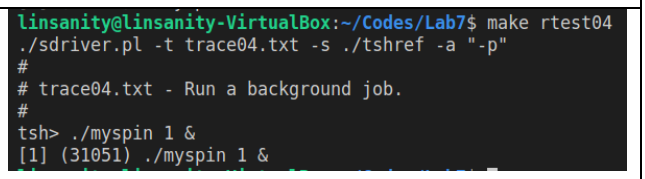
4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
	
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
	
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
	
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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (31058) ./myspin 2 &
tsh> ./myspin 3 &
[2] (31060) ./myspin 3 &
tsh> jobs
[1] (31058) Running ./myspin 2 &
[2] (31060) Running ./myspin 3 &
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (31070) ./myspin 2 &
tsh> ./myspin 3 &
[2] (31072) ./myspin 3 &
tsh> jobs
[1] (31070) Running ./myspin 2 &
[2] (31072) 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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (31700) terminated by signal 2
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (31647) 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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (31857) ./myspin 4 &
tsh> ./myspin 5
Job [2] (31859) terminated by signal 2
tsh> jobs
[1] (31857) Running ./myspin 4 &
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (31868) ./myspin 4 &
tsh> ./myspin 5
Job [2] (31870) terminated by signal 2
tsh> jobs
[1] (31868) 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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (32030) ./myspin 4 &
tsh> ./myspin 5
Job [2] (32032) stopped by signal 20
tsh> jobs
[1] (32030) Running ./myspin 4 &
[2] (32032) Stopped ./myspin 5
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (32040) ./myspin 4 &
tsh> ./myspin 5
Job [2] (32042) stopped by signal 20
tsh> jobs
[1] (32040) Running ./myspin 4 &
[2] (32042) 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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (32051) ./myspin 4 &
tsh> ./myspin 5
Job [2] (32053) stopped by signal 20
tsh> jobs
[1] (32051) Running ./myspin 4 &
[2] (32053) Stopped ./myspin 5
tsh> bg %2
[2] (32053) ./myspin 5
tsh> jobs
[1] (32051) Running ./myspin 4 &
[2] (32053) Running ./myspin 5
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (32063) ./myspin 4 &
tsh> ./myspin 5
Job [2] (32065) stopped by signal 20
tsh> jobs
[1] (32063) Running ./myspin 4 &
[2] (32065) Stopped ./myspin 5
tsh> bg %2
[2] (32065) ./myspin 5
tsh> jobs
[1] (32063) Running ./myspin 4 &
[2] (32065) 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

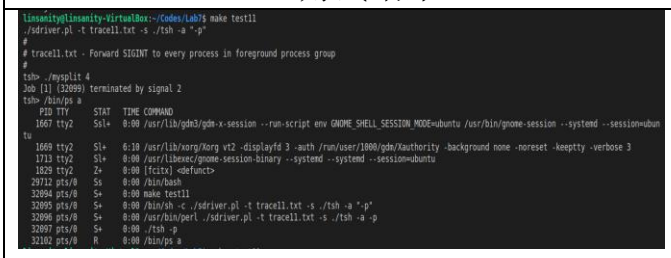
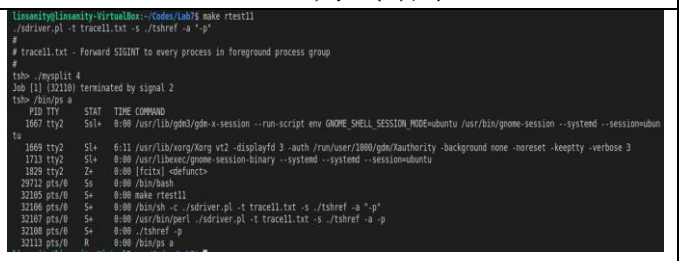
```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (32076) ./myspin 4 &
tsh> fg %1
Job [1] (32076) stopped by signal 20
tsh> jobs
[1] (32076) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (32087) ./myspin 4 &
tsh> fg %1
Job [1] (32087) stopped by signal 20
tsh> jobs
[1] (32087) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

测试结论	相同

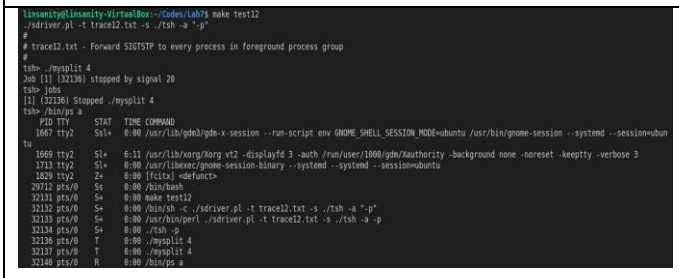
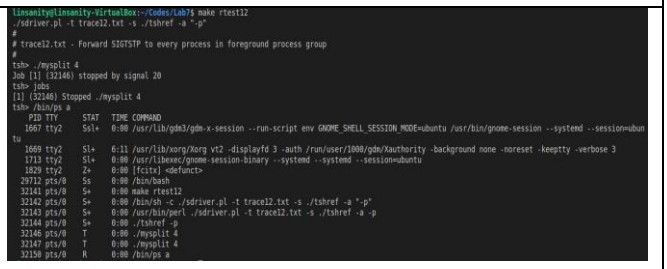
4.3.11 测试用例 trace11.txt

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
	
测试结论	相同

4.3.12 测试用例 trace12.txt

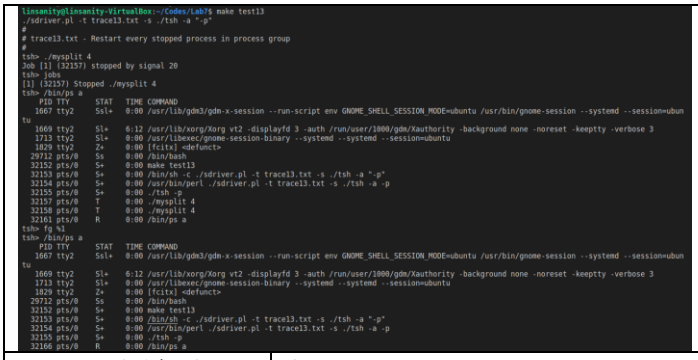
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

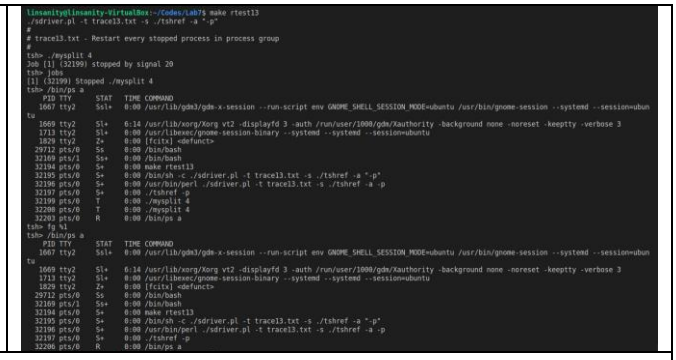
tsh 测试结果	tshref 测试结果
	
测试结论	相同

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
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ make test1 ./sdriver.pl -t trace14.txt -s ./tsh -a "-p" # # trace14.txt - Simple error handling # tsh> ./bogus ./bogus: Command not found tsh> ./myspin 4 & [1] (32219) ./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] (32219) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (32219) ./myspin 4 & tsh> jobs [1] (32219) Running ./myspin 4 &	linsanity@linsanity-VirtualBox:~/Codes/Lab7$ 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] (32238) ./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] (32238) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (32238) ./myspin 4 & tsh> jobs [1] (32238) 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

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ 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] (32258) terminated by signal 2
tsh> ./myspin 3 &
[1] (32264) ./myspin 3 &
tsh> ./myspin 4 &
[2] (32266) ./myspin 4 &
tsh> jobs
[1] (32264) Running ./myspin 3 &
[2] (32266) Running ./myspin 4 &
tsh> fg %1
Job [1] (32264) stopped by signal 20
tsh> jobs
[1] (32264) Stopped ./myspin 3 &
[2] (32266) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (32264) ./myspin 3 &
tsh> jobs
[1] (32264) Running ./myspin 3 &
[2] (32266) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
linsanity@linsanity-VirtualBox:~/Codes/Lab7$ 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] (32284) terminated by signal 2
tsh> ./myspin 3 &
[1] (32286) ./myspin 3 &
tsh> ./myspin 4 &
[2] (32288) ./myspin 4 &
tsh> jobs
[1] (32286) Running ./myspin 3 &
[2] (32288) Running ./myspin 4 &
tsh> fg %1
Job [1] (32286) stopped by signal 20
tsh> jobs
[1] (32286) Stopped ./myspin 3 &
[2] (32288) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (32286) ./myspin 3 &
tsh> jobs
[1] (32286) Running ./myspin 3 &
[2] (32288) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

测试结论	相同

4.4 自测试评分

根据节 4.3 的自测试结果，程序的测试评分为：　　15　。

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

收获了关于 shell 程序的整体框架搭建的相关知识，了解了关于异常控制流和进程的相关知识，知道了基本的信号处理的方式和流程。

5.2 请给出对本次实验内容的建议

建议增加更多关于 shell 框架以及相关函数实现的讲解，以及先关驱动程序 driver 以及 Makefile 等程序的知识。

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.