

# 哈尔滨工业大学

# 实验报告

## 实验（七）

题    目 TinyShell  
微壳

专    业 计算机类

学    号 1190501614

班    级 1903006

学    生 cgh

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.06.05

计算机科学与技术学院

# 目 录

<b>第 1 章 实验基本信息</b> .....	<b>4 -</b>
1.1 实验目的.....	4 -
1.2 实验环境与工具 .....	4 -
1.2.1 硬件环境.....	4 -
1.2.2 软件环境.....	4 -
1.2.3 开发工具.....	4 -
1.3 实验预习 .....	4 -
<b>第 2 章 实验预习</b> .....	<b>5 -</b>
2.1 进程的概念、创建和回收方法（5 分） .....	5 -
2.2 信号的机制、种类（5 分） .....	5 -
2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分） .....	6 -
2.4 什么是 SHELL，功能和处理流程（5 分） .....	7 -
<b>第 3 章 TINY SHELL 的设计与实现</b> .....	<b>8 -</b>
3.1.1 VOID EVAL(CHAR *CMDLINE)函数（10 分） .....	8 -
3.1.2 INT BUILTIN_CMD(CHAR **ARGV)函数（5 分） .....	8 -
3.1.3 VOID DO_BGFG(CHAR **ARGV) 函数（5 分） .....	8 -
3.1.4 VOID WAITFG(PID_T PID) 函数（5 分） .....	9 -
3.1.5 VOID SIGCHLD_HANDLER(INT SIG) 函数（10 分） .....	9 -
<b>第 4 章 TINY SHELL 测试</b> .....	<b>27 -</b>
4.1 测试方法.....	27 -
4.2 测试结果评价 .....	27 -
4.3 自测试结果.....	27 -
4.3.1 测试用例 trace01.txt .....	27 -
4.3.2 测试用例 trace02.txt .....	28 -
4.3.3 测试用例 trace03.txt .....	28 -
4.3.4 测试用例 trace04.txt .....	28 -
4.3.5 测试用例 trace05.txt .....	28 -
4.3.6 测试用例 trace06.txt .....	29 -
4.3.7 测试用例 trace07.txt .....	29 -
4.3.8 测试用例 trace08.txt .....	29 -
4.3.9 测试用例 trace09.txt .....	30 -
4.3.10 测试用例 trace10.txt .....	30 -
4.3.11 测试用例 trace11.txt .....	30 -
4.3.12 测试用例 trace12.txt .....	31 -
4.3.13 测试用例 trace13.txt .....	31 -

4.3.14 测试用例 <i>trace14.txt</i> .....	- 32 -
4.3.15 测试用例 <i>trace15.txt</i> .....	- 32 -
<b>第 5 章 评测得分 .....</b>	<b>- 34 -</b>
<b>第 6 章 总结 .....</b>	<b>- 35 -</b>
5.1 请总结本次实验的收获 .....	- 35 -
5.2 请给出对本次实验内容的建议 .....	- 35 -
<b>参考文献 .....</b>	<b>- 36 -</b>

## 第 1 章 实验基本信息

### 1.1 实验目的

理解现代计算机系统进程与并发的基本知识  
掌握 linux 异常控制流和信号机制的基本原理和相关系统函数  
掌握 shell 的基本原理和实现方法  
深入理解 Linux 信号响应可能导致的并发冲突及解决方法  
培养 Linux 下的软件系统开发与测试能力

### 1.2 实验环境与工具

#### 1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

#### 1.2.2 软件环境

Windows7 64 位以上; VirtualBox/Vmware 11 以上; Ubuntu 16.04 LTS 64 位/  
优麒麟 64 位

#### 1.2.3 开发工具

Ubuntu

### 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（Process）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

进程创建，是指操作系统创建一个新的进程。UNIX 系统用 `fork()` 系统调用，而 windows 系统用 `CreatProcess()`。进程创建的时机有：

- （1）系统初始化。系统的调度进程创建 `init` 进程。
- （2）执行中的进程调用了 `fork()` 系统函数。程序中有 `fork()` 函数。
- （3）用户登录，用户命令请求创建进程。例如：用户双击一个图标。
- （4）一个批处理作业初始化。大型机、高性能计算机用户提交一个课题，则系统建立作业控制块，在作业调度后在系统内存中创建进程。

进程回收的工作方式是，在发生所配置的回收事件时，回收应用程序或应用程序池。回收 OOP 应用程序的原理很简单：IIS 调入 `COM+ RecycleProcess()` 方法，以请求回收由 `DLLhost.exe` 托管的应用程序进程。触发回收事件后，将发生下列事件：

创建新的进程，以为对被回收应用程序的新的应用程序请求提供服务。

给予被回收的进程一些时间，以便完成对旧的应用程序请求的处理。时间间隔是用 `ShutdownTimeLimit` 配置数据库关键字配置的。

如果达到 `ShutdownTimeLimit`，而旧的进程还没停止，那么 IIS 将终止旧进程。

### 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 来标识，每一个信号对应唯一的 ID。

发送信号：内核通过更新目的进程上下文的某种状态，发送一个信号给目的

进程。

接收信号：内核强迫目的进程以某种方式对信号的发送做出反应，则接收了这种信号。

信号的种类

序号	名称	默认行为	相应事件
1	SIGHUP	终止	终端线挂断
2	SIGINT	终止	来自键盘的中断
3	SIGQUIT	终止	来自键盘的退出
4	SIGILL	终止	非法指令
5	SIGTRAP	终止并转储内存 <sup>①</sup>	跟踪陷阱
6	SIGABRT	终止并转储内存 <sup>①</sup>	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 <sup>①</sup>	浮点异常
9	SIGKILL	终止 <sup>②</sup>	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 <sup>①</sup>	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 SIGCONT <sup>②</sup>	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）

进程可以通过 kill 函数向包括它本身在内的其他进程发送一个信号，如果程序没有发送这个信号的权限，对 kill 函数的调用就将失败，而失败的常见原因是目标进程由另一个用户所拥有。

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。显示阻塞进程：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。

处理程序的设置方法：

- 1) 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。
- 2) 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）

什么是 shell：Shell 是一个用 C 语言编写的程序，他是用户使用 Linux 的桥梁。Shell 既是一种命令语言，又是一种程序设计语言，Shell 是指一种应用程序。

功能：Shell 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。

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：对用户输入的命令行进行解析，对于命令（quit, jobs, bg, fg）立即执行，否则创建子进程并在该子进程的上下文中运行，前台任务需要等待它运行结束

参 数：char \*cmdline

处理流程：首先调用 `parseline` 函数解析命令行的内容，判断命令是否为空  
如果输入的不是内置函数或内置命令，则在 `jobs` 中创建子进程，在子进程的上下文中运行它  
否则立即执行该命令

要点分析：

1. 防止对全局变量修改，阻塞 `SIGCHLD`、`SIGINT`、`SIGSTP` 信号后再创建子进程，创建子进程后用 `addjob` 将子进程的 `job` 加入 `jobs`，用 `sigprocmask` 解除阻塞。
2. 子进程从父进程处继承了信号阻塞向量，子进程必须确保在执行新程序之前解除对 `SIGCHLD` 的阻塞。
3. 如果是前台任务需要调用 `waitfg` 等待前台任务结束

#### 3.1.2 int builtin\_cmd(char \*\*argv) 函数 (5 分)

函数功能：如果是内置命令则执行相应任务；如果是 quit 命令则退出；如果是 jobs 命令则列出任务列表，否则放回 0

参 数：char \*\*argv

处理流程：先比较 `argv[0]` 是否为 quit，如果是，则退出程序  
然后比较 `argv[0]` 是否为 fg 或 bg，如果是，则执行相应任务  
最后比较是否为 jobs，如果是，则列出 job 的信息  
否则返回 0

要点分析：注意判断 `argv[0]` 是否为空，

#### 3.1.3 void do\_bgfg(char \*\*argv) 函数 (5 分)



函数功能：实现内置命令 `bg` 和 `fg`

参 数：char \*\*argv

处理流程：1. 解析命令行，判断读入的格式然后获得 job 的 pid  
2. 如果是 `bg` 命令，向 job 的进程组发送 `SIGCONT` 信号，并将状态设为 `BG`，打印相关信息  
3. 如果是 `fg` 命令，向 job 的进程组发送 `SIGCONT` 信号，并将状态设为 `FG`，打印相关信息，调用 `waitfg` 等待前台任务结束

要点分析：注意命令是否为空  
注意判断 process 是否存在  
注意判断 job 是否存在  
注意输入参数的格式是否规范  
分类讨论 `bg` 和 `fg`

### 3.1.4 void waitfg(pid\_t pid) 函数（5 分）

函数功能：等待前台任务结束

参 数：pid\_t pid

处理流程：通过 `while` 循环实现，每次 `sleep(1)`，直到当前任务的 PID 不再是 pid

要点分析：调用 `fgpid` 函数得到当前任务的 PID

### 3.1.5 void sigchld\_handler(int sig) 函数（10 分）

函数功能：捕获 `SIGCHLD` 信号并处理僵尸子进程

参 数：int sig

处理流程：

1. 临时变量保存 `errno`
2. 如果子进程正常退出或调用 `exit` 退出，则阻塞信号后删除该 job，解除阻塞
3. 如果子进程以及停止则打印相关信息
4. 如果子进程终止，打印信息后阻塞信号删除该 job，然后解除阻塞
5. 通过 `while` 回收所有僵尸子进程

要点分析：注意保存 `errno`，防止被修改

通过调用 `WIFEXITED`，`WIFSTOPPED`，`WIFSIGNALED` 来判断子进程的返回状态

## 3.2 程序实现（tsh.c 的全部内容）（10 分）

```
/*
 * tsh - A tiny shell program with job control
 *
 * <陈广焕 1190501614>
 */
#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(argv[0]==NULL) //判断指令是否为空
    {
        return 0;
    }
}
```



```
    }
    if(!strcmp(argv[0],"quit"))    //退出程序
    {
        exit(0);
    }
    else if(!strcmp(argv[0],"bg") || (!strcmp(argv[0],"fg")))    //执行相
应任务
    {
        do_bgfg(argv);
        return 1;
    }
    else if(!strcmp(argv[0],"jobs"))    //列出所有任务
    {
        listjobs(jobs);
        return 1;
    }
    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)
{

```

```

while(fgpid(jobs) == pid)    //等待前台不是 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; //进程 pid
    sigset_t mask_whloe, prev;

    sigfillset(&mask_whloe);    //防止卡死
    //非挂起方式处理
    while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)    //回收
僵尸进程
    {
        if (WIFEXITED(status))    //正常结束退出
        {
            sigprocmask(SIG_BLOCK, &mask_whloe, &prev);    //阻塞信号
            deletejob(jobs, pid);    //删除任务
            sigprocmask(SIG_SETMASK, &prev, NULL);    //解除阻塞
        }

        if (WIFSTOPPED(status))    //任务停止
        {
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid),
pid, WSTOPSIG(status));
        }

        if (WIFSIGNALED(status))    //任务终止

```

```

        {
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(pid), pid, WTERMSIG(status));
            sigprocmask(SIG_BLOCK, &mask_whloee, &prev);    //阻塞
            deletejob(jobs, pid);
            sigprocmask(SIG_SETMASK, &prev, NULL);    //恢复
        }
    }
    errno = olderrno;    //恢复 errno
    return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenever 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==0)
    {
        return;
    }
    kill(-pid, SIGINT);    //向正在运行的程序发送 SIGINT
    errno = olderrno; //»Öžžerrno
}

/*
 * 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)
{
    pid_t pid = fgpid(jobs);

    if(pid==0)
    {
        return;
    }
    getjobpid(jobs, pid)->state = ST;
    kill(-pid, SIGTSTP);    //向正在运行的程序发送 SIGTSTP

```

```
    printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(pid), pid, sig
);
    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

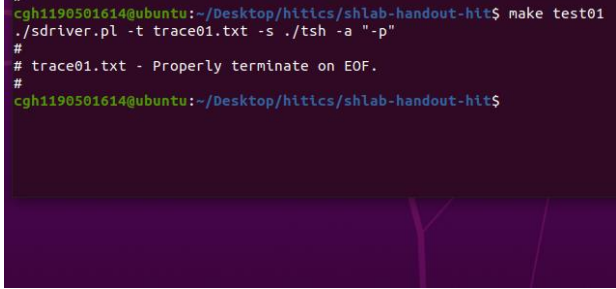
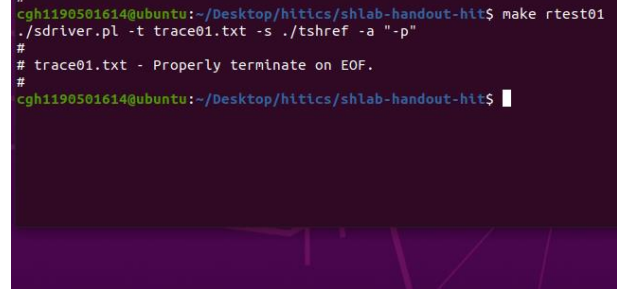
(2) 测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令, 每次运行的输出都会不同, 但每个 mysplrit 进程的运行状态应该相同。

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	
			
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
<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. # cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$</pre>	<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest02 ./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. #</pre>
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
<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh&gt; quit cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$</pre>	<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest03 ./sdriver.pl -t trace03.txt -s ./tshref -a "-p" # # trace03.txt - Run a foreground job. # tsh&gt; quit cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$</pre>
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
<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test04 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh&gt; ./myspin 1 &amp; [1] (2941) ./myspin 1 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$</pre>	<pre>cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest04 ./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh&gt; ./myspin 1 &amp; [1] (3041) ./myspin 1 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$</pre>
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
----------	-------------

<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test05 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh&gt; ./myspin 2 &amp; [1] (3198) ./myspin 2 &amp; tsh&gt; ./myspin 3 &amp; [2] (3200) ./myspin 3 &amp; tsh&gt; jobs [1] (3198) Running ./myspin 2 &amp; [2] (3200) Running ./myspin 3 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest05 ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh&gt; ./myspin 2 &amp; [1] (3333) ./myspin 2 &amp; tsh&gt; ./myspin 3 &amp; [2] (3335) ./myspin 3 &amp; tsh&gt; jobs [1] (3333) Running ./myspin 2 &amp; [2] (3335) Running ./myspin 3 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>
测试结论	相同

#### 4.3.6 测试用例 trace06.txt

<p style="text-align: center;">tsh 测试结果</p> <pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test06 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh&gt; ./myspin 4 Job [1] (3423) terminated by signal 2 cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>	<p style="text-align: center;">tshref 测试结果</p> <pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest06 ./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh&gt; ./myspin 4 Job [1] (3558) terminated by signal 2 cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>
测试结论	相同

#### 4.3.7 测试用例 trace07.txt

<p style="text-align: center;">tsh 测试结果</p> <pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test07 ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (3862) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (3864) terminated by signal 2 tsh&gt; jobs [1] (3862) Running ./myspin 4 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>	<p style="text-align: center;">tshref 测试结果</p> <pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (4061) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (4063) terminated by signal 2 tsh&gt; jobs [1] (4061) Running ./myspin 4 &amp; cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>
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
----------	-------------

<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test08 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (4169) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (4171) stopped by signal 20 tsh&gt; jobs Job [2] (4171) stopped by signal 20 [1] (4169) Running ./myspin 4 &amp; [2] (4171) Stopped ./myspin 5 cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest08 ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh&gt; ./myspin 4 &amp; [1] (4264) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (4266) stopped by signal 20 tsh&gt; jobs [1] (4264) Running ./myspin 4 &amp; [2] (4266) Stopped ./myspin 5 cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ </pre>
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
<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh&gt; ./myspin 4 &amp; [1] (4375) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (4377) stopped by signal 20 tsh&gt; jobs Job [2] (4377) stopped by signal 20 [1] (4375) Running ./myspin 4 &amp; [2] (4377) Stopped ./myspin 5 tsh&gt; bg %2 [2] (4377) ./myspin 5 tsh&gt; jobs [1] (4375) Running ./myspin 4 &amp; [2] (4377) Running ./myspin 5 </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh&gt; ./myspin 4 &amp; [1] (4470) ./myspin 4 &amp; tsh&gt; ./myspin 5 Job [2] (4472) stopped by signal 20 tsh&gt; jobs [1] (4470) Running ./myspin 4 &amp; [2] (4472) Stopped ./myspin 5 tsh&gt; bg %2 [2] (4472) ./myspin 5 tsh&gt; jobs [1] (4470) Running ./myspin 4 &amp; [2] (4472) Running ./myspin 5 </pre>
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
<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test10 ./sdriver.pl -t trace10.txt -s ./tsh -a "-p" # # trace10.txt - Process fg builtin command. # tsh&gt; ./myspin 4 &amp; [1] (4583) ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (4583) stopped by signal 20 tsh&gt; jobs Job [1] (4583) stopped by signal 20 [1] (4583) Stopped ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; jobs </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest10 ./sdriver.pl -t trace10.txt -s ./tshref -a "-p" # # trace10.txt - Process fg builtin command. # tsh&gt; ./myspin 4 &amp; [1] (4695) ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (4695) stopped by signal 20 tsh&gt; jobs [1] (4695) Stopped ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; jobs </pre>
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
----------	-------------

<pre> cgh1190501614ubuntu:~/Desktop/htics/shlab-handout-ht\$ make test11 ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh&gt; ./mysplit 4 Job [1] (4795) terminated by signal 2 tsh&gt; /bin/ps a # PID TTY STAT TIME COMMAND 1609 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 1615 tty2 Sl+ 0:09 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/g dm/Xauthority -background none -nolisten -keeptty -verbose 3 1646 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses sion=ubuntu 1729 tty2 Z+ 0:00 [fcitx] &lt;defunct&gt; 2578 pts/0 Ss 0:00 bash 4790 pts/0 S+ 0:00 make test11 4791 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 4792 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 4793 pts/0 S+ 0:00 ./tsh -p 4798 pts/0 R 0:00 /bin/ps a </pre>	<pre> cgh1190501614ubuntu:~/Desktop/htics/shlab-handout-ht\$ make rtest11 ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh&gt; ./mysplit 4 Job [1] (4912) terminated by signal 2 tsh&gt; /bin/ps a # PID TTY STAT TIME COMMAND 1609 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 1615 tty2 Sl+ 0:09 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd m/Xauthority -background none -nolisten -keeptty -verbose 3 1646 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses sion=ubuntu 1729 tty2 Z+ 0:00 [fcitx] &lt;defunct&gt; 2578 pts/0 Ss 0:00 bash 4907 pts/0 S+ 0:00 make rtest11 4908 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 4909 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 4910 pts/0 S+ 0:00 ./tshref -p 4918 pts/0 R 0:00 /bin/ps a </pre>
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
<pre> cgh1190501614ubuntu:~/Desktop/htics/shlab-handout-ht\$ make test12 ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" # # trace12.txt - Forward SIGTSTP to every process in foreground process group # tsh&gt; ./mysplit 4 Job [Job [1] (5054) stopped by signal 20 1) (5054) stopped by signal 20 tsh&gt; jobs [1] (5054) Stopped ./mysplit 4 tsh&gt; /bin/ps a # PID TTY STAT TIME COMMAND 1609 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 1615 tty2 Sl+ 0:09 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd m/Xauthority -background none -nolisten -keeptty -verbose 3 1646 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses sion=ubuntu 1729 tty2 Z+ 0:00 [fcitx] &lt;defunct&gt; 2578 pts/0 Ss 0:00 bash 5049 pts/0 S+ 0:00 make test12 5050 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" 5051 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" 5052 pts/0 S+ 0:00 ./tsh -p 5054 pts/0 T 0:00 ./mysplit 4 5055 pts/0 T 0:00 ./mysplit 4 5058 pts/0 R 0:00 /bin/ps a </pre>	<pre> cgh1190501614ubuntu:~/Desktop/htics/shlab-handout-ht\$ make rtest12 ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" # # trace12.txt - Forward SIGTSTP to every process in foreground process group # tsh&gt; ./mysplit 4 Job [1] (5330) stopped by signal 20 tsh&gt; jobs [1] (5330) Stopped ./mysplit 4 tsh&gt; /bin/ps a # PID TTY STAT TIME COMMAND 1609 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 1615 tty2 Sl+ 0:11 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd m/Xauthority -background none -nolisten -keeptty -verbose 3 1646 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses sion=ubuntu 1729 tty2 Z+ 0:00 [fcitx] &lt;defunct&gt; 2578 pts/0 Ss 0:00 bash 5325 pts/0 S+ 0:00 make rtest12 5326 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" 5327 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" 5328 pts/0 S+ 0:00 ./tshref -p 5330 pts/0 T 0:00 ./mysplit 4 5331 pts/0 T 0:00 ./mysplit 4 5337 pts/0 R 0:00 /bin/ps a </pre>
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
----------	-------------



## 计算机系统实验报告

<pre> cgh1190501614@ubuntu:~/Desktop/htics/shlab-handout-hit\$ make test13 ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" # # trace13.txt - Restart every stopped process in process group # tsh&gt; ./mysplit 4 Job [1] (5435) stopped by signal 20 tsh&gt; jobs Job [1] (5435) stopped by signal 20 [1] (5435) Stopped ./mysplit 4 tsh&gt; /bin/ps a   PID TTY          STAT TIME COMMAND 1609 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 1615 tty2    Sl+   0:11 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit y -background none -noreset -keeptty -verbose 3 1646 tty2    Sl+   0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1729 tty2    Z+    0:00 [fcitx] &lt;defunct&gt; 2578 pts/0  Ss    0:00 bash 5430 pts/0  S+    0:00 make test13 5431 pts/0  S+    0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 5432 pts/0  S+    0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 5433 pts/0  S+    0:00 ./tsh -p 5435 pts/0  T     0:00 ./mysplit 4 5436 pts/0  T     0:00 ./mysplit 4 5442 pts/0  R     0:00 /bin/ps a tsh&gt; fg %1 tsh&gt; /bin/ps a   PID TTY          STAT TIME COMMAND 1609 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 1615 tty2    Sl+   0:11 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit y -background none -noreset -keeptty -verbose 3 1646 tty2    Sl+   0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1729 tty2    Z+    0:00 [fcitx] &lt;defunct&gt; 2578 pts/0  Ss    0:00 bash 5430 pts/0  S+    0:00 make test13 5431 pts/0  S+    0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 5432 pts/0  S+    0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 5433 pts/0  S+    0:00 ./tsh -p 5440 pts/0  R     0:00 /bin/ps a </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/htics/shlab-handout-hit\$ make rtest13 ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" # # trace13.txt - Restart every stopped process in process group # tsh&gt; ./mysplit 4 Job [1] (5611) stopped by signal 20 tsh&gt; jobs [1] (5611) Stopped ./mysplit 4 tsh&gt; /bin/ps a   PID TTY          STAT TIME COMMAND 1609 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 1615 tty2    Sl+   0:13 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit y -background none -noreset -keeptty -verbose 3 1646 tty2    Sl+   0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1729 tty2    Z+    0:00 [fcitx] &lt;defunct&gt; 2578 pts/0  Ss    0:00 bash 5606 pts/0  S+    0:00 make rtest13 5607 pts/0  S+    0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 5608 pts/0  S+    0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 5609 pts/0  S+    0:00 ./tshref -p 5611 pts/0  T     0:00 ./mysplit 4 5612 pts/0  T     0:00 ./mysplit 4 5615 pts/0  R     0:00 /bin/ps a tsh&gt; fg %1 tsh&gt; /bin/ps a   PID TTY          STAT TIME COMMAND 1609 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 1615 tty2    Sl+   0:13 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthorit y -background none -noreset -keeptty -verbose 3 1646 tty2    Sl+   0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1729 tty2    Z+    0:00 [fcitx] &lt;defunct&gt; 2578 pts/0  Ss    0:00 bash 5606 pts/0  S+    0:00 make rtest13 5607 pts/0  S+    0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 5608 pts/0  S+    0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 5609 pts/0  S+    0:00 ./tshref -p 5621 pts/0  R     0:00 /bin/ps a </pre>
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
<pre> cgh1190501614@ubuntu:~/Desktop/htics/shlab-handout-hit\$ make test14 ./sdriver.pl -t trace14.txt -s ./tsh -a "-p" # # trace14.txt - Simple error handling # tsh&gt; ./bogus ./bogus: Command not found tsh&gt; ./myspin 4 &amp; [1] (5744) ./myspin 4 &amp; tsh&gt; fg fg command requires PID or %jobid argument tsh&gt; bg bg command requires PID or %jobid argument tsh&gt; fg a fg: argument must be a PID or %jobid tsh&gt; bg a bg: argument must be a PID or %jobid tsh&gt; fg 9999999 (9999999): No such process tsh&gt; bg 9999999 (9999999): No such process tsh&gt; fg %2 %2: No such job tsh&gt; fg %1 Job [1] (5744) stopped by signal 20 tsh&gt; bg %2 Job [1] (5744) stopped by signal 20 %2: No such job tsh&gt; bg %1 [1] (5744) ./myspin 4 &amp; tsh&gt; jobs [1] (5744) Running ./myspin 4 &amp; </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/htics/shlab-handout-hit\$ make rtest14 ./sdriver.pl -t trace14.txt -s ./tshref -a "-p" # # trace14.txt - Simple error handling # tsh&gt; ./bogus ./bogus: Command not found tsh&gt; ./myspin 4 &amp; [1] (6517) ./myspin 4 &amp; tsh&gt; fg fg command requires PID or %jobid argument tsh&gt; bg bg command requires PID or %jobid argument tsh&gt; fg a fg: argument must be a PID or %jobid tsh&gt; bg a bg: argument must be a PID or %jobid tsh&gt; fg 9999999 (9999999): No such process tsh&gt; bg 9999999 (9999999): No such process tsh&gt; fg %2 %2: No such job tsh&gt; fg %1 Job [1] (6517) stopped by signal 20 tsh&gt; bg %2 %2: No such job tsh&gt; bg %1 [1] (6517) ./myspin 4 &amp; tsh&gt; jobs [1] (6517) Running ./myspin 4 &amp; </pre>
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
----------	-------------



<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make test15 ./sdriver.pl -t trace15.txt -s ./tsh -a "-p" # # trace15.txt - Putting it all together # tsh&gt; ./bogus ./bogus: Command not found tsh&gt; ./myspin 10 Job [1] (6692) terminated by signal 2 tsh&gt; ./myspin 3 &amp; [1] (6694) ./myspin 3 &amp; tsh&gt; ./myspin 4 &amp; [2] (6696) ./myspin 4 &amp; tsh&gt; jobs [1] (6694) Running ./myspin 3 &amp; [2] (6696) Running ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (6694) stopped by signal 20 tsh&gt; jobs Job [1] (6694) stopped by signal 20 [1] (6694) Stopped ./myspin 3 &amp; [2] (6696) Running ./myspin 4 &amp; tsh&gt; bg %3 %3: No such job tsh&gt; bg %1 [1] (6694) ./myspin 3 &amp; tsh&gt; jobs [1] (6694) Running ./myspin 3 &amp; [2] (6696) Running ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; quit </pre>	<pre> cgh1190501614@ubuntu:~/Desktop/hitcs/shlab-handout-hit\$ make rtest15 ./sdriver.pl -t trace15.txt -s ./tshref -a "-p" # # trace15.txt - Putting it all together # tsh&gt; ./bogus ./bogus: Command not found tsh&gt; ./myspin 10 Job [1] (6837) terminated by signal 2 tsh&gt; ./myspin 3 &amp; [1] (6839) ./myspin 3 &amp; tsh&gt; ./myspin 4 &amp; [2] (6841) ./myspin 4 &amp; tsh&gt; jobs [1] (6839) Running ./myspin 3 &amp; [2] (6841) Running ./myspin 4 &amp; tsh&gt; fg %1 Job [1] (6839) stopped by signal 20 tsh&gt; jobs [1] (6839) Stopped ./myspin 3 &amp; [2] (6841) Running ./myspin 4 &amp; tsh&gt; bg %3 %3: No such job tsh&gt; bg %1 [1] (6839) ./myspin 3 &amp; tsh&gt; jobs [1] (6839) Running ./myspin 3 &amp; [2] (6841) Running ./myspin 4 &amp; tsh&gt; fg %1 tsh&gt; quit </pre>
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

通过本次实验对系统进程、异常控制流和信号机制有了更深的认识。

### 5.2 请给出对本次实验内容的建议

## 参考文献