

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 1190201308

班 级 1903006

学 生 陈东鑫

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 6 月 2 日

计算机科学与技术学院

目 录

第 1 章 实验基本信息.....	- 4 -
1.1 实验目的.....	- 4 -
1.2 实验环境与工具.....	- 4 -
1.2.1 硬件环境.....	错误! 未定义书签。
1.2.2 软件环境.....	错误! 未定义书签。
1.2.3 开发工具.....	错误! 未定义书签。
1.3 实验预习.....	- 4 -
第 2 章 实验预习.....	- 5 -
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）	- 8 -
第 3 章 TINY SHELL 的设计与实现.....	- 10 -
3.1.1 VOID EVAL(CHAR *CMDLINE)函数（10 分）	- 10 -
3.1.2 INT BUILTIN_CMD(CHAR **ARGV)函数（5 分）	- 10 -
3.1.3 VOID DO_BGFG(CHAR **ARGV) 函数（5 分）	- 10 -
3.1.4 VOID WAITFG(PID_T PID) 函数（5 分）	- 11 -
3.1.5 VOID SIGCHLD_HANDLER(INT SIG) 函数（10 分）	- 11 -
第 4 章 TINY SHELL 测试.....	- 26 -
4.1 测试方法.....	- 26 -
4.2 测试结果评价.....	- 26 -
4.3 自测试结果.....	- 26 -
4.3.1 测试用例 trace01.txt.....	- 26 -
4.3.2 测试用例 trace02.txt.....	- 27 -
4.3.3 测试用例 trace03.txt.....	- 27 -
4.3.4 测试用例 trace04.txt.....	- 27 -
4.3.5 测试用例 trace05.txt.....	- 27 -
4.3.6 测试用例 trace06.txt.....	- 28 -
4.3.7 测试用例 trace07.txt.....	- 28 -
4.3.8 测试用例 trace08.txt.....	- 28 -
4.3.9 测试用例 trace09.txt.....	- 29 -
4.3.10 测试用例 trace10.txt.....	- 29 -
4.3.11 测试用例 trace11.txt.....	- 29 -
4.3.12 测试用例 trace12.txt.....	- 30 -
4.3.13 测试用例 trace13.txt.....	- 30 -

4.3.14 测试用例 <i>trace14.txt</i>	- 31 -
4.3.15 测试用例 <i>trace15.txt</i>	- 31 -
4.4 自测试评分.....	错误！未定义书签。
第 5 章 总结	- 34 -
5.1 请总结本次实验的收获.....	- 34 -
5.2 请给出对本次实验内容的建议.....	- 34 -
参考文献	- 35 -

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; 1.60GHz; 8G RAM; 256GHD Disk。

1.2.2 软件环境

Windows10 64 位。

1.2.3 开发工具

VM VirtualBox 6.1; Ubuntu 20.04 LTS 64 位;
Visual Studio 2019 64 位; CodeBlocks 17.12 64 位; vi/vim/gedit+gcc。

1.3 实验预习

见第二章

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）

进程的经典定义是一个执行中程序的实例，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。

每次用户通过向 shell 输入一个可执行目标文件的名字，运行程序时，shell 会创建一个新的进程，然后在这个新进程的上下文中运行这个可执行目标文件。应用程序也能够创建新进程，并且在这个新进程的上下文中运行它们自己的代码或其他应用程序。

父进程通过调用 fork 函数创建一个新的运行的子进程。

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中，直到它被父进程回收。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。如果一个父进程终止了，内核会安排 init 进程成为它的孤儿进程的养父。

2.2 信号的机制、种类（5 分）

一个信号就剩一条小消息，它通知进程系统中发生了一个某种类型的事件。

每种信号类型都对应于某种系统事件，底层的硬件异常是由内核异常处理程序处理的，正常情况下，对用户进程而言是不可见的。信号提供了一种机制，通知用户进程发生了这些异常。

信号种类

序号	名称	默认行为	相应事件
1	SIGHUP	终止	终端线挂断
2	SIGINT	终止	来自键盘的中断
3	SIGQUIT	终止	来自键盘的退出
4	SIGILL	终止	非法指令
5	SIGTRAP	终止并转储内存 ^①	跟踪陷阱
6	SIGABRT	终止并转储内存 ^①	来自 abort 函数的终止信号
7	SIGBUS	终止	总线错误
8	SIGFPE	终止并转储内存 ^①	浮点异常
9	SIGKILL	终止 ^②	杀死程序
10	SIGUSR1	终止	用户定义的信号 1
11	SIGSEGV	终止并转储内存 ^①	无效的内存引用 (段故障)
12	SIGUSR2	终止	用户定义的信号 2
13	SIGPIPE	终止	向一个没有读用户的管道做写操作
14	SIGALRM	终止	来自 alarm 函数的定时器信号
15	SIGTERM	终止	软件终止信号
16	SIGSTKFLT	终止	协处理器上的栈故障
17	SIGCHLD	忽略	一个子进程停止或者终止
18	SIGCONT	忽略	继续进程如果该进程停止
19	SIGSTOP	停止直到下一个 SIGCONT ^②	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）

1. 发送方法

(1) 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。比如，命令

```
linux> /bin/kill -9 15213
```

发送信号 9(SIGKILL) 给进程 15213。一个为负的 PID 会导致信号被发送到进程组 PID 中的每个进程。比如，命令

```
linux> /bin/kill -9 -15213
```

发送一个 SIGKILL 信号给进程组 15213 中的每个进程。注意，在此我们使用

完整路径/bin / kill，因为有些 Unix shell 有自己内置的 kill 命令。

(2)从键盘发送信号

Unix shell 使用作业 (job) 这个抽象概念来表示为对一条命令行求值而创建的进程。在任何时刻，至多只有一个前台作业和 0 个或多个后台作业。比如，键入

```
linux> ls / sort
```

会创建一个由两个进程组成的前台作业，这两个进程是通过 Unix 管道连接起来的：一个进程运行 ls 程序，另一个运行 sort 程序。shell 为每个作业创建一个独立的进程组。进程组 ID 通常取自作业中父进程中的一个。

(3) kill 函数发送信号

(4)alarm 函数发送信号

2.阻塞方法

Linux 提供阻塞信号的隐式和显式的机制：

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型的待处理的信号。

显式阻塞机制：应用程序可以使用 sigprocmask 函数和它的辅助函数，明确地阻塞和解除阻塞选定的信号。

3.处理程序设置方法

signal 函数可以通过下列三种方法之一来改变和信号 signum 相关联的行为：

如果 handler 是 SIG_IGN，那么忽略类型为 signum 的信号。

如果 handler 是 SIG_DFL，那么类型为 signum 的信号行为恢复为默认行为。

否则，handler 就是用户定义的函数的地址，这个函数被称为信号处理程序，只要进程接收到一个类型为 signum 的信号，就会调用这个程序。通过把处理程序的地址传递到 signal 函数从而改变默认行为，这叫做设置信号处理程序(installing the handler)。调用信号处理程序被称为捕获信号。执行信号处理程序被称为处理信号。

当一个进程捕获了一个类型为 k 的信号时，就会调用为信号设置的处理程序，一个整数参数被设置为 k，这个参数允许同一个处理函数捕获不同类型的信号。

当处理程序执行它的 return 语句时，控制（通常）传递回控制流中进程被信号接收中断位置处的指令。我们说“通常”是因为在某些系统中，被中断的系统调用会立即返回一个错误。

2.4 什么是 shell，功能和处理流程（5 分）

在计算机科学中，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(命令解析器)。它类似于 DOS 下的 `command.com` 和后来的 `cmd.exe`。它接收用户命令，然后调用相应的应用程序。

shell 负责确保用户在命令提示符后输入的命令被正确执行。其功能包括：

- (1) 读取输入并解析命令行
- (2) 替换特殊字符，比如通配符和历史命令符
- (3) 设置管道、重定向和后台处理
- (4) 处理信号
- (5) 程式执行的相关设置

处理流程：

1. Shell 首先从命令行中找出特殊字符（元字符），在将元字符翻译成间隔符号。元字符将命令行划分成小块 `tokens`。Shell 中的元字符如下所示：

`SPACE , TAB , NEWLINE , & , ; , (,) , < , > , |`

2. 程序块 `tokens` 被处理，检查他们是否是 shell 中所引用到的关键字。

3. 当程序块 `tokens` 被确定以后，shell 根据 `aliases` 文件中的列表来检查命令的第一个单词。如果这个单词出现在 `aliases` 表中，执行替换操作并且处理过程回到第一步重新分割程序块 `tokens`。

4. Shell 对 `~` 符号进行替换。

5. Shell 对所有前面带有 `$` 符号的变量进行替换。

6. Shell 将命令行中的内嵌命令表达式替换成命令；他们一般都采用 `$(command)` 标记法。

7. Shell 计算采用 `$(expression)` 标记的算术表达式。

8. Shell 将命令字符串重新划分为新的块 `tokens`。这次划分的依据是栏位分割符号，称为 `IFS`。缺省的 `IFS` 变量包含有：`SPACE , TAB` 和换行符号。

9. Shell 执行通配符 `* ? []` 的替换。

10. shell 把所有从处理的结果中用到的注释删除，并且按照下面的顺序实行命令的检查：

A. 内建的命令

B. shell 函数（由用户自己定义的）

C.可执行的脚本文件（需要寻找文件和 PATH 路径）

11. 在执行前的最后一步是初始化所有的输入输出重定向。

12. 最后，执行命令。

第 3 章 TinyShell 的设计与实现

总分 45 分

3.1 设计

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

函数功能：处理用户输入的命令行。

参 数：用户输入的命令行。

处理流程：先使用 `parseline()` 分解命令行，得到有效的输入参数，然后使用 `builtin_cmd` 函数判断命令行是否为内置命令：若是，则使用 `builtin_cmd` 处理；若不是，则 `fork` 子进程然后执行 `execve` 创建新的进程。

要点分析：在 `eval` 中，父进程必须在用 `fork` 创建子进程前，使用 `sigprocmask` 阻塞 `SIGCHLD` 信号，防止父进程刚刚创建完成子进程，子进程在未被加入到 `joblist` 时就被回收，这会在后面在列表中删除子进程造成异常；父进程创建完成子进程并用 `addjob` 记录后，用 `sigprocmask` 解除阻塞。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。

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

函数功能：判断命令行输入是否为内置命令，若是，则直接进行处理，若能返回到 `eval` 则返回 1，若不是，则返回 0。

参 数：使用 `parse()` 分解命令行输入得到的参数

处理流程：分别判断命令行参数是否为内置命令，“quit” “bg” “fg” “jobs”，然后分别跳转到各自的执行程序：其中“quit”直接退出程序；“bg”和“fg”使用函数 `do_bgfg` 进行处理，处理完毕后返回 1；“jobs”使用 `job` 的辅助操作函数 `listjobs` 列出 `job` 列表，成功后返回 1；若不是这些内置函数则返回 0。

要点分析：非结束进程的处理程序结束后不要忘记 `return 1`。

不要在 `do_bgfg` 附近设计信号阻塞，这会阻止 `shell` 接受 `SIGCONT` 信号，从而导致命令 `fg, bg` 失效。

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

函数功能：处理内置命令 `bg,fg` 命令

参 数：使用 `parse()` 分解命令行输入得到的参数

处理流程：首先根据 % 后面的数字得到 `joblist` 中相应的进程，然后根据 `fg/bg` 命令进行不同的处理：若为 `fg` 则将该进程转为前台进程，即将进程的状态改为 `FG`，并显式地等待其结束；若为 `bg` 则将该进程转为后台进程，即将该进程地状态改为 `BG`，然后输出该进程状态。

要点分析：不要在 `do_bgfg` 附近设计信号阻塞，这会阻止 `shell` 接受 `SIGCONT` 信号，导致命令 `fg,bg` 失效。

3.1.4 void waitfg(pid_t pid) 函数 (5 分)

函数功能：显式地等待前台进程结束。

参 数：待等待的进程 `pid`。

处理流程：先记下该进程的 `job` 结构体地址，进行 `busy loop` 等待该地址下的 `job` 被 `clearjob` 清除。

要点分析：需要先记下等待进程的 `job` 结构体地址，否则若在 `while` 判断语句中使用 `getjobpid` 会导致在 `job` 被从 `joblist` 删除后找不到地址而出现段异常。

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

函数功能：`SIGCHLD` 信号处理程序，在接收到 `SIGCHLD` 后回收所有僵死子进程。

参 数：接收到的信号。

处理流程：首先使用 `while(waitpid(-1,&status,WNOHANG|WUNTRACED)>0)` 特定的顺序回收所有僵死子进程，然后在循环体内进行信号类型的判断和提示的输出。若进程由于正常退出而终止，则直接将其从 `joblist` 中删除；若进程由于其他信号而被杀死，则打印提示信息后将其从 `joblist` 中删除；若进程由于信号停止，则将其状态改为 `ST`。

要点分析：在进程回收时循环体内处理时需要阻塞相关信号，防止在更改 `job` 的相关成员信息时，由于其他信号的中断而造成错误。

不要使用 `unix_error` 处理 `errno=ECHILD` 的情况，我们持续对子进程进行回收，这必会导致回收结束时 `waitpid` 返回 -1，并设置 `errno` 为 `ECHILD`，若使用 `unix_error("waitpid error")` 必会主动造成 `waitpid error:interrupted system call` 异常的出现。

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
 *
 * <陈东鑫-1190201308>
 */
#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){
    sigset_t mask_all,mask_prev;
    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK,&mask_all,&mask_prev);

    if(!strcmp(argv[0],"quit")) /* quit command */
        exit(0);
    if(!strcmp(argv[0],"bg")||!strcmp(argv[0],"fg")){ /* fg & bg command
*/
        sigprocmask(SIG_SETMASK,&mask_prev,NULL);
        do_bgfg(argv);
        return 1;
    }
    if(!strcmp(argv[0],"jobs")){ /* jobs command */
        listjobs(jobs);
        sigprocmask(SIG_SETMASK,&mask_prev,NULL);
        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){
    struct job_t *orig_jobpid = getjobpid(jobs,pid);/* get pid of job to
wait for */
    while(orig_jobpid->state==FG){ /* busy loop until fg job finish */
        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 child_pid;

    sigset_t mask_all,mask_prev;
    sigfillset(&mask_all);

    while ((child_pid = waitpid(-1, &status, WNOHANG|WUNTRACED)) > 0){
        sigprocmask(SIG_BLOCK,&mask_all,&mask_prev); /* block signal which
can affect our conduction*/
        if(WIFEXITED(status)){/* jobs that exits from progress normally */
            if(!deletejob(jobs,child_pid))
                printf("Job [%d] (%d) delete job
error",pid2jid(child_pid),child_pid);
        }
        else if(WIFSIGNALED(status)){/* jobs that be killed by some signals
*/
            printf("Job [%d] (%d) terminated by
signal %d\n",pid2jid(child_pid),child_pid,WTERMSIG(status));
            if(!deletejob(jobs,child_pid))
                printf("Job [%d] (%d) delete job
error",pid2jid(child_pid),child_pid);
        }
        else if(WIFSTOPPED(status)){/* jobs that be stopped by some signal
*/
            getjobpid(jobs,child_pid)->state=ST;
        }

        else
            printf("Job [%d] (%d) terminated
abnormally\n",pid2jid(child_pid),child_pid);
        sigprocmask(SIG_SETMASK,&mask_prev,NULL);
    }

    errno = olderrno;
    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;

    sigset_t mask_all,mask_prev;
    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK,&mask_all,&mask_prev);

    pid_t fg_pid = fgpid(jobs);
    if (kill(-(fg_pid), sig) < 0)/* send signal to shell(-0) or jobs(-pid)*/
        unix_error("ctrl c error");

    sigprocmask(SIG_SETMASK,&mask_prev,NULL);
    errno = olderrno;
    return;
}
/*
* sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
*   the user types ctrl-z at the keyboard. Catch it and suspend the
*   foreground job by sending it a SIGTSTP.
*/void sigtstp_handler(int sig){
    int olderrno = errno;

    sigset_t mask_all,mask_prev;
    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK,&mask_all,&mask_prev);

    pid_t fg_pid = fgpid(jobs);
    if (kill(-(fg_pid), sig) < 0)/* send signal to shell(-0) or jobs(-pid)*/
        unix_error("ctrl z error");
    printf("Job [%d] (%d) stopped by
signal %d\n",pid2jid(fg_pid),fg_pid,sig);

    sigprocmask(SIG_SETMASK,&mask_prev,NULL);
    errno = olderrno;
    return;
}
/*****
* End signal handlers
*****/
/*****
* Helper routines that manipulate the job list
*****/

```

```
/* clearjob - Clear the entries in a job struct */void clearjob(struct job_t
*job) {
    job->pid = 0;
    job->jid = 0;
    job->state = UNDEF;
    job->cmdline[0] = '\0';
}
/* initjobs - Initialize the job list */void initjobs(struct job_t *jobs)
{
    int i;

    for (i = 0; i < MAXJOBS; i++)
        clearjob(&jobs[i]);
}
/* maxjid - Returns largest allocated job ID */int maxjid(struct job_t
*jobs){
    int i, max=0;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].jid > max)
            max = jobs[i].jid;
    return max;
}
/* addjob - Add a job to the job list */int addjob(struct job_t *jobs, pid_t
pid, int state, char *cmdline){
    int i;

    if (pid < 1)
        return 0;

    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid == 0) {
            jobs[i].pid = pid;
            jobs[i].state = state;
            jobs[i].jid = nextjid++;
            if (nextjid > MAXJOBS)
                nextjid = 1;
            strcpy(jobs[i].cmdline, cmdline);
            if(verbose){
                printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].pid,
jobs[i].cmdline);
            }
            return 1;
        }
    }
    printf("Tried to create too many jobs\n");
}
```

```
    return 0;
}
/* deletejob - Delete a job whose PID=pid from the job list */int
deletejob(struct job_t *jobs, pid_t pid){
    int i;

    if (pid < 1)
        return 0;

    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid == pid) {
            clearjob(&jobs[i]);
            nextjid = maxjid(jobs)+1;
            return 1;
        }
    }
    return 0;
}
/* fgpid - Return PID of current foreground job, 0 if no such job */pid_t
fgpid(struct job_t *jobs) {
    int i;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].state == FG)
            return jobs[i].pid;
    return 0;
}
/* getjobpid - Find a job (by PID) on the job list */struct job_t
*getjobpid(struct job_t *jobs, pid_t pid) {
    int i;

    if (pid < 1)
        return NULL;
    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].pid == pid)
            return &jobs[i];
    return NULL;
}
/* getjobjid - Find a job (by JID) on the job list */struct job_t
*getjobjid(struct job_t *jobs, int jid){
    int i;

    if (jid < 1)
        return NULL;
    for (i = 0; i < MAXJOBS; i++)
```

```

        if (jobs[i].jid == jid)
            return &jobs[i];
    return NULL;
}
/* pid2jid - Map process ID to job ID */int pid2jid(pid_t pid){
    int i;

    if (pid < 1)
        return 0;
    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].pid == pid) {
            return jobs[i].jid;
        }
    return 0;
}
/* listjobs - Print the job list */void listjobs(struct job_t *jobs){
    int i;

    for (i = 0; i < MAXJOBS; i++) {
        if (jobs[i].pid != 0) {
            printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
            switch (jobs[i].state) {
                case BG:
                    printf("Running ");
                    break;
                case FG:
                    printf("Foreground ");
                    break;
                case ST:
                    printf("Stopped ");
                    break;
                default:
                    printf("listjobs: Internal error: job[%d].state=%d ",
                           i, jobs[i].state);
            }
            printf("%s", jobs[i].cmdline);
        }
    }
}
/*****
 * end job list helper routines
 *****/

/*****
 * Other helper routines
 *****/

```



```
*****/
/*
 * usage - print a help message
 */void usage(void){
    printf("Usage: shell [-hvp]\n");
    printf("    -h    print this message\n");
    printf("    -v    print additional diagnostic information\n");
    printf("    -p    do not emit a command prompt\n");
    exit(1);
}
/*
 * unix_error - unix-style error routine
 */void unix_error(char *msg){
    fprintf(stdout, "%s: %s\n", msg, strerror(errno));
    exit(1);
}
/*
 * app_error - application-style error routine
 */void app_error(char *msg){
    fprintf(stdout, "%s\n", msg);
    exit(1);
}
/*
 * Signal - wrapper for the sigaction function
 */handler_t *Signal(int signum, handler_t *handler)
{
    struct sigaction action, old_action;

    action.sa_handler = handler;
    sigemptyset(&action.sa_mask); /* block sigs of type being handled */
    action.sa_flags = SA_RESTART; /* restart syscalls if possible */

    if (sigaction(signum, &action, &old_action) < 0)
        unix_error("Signal error");
    return (old_action.sa_handler);
}
/*
 * sigquit_handler - The driver program can gracefully terminate the
 * child shell by sending it a SIGQUIT signal.
 */void sigquit_handler(int sig){
    printf("Terminating after receipt of SIGQUIT signal\n");
    exit(1);
}
```

第 4 章 TinyShell 测试

总分 15 分

4.1 测试方法

针对 tsh 和参考 shell 程序 tshref, 完成测试项目 4.1-4.15 的对比测试, 并将测试结果截图或者通过重定向保存到文本文件(例如: `./sdriver.pl -t trace01.txt -s ./tsh -a "-p" > tshresult01.txt`), 并填写完成 4.3 节的相应表格。

4.2 测试结果评价

tsh 与 tshref 的输出在以下两个方面可以不同:

(1) pid


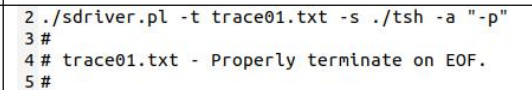
(2) 测试文件 trace11.txt, trace12.txt 和 trace13.txt 中的/bin/ps 命令, 每次运行的输出都会不同, 但每个 mysplit 进程的运行状态应该相同。

除了上述两方面允许的差异, tsh 与 tshref 的输出相同则判为正确, 如不同则给出原因分析。

4.3 自测试结果

填写以下各个测试用例的测试结果, 每个测试用例 1 分。

4.3.1 测试用例 trace01.txt

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> tshresult02.txt 1 # 2 # trace02.txt - Process builtin quit command. 3 # </pre>	<pre> 6 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" 7 # 8 # trace02.txt - Process builtin quit command. 9 # </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> tshresult03.txt 1 # 2 # trace03.txt - Run a foreground job. 3 # 4 tsh> quit </pre>	<pre> 10 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" 11 # 12 # trace03.txt - Run a foreground job. 13 # 14 tsh> quit </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> tshresult04.txt 1 # 2 # trace04.txt - Run a background job. 3 # 4 tsh> ./myspin 1 & 5 [1] (3395) ./myspin 1 & </pre>	<pre> 15 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" 16 # 17 # trace04.txt - Run a background job. 18 # 19 tsh> ./myspin 1 & 20 [1] (26252) ./myspin 1 & </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
----------	-------------

tshresult05.txt	
<pre> 1 # 2 # trace05.txt - Process jobs builtin command. 3 # 4 tsh> ./myspin 2 & 5 [1] (3399) ./myspin 2 & 6 tsh> ./myspin 3 & 7 [2] (3401) ./myspin 3 & 8 tsh> jobs 9 [1] (3399) Running ./myspin 2 & 10 [2] (3401) Running ./myspin 3 & </pre>	<pre> 21 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" 22 # 23 # trace05.txt - Process jobs builtin command. 24 # 25 tsh> ./myspin 2 & 26 [1] (26256) ./myspin 2 & 27 tsh> ./myspin 3 & 28 [2] (26258) ./myspin 3 & 29 tsh> jobs 30 [1] (26256) Running ./myspin 2 & 31 [2] (26258) Running ./myspin 3 & </pre>
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
<pre> tshresult06.txt 1 # 2 # trace06.txt - Forward SIGINT to foreground job. 3 # 4 tsh> ./myspin 4 5 Job [1] (3408) terminated by signal 2 </pre>	<pre> 32 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" 33 # 34 # trace06.txt - Forward SIGINT to foreground job. 35 # 36 tsh> ./myspin 4 37 Job [1] (26263) terminated by signal 2 </pre>
测试结论	相同/不同，原因分析如下：

4.3.7 测试用例 trace07.txt

tsh 测试结果	tshref 测试结果
<pre> tshresult07.txt 1 # 2 # trace07.txt - Forward SIGINT only to foreground job. 3 # 4 tsh> ./myspin 4 & 5 [1] (3412) ./myspin 4 & 6 tsh> ./myspin 5 7 Job [2] (3414) terminated by signal 2 8 tsh> jobs 9 [1] (3412) Running ./myspin 4 & </pre>	<pre> 38 ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" 39 # 40 # trace07.txt - Forward SIGINT only to foreground job. 41 # 42 tsh> ./myspin 4 & 43 [1] (26267) ./myspin 4 & 44 tsh> ./myspin 5 45 Job [2] (26269) terminated by signal 2 46 tsh> jobs 47 [1] (26267) Running ./myspin 4 & </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
----------	-------------

tshresult08.txt	
<pre> 1# 2# trace08.txt - Forward SIGTSTP only to foreground job. 3# 4 tsh> ./myspin 4 & 5 [1] (3419) ./myspin 4 & 6 tsh> ./myspin 5 7 Job [2] (3421) stopped by signal 20 8 tsh> jobs 9 [1] (3419) Running ./myspin 4 & 10 [2] (3421) Stopped ./myspin 5 </pre>	<pre> 48 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" 49 # 50 # trace08.txt - Forward SIGTSTP only to foreground job. 51 # 52 tsh> ./myspin 4 & 53 [1] (26274) ./myspin 4 & 54 tsh> ./myspin 5 55 Job [2] (26276) stopped by signal 20 56 tsh> jobs 57 [1] (26274) Running ./myspin 4 & 58 [2] (26276) Stopped ./myspin 5 </pre>
测试结论	相同

4.3.9 测试用例 trace09.txt

tsh 测试结果	
<pre> 1# 2# trace09.txt - Process bg builtin command 3# 4 tsh> ./myspin 4 & 5 [1] (3427) ./myspin 4 & 6 tsh> ./myspin 5 7 Job [2] (3429) stopped by signal 20 8 tsh> jobs 9 [1] (3427) Running ./myspin 4 & 10 [2] (3429) Stopped ./myspin 5 11 tsh> bg %2 12 [2] (3429) ./myspin 5 13 tsh> jobs 14 [1] (3427) Running ./myspin 4 & 15 [2] (3429) Running ./myspin 5 </pre>	<pre> 59 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" 60 # 61 # trace09.txt - Process bg builtin command 62 # 63 tsh> ./myspin 4 & 64 [1] (26281) ./myspin 4 & 65 tsh> ./myspin 5 66 Job [2] (26283) stopped by signal 20 67 tsh> jobs 68 [1] (26281) Running ./myspin 4 & 69 [2] (26283) Stopped ./myspin 5 70 tsh> bg %2 71 [2] (26283) ./myspin 5 72 tsh> jobs 73 [1] (26281) Running ./myspin 4 & 74 [2] (26283) Running ./myspin 5 </pre>
测试结论	相同

4.3.10 测试用例 trace10.txt

tsh 测试结果	
<pre> 1# 2# trace10.txt - Process fg builtin command. 3# 4 tsh> ./myspin 4 & 5 [1] (3436) ./myspin 4 & 6 tsh> fg %1 7 Job [1] (3436) stopped by signal 20 8 tsh> jobs 9 [1] (3436) Stopped ./myspin 4 & 10 tsh> fg %1 11 tsh> jobs </pre>	<pre> 75 ./sdriver.pl -t trace10.txt -s ./tsh -a "-p" 76 # 77 # trace10.txt - Process fg builtin command. 78 # 79 tsh> ./myspin 4 & 80 [1] (26290) ./myspin 4 & 81 tsh> fg %1 82 Job [1] (26290) stopped by signal 20 83 tsh> jobs 84 [1] (26290) Stopped ./myspin 4 & 85 tsh> fg %1 86 tsh> 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>cdxi190201308@cdxi190201308-VirtualBox:~/Hitics/shlab\$ 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] (2074) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE 1347 tty2 Sl+ 0:00 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses 2047 pts/0 Ss 0:00 bash 2069 pts/0 S+ 0:00 make test11 2070 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 2071 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 2072 pts/0 S+ 0:00 ./tsh -p 2077 pts/0 R 0:00 /bin/ps a</pre>		<pre>cdxi190201308@cdxi190201308-VirtualBox:~/Hitics/shlab\$ 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] (2062) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE 1347 tty2 Sl+ 0:07 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses 2047 pts/0 Ss 0:00 bash 2057 pts/0 S+ 0:00 make rtest11 2058 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 2059 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 2060 pts/0 S+ 0:00 ./tshref -p 2065 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>cdxi190201308@cdxi190201308-VirtualBox:~/Hitics/shlab\$ 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] (2093) stopped by signal 20 tsh> jobs [1] (2093) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE 1347 tty2 Rl+ 0:09 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses 2047 pts/0 Ss 0:00 bash 2080 pts/0 S+ 0:00 make test12 2089 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" 2090 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" 2091 pts/0 S+ 0:00 ./tsh -p 2093 pts/0 T 0:00 ./mysplit 4 2094 pts/0 T 0:00 ./mysplit 4 2097 pts/0 R 0:00 /bin/ps a</pre>		<pre>cdxi190201308@cdxi190201308-VirtualBox:~/Hitics/shlab\$ 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] (2083) stopped by signal 20 tsh> jobs [1] (2083) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SE 1347 tty2 Sl+ 0:00 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gd 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --ses 2047 pts/0 Ss 0:00 bash 2078 pts/0 S+ 0:00 make rtest12 2079 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" 2080 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" 2081 pts/0 S+ 0:00 ./tshref -p 2083 pts/0 T 0:00 ./mysplit 4 2084 pts/0 T 0:00 ./mysplit 4 2087 pts/0 R 0:00 /bin/ps a</pre>	
测试结论	相同/不同，原因分析如下：		

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>cdx1190201308@cdx1190201308-VirtualBox:~/htlcs/shlab\$ 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] (2117) stopped by signal 20 tsh> jobs [1] (2117) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESS 1347 tty2 Rl+ 0:10 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --sess 2047 pts/0 Ss 0:00 bash 2112 pts/0 S+ 0:00 make test13 2113 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 2114 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 2115 pts/0 S+ 0:00 ./tsh -p 2117 pts/0 T 0:00 ./mysplit 4 2118 pts/0 T 0:00 ./mysplit 4 2121 pts/0 R 0:00 /bin/ps a tsh> fg %1 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESS 1347 tty2 Rl+ 0:10 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --sess 2047 pts/0 Ss 0:00 bash 2112 pts/0 S+ 0:00 make test13 2113 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 2114 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 2115 pts/0 S+ 0:00 ./tsh -p 2117 pts/0 T 0:00 ./mysplit 4 2118 pts/0 T 0:00 ./mysplit 4 2121 pts/0 R 0:00 /bin/ps a</pre>	<pre>cdx1190201308@cdx1190201308-VirtualBox:~/htlcs/shlab\$ 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] (2104) stopped by signal 20 tsh> jobs [1] (2104) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESS 1347 tty2 Sl+ 0:00 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --sess 2047 pts/0 Ss 0:00 bash 2099 pts/0 S+ 0:00 make rtest13 2100 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 2101 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 2102 pts/0 S+ 0:00 ./tshref -p 2104 pts/0 T 0:00 ./mysplit 4 2105 pts/0 T 0:00 ./mysplit 4 2108 pts/0 R 0:00 /bin/ps a tsh> fg %1 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1340 tty2 Ssl+ 0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESS 1347 tty2 Sl+ 0:00 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm 1402 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --sess 2047 pts/0 Ss 0:00 bash 2099 pts/0 S+ 0:00 make rtest13 2100 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 2101 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 2102 pts/0 S+ 0:00 ./tshref -p 2104 pts/0 T 0:00 ./mysplit 4 2105 pts/0 T 0:00 ./mysplit 4 2108 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>tshresult14.txt 1 # 2 # trace14.txt - Simple error handling 3 # 4 tsh> ./bogus 5 ./bogus: Command not found 6 tsh> ./myspin 4 & 7 [1] (3481) ./myspin 4 & 8 tsh> fg 9 fg command requires PID or %jobid argument 10 tsh> bg 11 bg command requires PID or %jobid argument 12 tsh> fg a 13 fg: argument must be a PID or %jobid 14 tsh> bg a 15 bg: argument must be a PID or %jobid 16 tsh> fg 9999999 17 (9999999): No such process 18 tsh> bg 9999999 19 (9999999): No such process 20 tsh> fg %2 21 %2: No such job 22 tsh> fg %1 23 Job [1] (3481) stopped by signal 20 24 tsh> bg %2 25 %2: No such job 26 tsh> bg %1 27 [1] (3481) ./myspin 4 & 28 tsh> jobs 29 [1] (3481) Running ./myspin 4 &</pre>	<pre>150 ./sdriver.pl -t trace14.txt -s ./tsh -a "-p" 151 # 152 # trace14.txt - Simple error handling 153 # 154 tsh> ./bogus 155 ./bogus: Command not found 156 tsh> ./myspin 4 & 157 [1] (26326) ./myspin 4 & 158 tsh> fg 159 fg command requires PID or %jobid argument 160 tsh> bg 161 bg command requires PID or %jobid argument 162 tsh> fg a 163 fg: argument must be a PID or %jobid 164 tsh> bg a 165 bg: argument must be a PID or %jobid 166 tsh> fg 9999999 167 (9999999): No such process 168 tsh> bg 9999999 169 (9999999): No such process 170 tsh> fg %2 171 %2: No such job 172 tsh> fg %1 173 Job [1] (26326) stopped by signal 20 174 tsh> bg %2 175 %2: No such job 176 tsh> bg %1 177 [1] (26326) ./myspin 4 & 178 tsh> jobs 179 [1] (26326) Running ./myspin 4 &</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
----------	-------------

tshresult15.txt	
<pre> 1 # 2 # trace15.txt - Putting it all together 3 # 4 tsh> ./bogus 5 ./bogus: Command not found 6 tsh> ./myspin 10 7 Job [1] (3498) terminated by signal 2 8 tsh> ./myspin 3 & 9 [1] (3500) ./myspin 3 & 10 tsh> ./myspin 4 & 11 [2] (3502) ./myspin 4 & 12 tsh> jobs 13 [1] (3500) Running ./myspin 3 & 14 [2] (3502) Running ./myspin 4 & 15 tsh> fg %1 16 Job [1] (3500) stopped by signal 20 17 tsh> jobs 18 [1] (3500) Stopped ./myspin 3 & 19 [2] (3502) Running ./myspin 4 & 20 tsh> bg %3 21 %3: No such job 22 tsh> bg %1 23 [1] (3500) ./myspin 3 & 24 tsh> jobs 25 [1] (3500) Running ./myspin 3 & 26 [2] (3502) Running ./myspin 4 & 27 tsh> fg %1 28 tsh> quit </pre>	<pre> 180 ./sdriver.pl -t trace15.txt -s ./tsh -a "-p" 181 # 182 # trace15.txt - Putting it all together 183 # 184 tsh> ./bogus 185 ./bogus: Command not found 186 tsh> ./myspin 10 187 Job [1] (26343) terminated by signal 2 188 tsh> ./myspin 3 & 189 [1] (26345) ./myspin 3 & 190 tsh> ./myspin 4 & 191 [2] (26347) ./myspin 4 & 192 tsh> jobs 193 [1] (26345) Running ./myspin 3 & 194 [2] (26347) Running ./myspin 4 & 195 tsh> fg %1 196 Job [1] (26345) stopped by signal 20 197 tsh> jobs 198 [1] (26345) Stopped ./myspin 3 & 199 [2] (26347) Running ./myspin 4 & 200 tsh> bg %3 201 %3: No such job 202 tsh> bg %1 203 [1] (26345) ./myspin 3 & 204 tsh> jobs 205 [1] (26345) Running ./myspin 3 & 206 [2] (26347) Running ./myspin 4 & 207 tsh> fg %1 208 tsh> 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

6.1 请总结本次实验的收获

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。

6.2 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

- [1] lesliefish . wait 获取子进程退出状态 WIFEXITED 和 WIFSIGNALED 用法 [EB/OL].<https://blog.csdn.net/y396397735/article/details/53769865> ,2016-12-20/2021-06-06.
- [2] astrotycoon . wait 函数返回值总结 [EB/OL].<https://blog.csdn.net/astrotycoon/article/details/41172389> ,2014-11-16 /2021-06-06
- [3] sunblackshine . 父子进程 signal 出现 Interrupted system call 问题[EB/OL]. <https://www.cnblogs.com/sunblackshine/archive/2011/02/09/1950282.html> , 2011-02-09 /2021-06-06.
- [5] langzi989 .Unix 错误处理 [EB/OL]. <https://blog.csdn.net/u014630623/article/details/89067574> , 2019-04-07/2021-06-06 .
- [6] ahlbd . strchr[DB/OL]. <https://baike.so.com/doc/1880945-1989967.html> , 2018-08-26/2021-06-06.