哈爾濱工業大學

实验报告

实 验（七）

题　　目 TinyShell

微壳

专　　业 计算机类

学　　号 1190200128

班　　级 1903001

学　　生 詹先佑

指 导 教 师 郑贵滨

实 验 地 点 G712

实 验 日 期 2021 年 6 月 4 日

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

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

无。

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）

答：概念：一个正在运行的程序的实例。它提供给应用程序两个关键抽象：逻辑控制流、私有地址空间。

创建方法：父进程通过调用 fork 函数创建一个新的、处于运行状态的子进程。新创建的子进程几乎但不完全与父进程相同，子进程得到与父进程虚拟地址空间相同的但是独立的一份副本，子进程获得与父进程任何打开文件描述符相同的副本，最大区别是子进程有不同于父进程的 PID。

回收方法：

●父进程使用函数 wait 或者 waitpid 执行回收；

●父进程收到子进程的退出状态；

●内核删掉僵死的子进程。

2.2 信号的机制、种类（5 分）

信号机制：

Signal 就是一条小消息，它通知进程系统中发生了一个某种类型的事件，类似于异常和中断，从内核发送到（有时是在另一个进程的请求下）一个进程。在整个机制中包括了发送信号、接受信号、待处理位、待处理信号和阻塞信号。

发送信号是内核通过更新目的进程上下文中的某个状态， 发送（递送）一个信号给目的进程。发送信号可以是如下原因之一：内核检测到一个系统事件如除零错误(SIGFPE)或者子进程终止(SIGCHLD)；一个进程调用了 kill 系统调用，显式地请求内核发送一个信号到目的进程。

接收信号是当目的进程被内核强迫以某种方式对发送来的信号做出反应时，它就接收了信号。接收信号的反应方式包括了忽略这个信号(do nothing)；终止进

程(with optional core dump)；通过执行一个称为信号处理程序（signal handler） 的用户层函数捕获这个信号。

待处理位和阻塞位：是内核为每个进程维护着待处理位向量（pending）和阻塞位向量（blocked）。pending: 待处理信号集合，也称未决信号集合。若传送了一个类型为 k 的信号，内核会设置 pending 中的第 k 位（注册），若接收了(开始处理）一个类型为 k 的信号，内核将清除 pending 中的第 k 位。blocked: 被阻塞信号的集合，通过 sigprocmask 函数设置和清除

待处理信号和阻塞信号：一个发出而没有被接收的信号叫做待处理信号，也

称：未决信号。任何时刻，一种类型至多只有一个待处理(非实时的)信号。如果一个进程有一个类型为 k 的待处理信号，那么任何接下来发送到这个进程的类型为 k 的信号都会被丢弃。一个待处理信号最多只能被接收一次。一个进程可以选择阻塞接收某种信号，阻塞的信号仍可以被发送，但不会被接收，直到进程取消对该信号的阻塞。

信号种类：

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP 6) SIGABRT
7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2
13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT 17) SIGCHLD
18) SIGCONT 19) SIGSTOP 20) SIGTSTP 21) SIGTTIN 22) SIGTTOU
23) SIGURG 24) SIGXCPU 25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF
28) SIGWINCH 29) SIGIO 30) SIGPWR 31) SIGSYS

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

信号的发送方法：

（1） 用 /bin/kill 程序发送信号，/bin/kill 程序可以向另外的进程或进程组发送任意的信号。

（2） 从键盘发送信号，输入 ctrl-c /ctrl-z 会导致内核发送一个 SIGINT/SIGTSTP 信号到前台进程组中的每个作业

（3） 用 kill 函数发送信号

信号的阻塞方法：

（1） 阻塞和解除阻塞信号。隐式阻塞机制：内核默认阻塞与当前正在处理信号类型相同的待处理信号。显示阻塞和解除阻塞机制：sigprocmask 函数及其辅助函数可以明确地阻塞/解除阻塞选定的信号。

（2） 临时阻塞接收信号。

处理程序的设置方法：

使用 signal 函数，设定指定信号的处理程序(地址)，从而改变默认行为。

signal 函数

- 函数原型：

handler_t *signal(int signum, handler_t *handler)

- 功能：

可以修改和信号相关联的默认行为

- handler 的不同取值:
- SIG_IGN: 忽略类型为 signum 的信号
- SIG_DFL: 类型为 signum 的信号行为恢复为默认行为
- handler 是用户自定义函数的地址， 这个函数称为信号处理程序

2.4 什么是 shell，功能和处理流程（5 分）

答：

概念：shell 是一个交互型应用级程序，代表用户运行其他程序

功能：

●可交互，和非交互的使用 shell。在交互式模式，shell 从键盘接收输入；在非交互式模式，shell 从文件中获取输入。

●shell 中可以同步和异步的执行命令。在同步模式，shell 要等命令执行完，才能接收下面的输入。在异步模式，命令运行的同时，shell 就可接收其它的输入。重定向功能，可以更细致的控制命令的输入输出。另外，shell 允许设置命令的运行环境。

●shell 提供了少量的内置命令，以便自身功能更加完备和高效。

●shell 除了执行命令，还提供了变量，流程控制，引用和函数等，类似高级语言一样，能编写功能丰富的程序。

●shell 强大的的交互性除了可编程，还体现在作业控制，命令行编辑，历史命令，和别名等方面。

处理流程：shell 首先检查命令是否是内部命令，若不是再检查是否是一个应用程序（这里的应用程序可以是 Linux 本身的实用程序，如 ls 和 rm，也可以是购买的商业程序，如 xv，或者是自由软件，如 emacs）。然后 shell 在搜索路径里寻找这些应用程序（搜索路径就是一个能找到可执行程序的目录列表）。如果键入的命令不是一个内部命令并且在路径里没有找到这个可执行文件，将会显示一条错误信息。如果能够成功找到命令，该内部命令或应用程序将被分解为系统调用并传给 Linux 内核。

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：如果用户输入了 quit、jobs、bg、fg 其中一个 shell 命令，就会立即执行。否则会调用 fork 函数产生一个子进程在子进程的上下文中运行任务。如果这个任务在前台运行，那么就要等待它终止然后才可以返回。

参　　数：char *cmdline

处理流程：

（1） 定义相关变量 argv[MAXARGS]、bg、pid、mask

（2） 调用 parseline 函数解析命令行并且构建 argv 数组

（3） 调用 builtin_cmd 函数判断输入的命令是否为内置命令，如 quit、jobs、bg、fg，如果是就立即执行

（4） 如果不是内置命令，就会调用 fork 函数产生一个子进程在子进程的上下文中运行任务

要点分析：

（1） 创建子进程并且运行命令的过程中需要将 SIGCHLD、SIGINT、SIGSTP 信号阻塞

（2） 在子进程中，因为子进程几乎是父进程的复制，所以也会从父进程那里得到阻塞信号集合，在执行新的命令之前是需要解除阻塞的

（3） 调用 setpgid(0, 0)是创建了一个虚拟的进程组，然后使用 execve 函数执行。

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

函数功能：如果用户输入了正确的内置命令就立即执行

参　　数：char **argv

处理流程：

（1） 设置 5 个变量数 flag<n>记录 strcmp 函数的返回值，如果是内置命令，strcmp 函数会返回 0

（2） 用 5 个 if 语句判断用户输入的是否符合内置命令，若变量 flag<n>==0，就代表输入的是内置命令，然后调用相关函数立即执行命令，如果不是则返回

要点分析：

（1） 只有输入的是 quit 时，就直接使用 exit(0)结束程序，其他的情况都需要返回值
（2） 需要注意忽略单独的”&”的情况

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

函数功能：执行 bg 和 fg 这两个内置命令
参　　数：char **argv
处理流程：
（1） 首先是错误提示，如果只输入了 bg 或者 fg，后面没有相应的信息，就会报错
（2） 分析 PID 和%JID 两个参数。如果正确输入了相应的 PID 或者%JID 就使用 getjobpid 函数得到 job
（3） 如果是 bg 命令，就会发送 SIGCONT 命令，并且把 state 设为 BG
（4） 如果是 fg 命令，也会发送 SIGCONT 命令，并且把 state 设为 FG，并且调用 waitfg 等待当前的 job 不在前台执行。
要点分析：
（1） 需要对错误的各种情况有全面的考虑，比如没有相应的进程、没有相应的任务、输入的命令有格式问题等等
（2） 如果是 fg 命令，就需要调用 waitfg 函数对信号进行阻塞，也就是对命令行 tsh 的阻塞，因为当前在前台需要显示的是 fg 命令的执行过程，而不是 tsh，直到当前这个 job 结束，不再是前台进程
（3） 获取进程号 pid 或者工作组号 jid 的方式是通过判断 fg 和 bg 后面是数字还是%后面加数字的形式

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：阻塞信号直到指定 pid 的进程不再是前台进程。
参　　数：pid_t pid
处理流程：设置一个 while 循环，只要前台任务的 PID 和当前传入 waitfg 函数的 pid 相等，就执行 sleep(1)操作，休眠一秒，达到等待的功能。
要点分析：不建议使用 pause（）函数，while 循环的内部如果只是使用 pause(), 程序就会等待很长时间才再次检查循环终止条件，然后程序的运行时间会特别长

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

函数功能：当子进程收到 SIGSTOP 或者 SIGTSTP 然后终止或者停止时，内核会

给 shell 发送一个 SIGCHLD 信号，程序会回收所以僵死子进程，但是不会等待正在运行的子程序终止。

参　　数：int sig

处理流程：

（1）首先我们需要保存 errno 的值，然后执行　sigfillset(&mask_all);将所有信号都添加到 mask_all 阻塞集合里面

（2）然后使用 while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)最大可能的回收子进程

（3）在循环中先阻塞信号，然后判断进程是正常终止的，还是停止状态，或者是被信号终止的，并且使用 deletejob(jobs, pid)回收进程，完成循环以后解除对信号的阻塞

（4）恢复 errno 的值

要点分析：

（1）在处理子进程的过程中，使用了 while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)语句，WNOHANG |WUNTRACED 表示立即返回，如果等待集合中没有进程被中止或停止返回 0，否则孩子返回进程的 pid。

（2）判断进程的三个状态时，使用了 WIFSIGNALED、WIFSTOPPED、WIFEXITED 进行判断。WIFSIGNALED(status): 子进程被信号终止返回 true；WIFSTOPPED(status):当子进程接收到停止信号时 true; WIFEXITED(status):子进程正常退出情况下为 true。

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
 *
 * <Put your name and login ID here>
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
```

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

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

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

/* Global variables */
extern char **environ;          /* defined in libc */
char prompt[] = "tsh> ";        /* command line prompt (DO NOT
CHANGE) */
int verbose = 0;                /* if true, print additional output */
int nextjid = 1;                /* next job ID to allocate */
char sbuf[MAXLINE];             /* for composing sprintf messages */

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

```
/* Function prototypes */

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

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

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

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

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

/*
 * main - The shell's main routine
 */
int main(int argc, char **argv)
{
    char c;
    char cmdline[MAXLINE];
```

```
    int emit_prompt = 1; /* emit prompt (default) */

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

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

    /* Install the signal handlers */

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

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

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

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

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

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

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

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

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

    if (!builtin_cmd(argv)) {

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

  if (sigemptyset(&mask) < 0)
      unix_error("sigemptyset error");
  if (sigaddset(&mask, SIGCHLD))
      unix_error("sigaddset error");
  if (sigaddset(&mask, SIGINT))
      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)
{
    int flag1,flag2,flag3,flag4,flag5;
    flag1 = strcmp(argv[0],"quit");//判断是否输入的是 quit
    flag2   =strcmp(argv[0],"fg");//判断是否输入的是 fg
     flag3   =strcmp(argv[0],"bg");//判断是否输入的是 bg
     flag4   =strcmp(argv[0],"jobs");//判断是否输入的是 jobs
     flag5 = strcmp(argv[0],"&");//判断是否输入的是&
    if(flag1 == 0)
    {
        exit(0); //如果输入的是 quit 则退出程序
    }
    if(flag2==0)
    {
        do_bgfg(argv) ;//如果输入的是 fg 则调用  do_bgfg 函数实现相
应的功能
        return 1;
    }
    if(flag3==0)
    {
        do_bgfg(argv) ;//如果输入的是 bg 则调用  do_bgfg 函数实现相
应的功能
        return 1;
    }
    if(flag4==0)
    {
```

```
        listjobs(jobs) ;//如果输入的是 jobs 则调用 listjobs 函数显示当前暂停的进程
        return 1;
    }
    if(flag5 == 0)//如果输入的是&则忽略它
    {
        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(pid==fgpid(jobs))//只要指定为 pid 的进程还在前台运行循环
就继续
    {
        sleep(1);//休眠 1 秒
    }
    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;
    sigset_t mask_all , prev_all;
    pid_t pid;
    sigfillset(&mask_all);
    int status;
    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) //只要有子进程被终止或者停止就回收这个进程
    {
        sigprocmask(SIG_BLOCK, &mask_all, &prev_all);//阻塞信号
            if (WIFSIGNALED(status)) //如果是被信号终止的子进程
            {
            printf("Job [%d] (%d) terminated by signal %d\n", pid2jid(status),pid,WTERMSIG(status));
            deletejob(jobs, pid);    //回收进程
             }
             if (WIFSTOPPED(status)) //如果是停止的子进程
             {
            printf("Job [%d] (%d) stopped by signal %d\n", pid2jid(status),pid, WSTOPSIG(status));
             }
             if(WIFEXITED(status))//如果是正常的终止的子进程
             {
             deletejob(jobs, pid); //回收进程
```

```
            }
        sigprocmask(SIG_SETMASK,&prev_all, NULL);//解除阻塞

    }
    errno = olderrno;
     return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *      user types ctrl-c at the keyboard.  Catch it and send it along
 *      to the foreground job.
 */
void sigint_handler(int sig)
{
    int olderrno = errno;
    sigset_t mask_all , prev_all;
    pid_t pid;
    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK, &mask_all, &prev_all);//阻塞信号
    pid = fgpid(jobs);//获得前台进程的 pid
    if(pid == 0)
    return ;
    sigprocmask(SIG_SETMASK,&prev_all, NULL);//解除阻塞
    kill(-pid, SIGINT);//发送信号给前台进程
    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 , prev_all;
    pid_t pid;
```

```
    sigfillset(&mask_all);
    sigprocmask(SIG_BLOCK, &mask_all, &prev_all);//阻塞信号
    pid = fgpid(jobs);//获得前台进程的 pid
    getjobpid(jobs,pid)->state = ST;
    if(pid == 0)
    return ;
    sigprocmask(SIG_SETMASK,&prev_all, NULL);//解除阻塞
    kill(-pid, SIGTSTP);//发送信号给前台进程
    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
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test01 ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest01 ./sdriver.pl -t trace01.txt -s ./tshref -a "-p" # # trace01.txt - Properly terminate on EOF. #
测试结论	相同/不同，原因分析如下：相同

4.3.2 测试用例 trace02.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. #	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest02 ./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. #
测试结论	相同/不同，原因分析如下：相同

4.3.3 测试用例 trace03.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest03 ./sdriver.pl -t trace03.txt -s ./tshref -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit
测试结论	相同/不同，原因分析如下：相同

4.3.4 测试用例 trace04.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test04 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (5750) ./myspin 1 &	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest04 ./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (5756) ./myspin 1 &
测试结论	相同/不同，原因分析如下：相同

4.3.5 测试用例 trace05.txt

tsh 测试结果	tshref 测试结果

zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test05 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (5772) ./myspin 2 & tsh> ./myspin 3 & [2] (5775) ./myspin 3 & tsh> jobs [1] (5772) Running ./myspin 2 & [2] (5775) Running ./myspin 3 &	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest05 ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (5782) ./myspin 2 & tsh> ./myspin 3 & [2] (5784) ./myspin 3 & tsh> jobs [1] (5782) Running ./myspin 2 & [2] (5784) Running ./myspin 3 &
测试结论	相同/不同，原因分析如下：相同

4.3.6 测试用例 trace06.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test06 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [0] (5792) terminated by signal 2	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest06 ./sdriver.pl -t trace06.txt -s ./tshref -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 Job [1] (5798) terminated by signal 2
测试结论	相同/不同，原因分析如下：相同

4.3.7 测试用例 trace07.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test07 ./sdriver.pl -t trace07.txt -s ./tsh -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (5815) ./myspin 4 & tsh> ./myspin 5 Job [0] (5817) terminated by signal 2 tsh> jobs [1] (5815) Running ./myspin 4 &	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest07 ./sdriver.pl -t trace07.txt -s ./tshref -a "-p" # # trace07.txt - Forward SIGINT only to foreground job. # tsh> ./myspin 4 & [1] (5824) ./myspin 4 & tsh> ./myspin 5 Job [2] (5826) terminated by signal 2 tsh> jobs [1] (5824) Running ./myspin 4 &
测试结论	相同/不同，原因分析如下：相同

4.3.8 测试用例 trace08.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test08 ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (3382) ./myspin 4 & tsh> ./myspin 5 tsh> jobs Job [2] (3384) stopped by signal 20 [1] (3382) Running ./myspin 4 & [2] (3384) Stopped ./myspin 5	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest08 ./sdriver.pl -t trace08.txt -s ./tshref -a "-p" # # trace08.txt - Forward SIGTSTP only to foreground job. # tsh> ./myspin 4 & [1] (5841) ./myspin 4 & tsh> ./myspin 5 Job [2] (5843) stopped by signal 20 tsh> jobs [1] (5841) Running ./myspin 4 & [2] (5843) Stopped ./myspin 5
测试结论	相同/不同，原因分析如下：相同

4.3.9 测试用例 trace09.txt

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (3400) ./myspin 4 & tsh> ./myspin 5 tsh> jobs Job [0] (3402) stopped by signal 20 [1] (3400) Running ./myspin 4 & [2] (3402) Stopped ./myspin 5 tsh> bg %2 [2] (3402) ./myspin 5 tsh> jobs [1] (3400) Running ./myspin 4 & [2] (3402) Running ./myspin 5	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (3411) ./myspin 4 & tsh> ./myspin 5 Job [2] (3413) stopped by signal 20 tsh> jobs [1] (3411) Running ./myspin 4 & [2] (3413) Stopped ./myspin 5 tsh> bg %2 [2] (3413) ./myspin 5 tsh> jobs [1] (3411) Running ./myspin 4 & [2] (3413) Running ./myspin 5
测试结论	相同/不同，原因分析如下：相同

4.3.10 测试用例 trace10.txt

tsh 测试结果	tshref 测试结果

zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test10 ./sdriver.pl -t trace10.txt -s ./tsh -a "-p" # # trace10.txt - Process fg builtin command. # tsh> ./myspin 4 & [1] (3422) ./myspin 4 & tsh> fg %1 tsh> jobs Job [0] (3422) stopped by signal 20 [1] (3422) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest10 ./sdriver.pl -t trace10.txt -s ./tshref -a "-p" # # trace10.txt - Process fg builtin command. # tsh> ./myspin 4 & [1] (3432) ./myspin 4 & tsh> fg %1 Job [1] (3432) stopped by signal 20 tsh> jobs [1] (3432) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs
测试结论	相同/不同，原因分析如下：相同

4.3.11 测试用例 trace11.txt

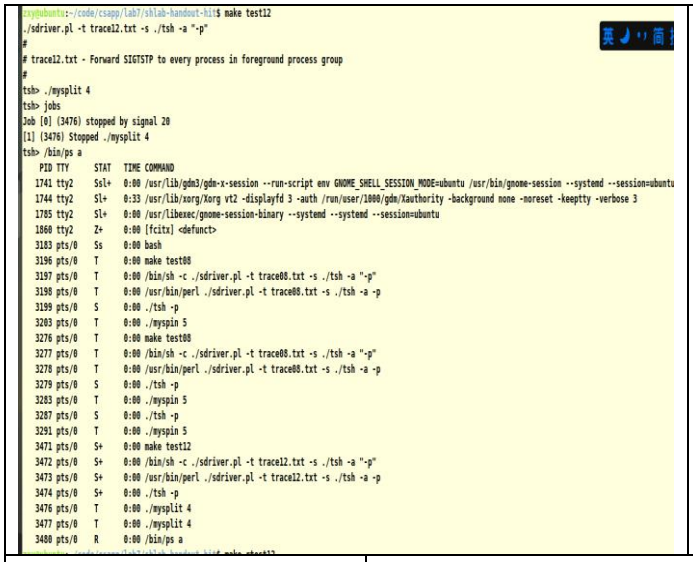
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test11 ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh> ./mysplit 4 Job [0] (3442) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1741 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 1744 tty2 Sl+ 0:26 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3 1785 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1860 tty2 Z+ 0:00 [fcitx] <defunct> 3183 pts/0 Ss 0:00 bash 3196 pts/0 T 0:00 make test08 3197 pts/0 T 0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" 3198 pts/0 T 0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p 3199 pts/0 S 0:00 ./tsh -p 3203 pts/0 T 0:00 ./myspin 5 3276 pts/0 T 0:00 make test08 3277 pts/0 T 0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" 3278 pts/0 T 0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p 3279 pts/0 S 0:00 ./tsh -p 3283 pts/0 T 0:00 ./myspin 5 3287 pts/0 S 0:00 ./tsh -p 3291 pts/0 T 0:00 ./myspin 5 3437 pts/0 S+ 0:00 make test11 3438 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 3439 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p 3440 pts/0 S+ 0:00 ./tsh -p 3445 pts/0 R 0:00 /bin/ps a	zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest11 ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" # # trace11.txt - Forward SIGINT to every process in foreground process group # tsh> ./mysplit 4 Job [1] (3452) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND 1741 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 1744 tty2 Rl+ 0:27 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3 1785 tty2 Sl+ 0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu 1860 tty2 Z+ 0:00 [fcitx] <defunct> 3183 pts/0 Ss 0:00 bash 3196 pts/0 T 0:00 make test08 3197 pts/0 T 0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" 3198 pts/0 T 0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p 3199 pts/0 S 0:00 ./tsh -p 3203 pts/0 T 0:00 ./myspin 5 3276 pts/0 T 0:00 make test08 3277 pts/0 T 0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p" 3278 pts/0 T 0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p 3279 pts/0 S 0:00 ./tsh -p 3283 pts/0 T 0:00 ./myspin 5 3287 pts/0 S 0:00 ./tsh -p 3291 pts/0 T 0:00 ./myspin 5 3447 pts/0 S+ 0:00 make rtest11 3448 pts/0 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 3449 pts/0 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p 3450 pts/0 S+ 0:00 ./tshref -p 3455 pts/0 R 0:00 /bin/ps a
测试结论	相同/不同，原因分析如下：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



```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
tsh> jobs
Job [0] (3476) stopped by signal 20
[1] (3476) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1741 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
   1744 tty2     Sl+    0:33 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1785 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
   1860 tty2     Z+     0:00 [fcitx] <defunct>
   3183 pts/0    Ss     0:00 bash
   3196 pts/0    T      0:00 make test08
   3197 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3198 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3199 pts/0    S      0:00 ./tsh -p
   3203 pts/0    T      0:00 ./myspin 5
   3276 pts/0    T      0:00 make test08
   3277 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3278 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3279 pts/0    S      0:00 ./tsh -p
   3283 pts/0    T      0:00 ./myspin 5
   3287 pts/0    S      0:00 ./tsh -p
   3291 pts/0    T      0:00 ./myspin 5
   3471 pts/0    S+     0:00 make test12
   3472 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
   3473 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
   3474 pts/0    S+     0:00 ./tsh -p
   3476 pts/0    T      0:00 ./mysplit 4
   3477 pts/0    T      0:00 ./mysplit 4
   3480 pts/0    R      0:00 /bin/ps a
```

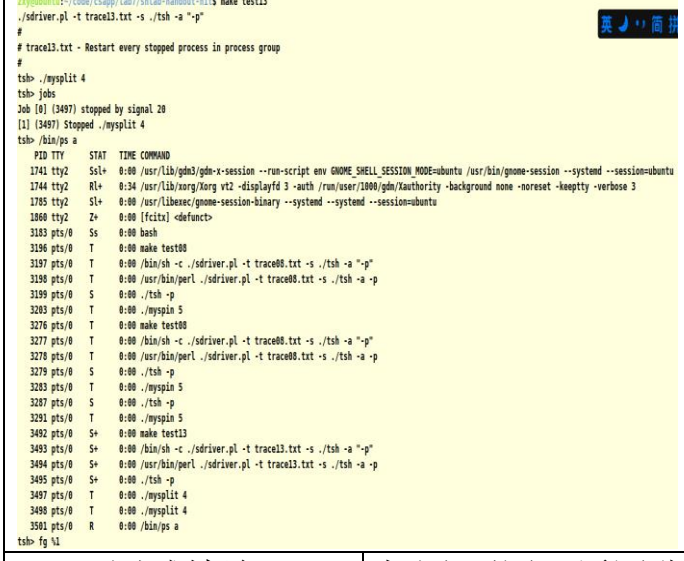
```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest12
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (3486) stopped by signal 20
tsh> jobs
[1] (3486) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1741 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
   1744 tty2     Sl+    0:33 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1785 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
   1860 tty2     Z+     0:00 [fcitx] <defunct>
   3183 pts/0    Ss     0:00 bash
   3196 pts/0    T      0:00 make test08
   3197 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3198 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3199 pts/0    S      0:00 ./tsh -p
   3203 pts/0    T      0:00 ./myspin 5
   3276 pts/0    T      0:00 make test08
   3277 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3278 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3279 pts/0    S      0:00 ./tsh -p
   3283 pts/0    T      0:00 ./myspin 5
   3287 pts/0    S      0:00 ./tsh -p
   3291 pts/0    T      0:00 ./myspin 5
   3481 pts/0    S+     0:00 make rtest12
   3482 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
   3483 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
   3484 pts/0    S+     0:00 ./tshref -p
   3486 pts/0    T      0:00 ./mysplit 4
   3487 pts/0    T      0:00 ./mysplit 4
   3490 pts/0    R      0:00 /bin/ps a
```

测试结论	相同/不同，原因分析如下：相同

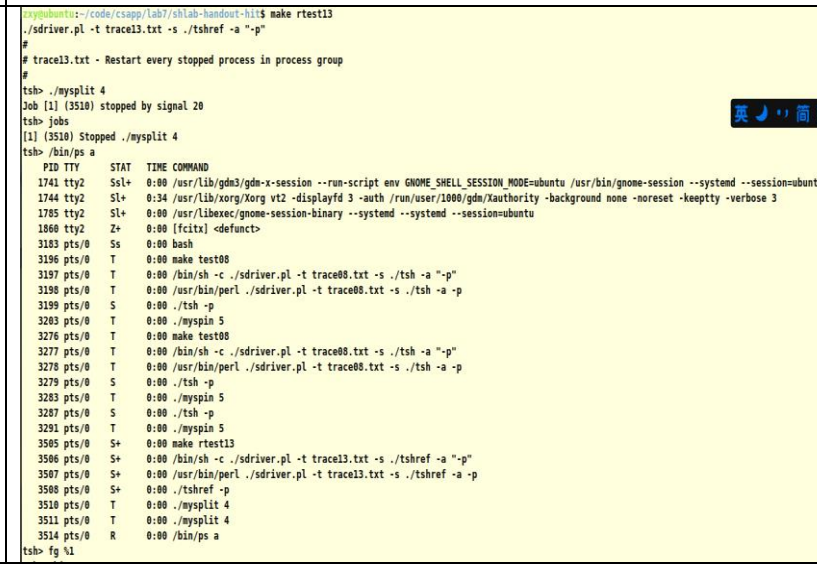
4.3.13 测试用例 trace13.txt

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



```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
tsh> jobs
Job [0] (3497) stopped by signal 20
[1] (3497) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1741 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
   1744 tty2     Rl+    0:34 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1785 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
   1860 tty2     Z+     0:00 [fcitx] <defunct>
   3183 pts/0    Ss     0:00 bash
   3196 pts/0    T      0:00 make test08
   3197 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3198 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3199 pts/0    S      0:00 ./tsh -p
   3203 pts/0    T      0:00 ./myspin 5
   3276 pts/0    T      0:00 make test08
   3277 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3278 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3279 pts/0    S      0:00 ./tsh -p
   3283 pts/0    T      0:00 ./myspin 5
   3287 pts/0    S      0:00 ./tsh -p
   3291 pts/0    T      0:00 ./myspin 5
   3492 pts/0    S+     0:00 make test13
   3493 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   3494 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   3495 pts/0    S+     0:00 ./tsh -p
   3497 pts/0    T      0:00 ./mysplit 4
   3498 pts/0    T      0:00 ./mysplit 4
   3501 pts/0    R      0:00 /bin/ps a
tsh> fg %1
```



```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (3510) stopped by signal 20
tsh> jobs
[1] (3510) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1741 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubunt
   1744 tty2     Sl+    0:34 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -background none -noreset -keeptty -verbose 3
   1785 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
   1860 tty2     Z+     0:00 [fcitx] <defunct>
   3183 pts/0    Ss     0:00 bash
   3196 pts/0    T      0:00 make test08
   3197 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3198 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3199 pts/0    S      0:00 ./tsh -p
   3203 pts/0    T      0:00 ./myspin 5
   3276 pts/0    T      0:00 make test08
   3277 pts/0    T      0:00 /bin/sh -c ./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
   3278 pts/0    T      0:00 /usr/bin/perl ./sdriver.pl -t trace08.txt -s ./tsh -a -p
   3279 pts/0    S      0:00 ./tsh -p
   3283 pts/0    T      0:00 ./myspin 5
   3287 pts/0    S      0:00 ./tsh -p
   3291 pts/0    T      0:00 ./myspin 5
   3505 pts/0    S+     0:00 make rtest13
   3506 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
   3507 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
   3508 pts/0    S+     0:00 ./tshref -p
   3510 pts/0    T      0:00 ./mysplit 4
   3511 pts/0    T      0:00 ./mysplit 4
   3514 pts/0    R      0:00 /bin/ps a
tsh> fg %1
```

测试结论	相同/不同，原因分析如下：相同

4.3.14 测试用例 trace14.txt

tsh 测试结果	tshref 测试结果

<table>
<tr>
<td>

```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ 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] (3531) ./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
tsh> bg %2
Job [0] (3531) stopped by signal 20
%2: No such job
tsh> bg %1
[1] (3531) ./myspin 4 &
tsh> jobs
[1] (3531) Running ./myspin 4 &
```

</td>
<td>

```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (3551) ./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] (3551) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (3551) ./myspin 4 &
tsh> jobs
[1] (3551) Running ./myspin 4 &
```

</td>
</tr>
<tr><td>测试结论</td><td colspan="2">相同/不同，原因分析如下：相同</td></tr>
</table>

4.3.15 测试用例 trace15.txt

tsh 测试结果	tshref 测试结果

```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [0] (3572) terminated by signal 2
tsh> ./myspin 3 &
[1] (3574) ./myspin 3 &
tsh> ./myspin 4 &
[2] (3576) ./myspin 4 &
tsh> jobs
[1] (3574) Running ./myspin 3 &
[2] (3576) Running ./myspin 4 &
tsh> fg %1
tsh> jobs
Job [0] (3574) stopped by signal 20
[1] (3574) Stopped ./myspin 3 &
[2] (3576) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (3574) ./myspin 3 &
tsh> jobs
[1] (3574) Running ./myspin 3 &
[2] (3576) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
zxy@ubuntu:~/code/csapp/lab7/shlab-handout-hit$ make rtest15
./sdriver.pl -t trace15.txt -s ./tshref -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (3613) terminated by signal 2
tsh> ./myspin 3 &
[1] (3615) ./myspin 3 &
tsh> ./myspin 4 &
[2] (3617) ./myspin 4 &
tsh> jobs
[1] (3615) Running ./myspin 3 &
[2] (3617) Running ./myspin 4 &
tsh> fg %1
Job [1] (3615) stopped by signal 20
tsh> jobs
[1] (3615) Stopped ./myspin 3 &
[2] (3617) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (3615) ./myspin 3 &
tsh> jobs
[1] (3615) Running ./myspin 3 &
[2] (3617) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

测试结论	相同/不同，原因分析如下：相同

第 5 章 评测得分

总分 20 分

实验程序统一测试的评分（教师评价）：

（1）正确性得分：_____（满分 10）

（2）性能加权得分：_____（满分 10）

第 6 章 总结

5.1 请总结本次实验的收获

（1）了解了信号的处理机制和进程之间的关系
（2）加强了对于 shell 的理解

5.2 请给出对本次实验内容的建议

无。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 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.