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　1190201816

班　　级　1903012

学　　生　樊红雨

指 导 教 师　史先俊

实 验 地 点　G709

实 验 日 期　2021.6.2

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

Visual Studio 2010 64 位以上；TestStudio；Gprof;Valgrind 等

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）

1.进程的概念：

在操作系统的角度来看进程是操作系统分配资源的最小单位。

简单来说进程就是处于执行期的程序(目标码存放在某种存储介质上)。但进程并不局限于一段可执行程序代码(代码段)。通常进程还要包含其他资源，像打开的文件(即在 Linux 中对应的文件描述符)，挂起的信号，内核内部数据，处理器的状态，一个或多个具有内存映射的内存地址空间及一个或多个执行线程，当然包括用来存放全局变量的数据段等。

2. 进程的创建：

进程的创建使用 fork()函数，该函数通过系统调用复制一个现有进程来创建一个全新的进程，调用 fork()的进程称为父进程，新产生的进程称为子进程。子进程中，fork 返回 0；父进程中，返回子进程的 PID；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。

3. 进程的回收：

当进程终止时，它仍然消耗系统资源，被称为“僵尸 zombie” 进程。父进程通过 wait 函数回收子进程，对于函数 int wait(int *child_status)： 挂起当前进程的执行直到它的一个子进程终止，返回已终止子进程的 PID。

2.2 信号的机制、种类（5 分）

1.信号的机制： 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 信号类型是用小整数 ID 来标识的(1-30) 信号中唯一的信息是它的 ID 和它的到达。 包括发送信号：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；接受信号：当目的进程被内核强迫以某种方式对信号的发送做 出反应时，它就接收了信号。 发送信号的原因：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。

反应的方式： 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。

2.信号种类：

编号	信号名称	缺省动作	说明
1	SIGHUP	终止	终止控制终端或进程
2	SIGINT	终止	键盘产生的中断(Ctrl-C)
3	SIGQUIT	dump	键盘产生的退出
4	SIGILL	dump	非法指令
5	SIGTRAP	dump	debug 中断
6	SIGABRT / SIGIOT	dump	异常中止
7	SIGBUS / SIGEMT	dump	总线异常/EMT 指令
8	SIGFPE	dump	浮点运算溢出
9	SIGKILL	终止	强制进程终止
10	SIGUSR1	终止	用户信号,进程可自定义用途
11	SIGSEGV	dump	非法内存地址引用
12	SIGUSR2	终止	用户信号，进程可自定义用途
13	SIGPIPE	终止	向某个没有读取的管道中写入数据
14	SIGALRM	终止	时钟中断(闹钟)
15	SIGTERM	终止	进程终止
16	SIGSTKFLT	终止	协处理器栈错误
17	SIGCHLD	忽略	子进程退出或中断
18	SIGCONT	继续	如进程停止状态则开始运行
19	SIGSTOP	停止	停止进程运行
20	SIGSTP	停止	键盘产生的停止
21	SIGTTIN	停止	后台进程请求输入
22	SIGTTOU	停止	后台进程请求输出
23	SIGURG	忽略	socket 发生紧急情况
24	SIGXCPU	dump	CPU 时间限制被打破
25	SIGXFSZ	dump	文件大小限制被打破

26	SIGVTALRM	终止	虚拟定时时钟
27	SIGPROF	终止	profile timer clock
28	SIGWINCH	忽略	窗口尺寸调整
29	SIGIO/SIGPOLL	终止	I/O 可用
30	SIGPWR	终止	电源异常
31	SIGSYS / SYSUNUSED	dump	系统调用异常

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

一. 发送信号

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 Examples：/bin/kill －9 24818 发送信号 9（SIGKILL）给进程 24818 /bin/kill －9 －24817 发送信号 SIGKILL 给进程组 24817 中的每个进程 （负的 PID 会导致信号被发送到进 程组 PID 中的每个进程）

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 SIGINT 默认情况是终止前台作业 SIGTSTP 默认情况是停止（挂起）前台作业。

3. 发送信号的函数主要有 kill(),raise(),alarm(),pause()

(1)kill()和 raise() kill()函数和熟知的 kill 系统命令一样，可以发送信号给信号和进程组（实际上 kill 系统命令只是 kill 函数的一个用户接口），需要注意的是他不仅可以终止进程(发送 SIGKILL 信号)，也可以向进程发送其他信号。与 kill 函数不同的是 raise()函数允许进程向自身发送信号。

(2)函数格式：kill 函数的语法格式：int kill(pid_t pid,int sig)，函数传入值为 sig 信号和 pid，返回值成功为 0，出错为-1。

raise()函数语法要点：

int raise(int sig),函数传入值为 sig 信号，返回值成功为 0，出错为-1

(3)alarm()和 pause() alarm()-----也称为闹钟函数，可以在进程中设置一个定时器，等到时间到达时，就会想进程发送 SIGALARM 信号，注意的是一个进程只能有一个闹钟时间，如果调用 alarm()之前已经设置了闹钟时间，那么任何以前的闹钟时间都会被新值所代替

pause()----此函数用于将进程挂起直到捕捉到信号为止，这个函数很常用，通 常用于判断信号是否已到

alarm()函数语法： unsigned int alram(unsigned int seconds)，函数传入值为 seconds 指定秒数，返回值如果成功且在调用函数前设置了闹钟时间，则返回闹钟时间的剩余时间，否则返回 0，出错返回-1

pause()函数语法如下： int pause(void)，返回-1，并且把 error 值设为 ETNTR

二. 阻塞信号

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 如： 一个 SIGINT 信号处理程序不能被另一个 SIGINT 信号中断 （此时另一个 SIGINT 信号被阻塞）

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

选定的信号辅助函数：

sigempty(set) - 初始化 set 为空集合

sigfill(set) - 把每个信号都添加到 set 中

sigadd(set) - 把指定的信号 signum 添加到 set

sigdel(set) - 从 set 中删除指定的信号 signum

三. 设置信号处理程序

可以使用 signal 函数修改和信号 signum 相关联的默认行为: handler_t *signal(int signum, handler_t *handler)

handler 的不同取值:

1. SIG_IGN: 忽略类型为 signum 的信号

2. SIG_DFL: 类型为 signum 的信号行为恢复为默认行为

3. 否则， handler 就是用户定义的函数的地址，这个函数称为信号处理程序

只要进程接收到类型为 signum 的信号就会调用信号处理程序

将处理程序的地址传递到 signal 函数从而改变默认行为，这叫作设置信号处理程序。调用信号处理程序称为捕获信号

执行信号处理程序称为处理信号

当处理程序执行 return 时，控制会传递到控制流中被信号接收所中断的指令处

2.4 什么是 shell，功能和处理流程（5 分）

1 定义：
shell 是一个交互型应用级程序，代表用户运行其他程序。是系统的用户界面，提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。
2.功能：
其实 shell 也是一支程序，它由输入设备读取命令，再将其转为计算机可以了解的机械码，然后执行它。各种操作系统都有它自己的 shell，以 DOS 为例，它的 shell 就是 command.com 文件。如同 DOS 下有 NDOS，4DOS，DRDOS 等不同的命令解译程序可以取代标准的 command.com ，UNIX 下除了 Bourne shell（/bin/sh）外还有 C shell（/bin/csh）、Korn shell（/bin/ksh）、Bourne again shell（/bin/bash）、Tenex C shell（tcsh）等其它的 shell。UNIX/linux 将 shell 独立于核心程序之外，使得它就如同一般的应用程序， 可以在不影响操作系统本身的情况下进行修改、更新版本或是添加新的功能。
Shell 是一个命令解释器，它解释由用户输入的命令并且把它们送到内核。不仅如

此，Shell 有自己的编程语言用于对命令的编辑，它允许用户编写由 shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果

3.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：解析和解释命令行的主例程
参　　数：命令行字符串 Char* cmdline
处理流程：

（1） 调用函数 parseline 处理 cmdline 字符串，将其转化为参数数组 argv。函数 parseline 返回 bg，判断是否后台进行。如果该任务是前台任务那么需要等到它运行结束才返回。

（2） 调用函数 buildin_cmd，判断输入是否为内建的命令行（quit，bg，fg，jobs），若是则立即执行。
若不是内建命令，fork 一个新的子进程并且将该任务在子进程的上下文中运行。

要点分析：

（1） 每个子进程必须拥有自己独一无二的进程组 ID。

（2） 但在收到 ctrl+c 或者 ctrl+z 时，子进程会被归为与 tsh 一个进程组中，一起被有关信号终止。

（3） 为防止出现子进程与信号处理进程竞争 jobs，在 fork 新进程前阻塞 SIGCHLD、SIGINT、SIGSTP 信号。

（4） 在父进程创建子进程前，用 sigprocmask 阻塞 ，在父进程创建子进程并 addjob 记录后，用 sigprocmask 解除阻塞。

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

函数功能：判断当前函数是否是内置函数，如果是则立即执行
参　　数：命令行参数 char **argv
处理流程：

（1） 判断 argv 是否为 quit ，若是则调用 exit(0) 。

（2） 判断 argv 是否为 bg 或 fg ，若是则调用函数 do_fgbg ，返回 1。

（3） 判断 argv 是否为 jobs ，若是则调用函数 listjobs ，返回 1。

（4） 若都不是，则返回 0。

要点分析：

该函数主要判断参数是否为内置命令，若是，则执行相应的操作。否则，返回 0.

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

函数功能：执行 bg、fg 的函数功能
参　　数：命令行参数 char **argv
处理流程：
（1） 解析传入命令行参数，判断 bg、fg 后读入 jobpid，调用 getjobpid 获得 job。
（2） 若为 bg，将该作业的状态修改为 BG，向进程组发送 SIGCONT 信号。
（3） 若为 fg，将该进程的状态修改为 FG，并调用 waitfg（）函数执行前台指令。
要点分析：
（1） 判断指令是否合法，检查输入合法性。
（2） 需要实时修改进程作业状态，向目标进程所在的进程组发送 SIGCONT 信号。
（3） 在 fg 命令中，需要调用 waitfg 阻塞 tsh 进行，使前台进程一直为该进程。

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：在 eval 函数中调用，用来等待前台子进程的完成。
参　　数：前台运行的作业进程的 pid
处理流程：
利用 fgpid 函数查找到当前在前台运行的作业进程的 pid，并与传入的 pid 进行比较。如果相等，则进行循环，并调用 sleep 函数达到等待的效果。
要点分析：
建议使用 busy loop：

- 在waitfg函数中，在sleep函数附近使用busy loop，例如：

```
while (xxxxx)  sleep(1);
```

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

函数功能：SIGCHLD 信号的响应函数
参　　数：传入信号 int sig
处理流程：
（1） 保存当前的 errno
（2） 处理停止或终止的子进程。
a) 如果子进程正常终止，则先阻塞信号，删除进程，再恢复信号。
b) 如果子进程已经停止则打印信息。
c) 如果子进程是因一个信号终止，信号未被捕获，则打印信息，阻塞信号，删除进程后恢复信号。
（3） 恢复（1）中保存的 errno。

要点分析：
（1） 运用 waitpid（）函数并且用 WNOHANG|WUNTRACED 作为参数，该参数

的作用是判断当前进程中是否存在已经停止或者终止的进程，如果存在则返回 pid，不存在则立即返回。

（2） 通过 waitpid 传入的 status，分别调用 WIFEXITED，WIFSTOPPED，WIFSIGNALED，判断子进程不同的返回状态。

（3） 保存恢复 errono。

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.
```

```
(38)  */
(39)
(40)  /* Global variables */
(41)  extern char **environ;      /* defined in libc */
(42)  char prompt[] = "tsh> ";    /* command line prompt (DO NOT CHANGE)
      */
(43)  int verbose = 0;            /* if true, print additional output */
(44)  int nextjid = 1;            /* next job ID to allocate */
(45)  char sbuf[MAXLINE];         /* for composing sprintf messages */
(46)
(47)  struct job_t {              /* The job struct */
(48)      pid_t pid;              /* job PID */
(49)      int jid;                /* job ID [1, 2, ...] */
(50)      int state;              /* UNDEF, BG, FG, or ST */
(51)      char cmdline[MAXLINE];  /* command line */
(52)  };
(53)  struct job_t jobs[MAXJOBS]; /* The job list */
(54)  /* End global variables */
(55)

(56)  /* Function prototypes */
(57)
(58)  /* Here are the functions that you will implement */
(59)  void eval(char *cmdline);
(60)  int builtin_cmd(char **argv);
(61)  void do_bgfg(char **argv);
(62)  void waitfg(pid_t pid);
(63)
(64)  void sigchld_handler(int sig);
(65)  void sigtstp_handler(int sig);
(66)  void sigint_handler(int sig);
(67)
(68)  /* Here are helper routines that we've provided for you */
(69)  int parseline(const char *cmdline, char **argv);
(70)  void sigquit_handler(int sig);
(71)
(72)  void clearjob(struct job_t *job);
(73)  void initjobs(struct job_t *jobs);
(74)  int maxjid(struct job_t *jobs);
(75)  int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline)
      ;
(76)  int deletejob(struct job_t *jobs, pid_t pid);
(77)  pid_t fgpid(struct job_t *jobs);
(78)  struct job_t *getjobpid(struct job_t *jobs, pid_t pid);
(79)  struct job_t *getjobjid(struct job_t *jobs, int jid);
```

```
(80)  int pid2jid(pid_t pid);
(81)  void listjobs(struct job_t *jobs);
(82)
(83)  void usage(void);
(84)  void unix_error(char *msg);
(85)  void app_error(char *msg);
(86)  typedef void handler_t(int);
(87)  handler_t *Signal(int signum, handler_t *handler);
(88)
(89)  /*
(90)   * main - The shell's main routine
(91)   */
(92)  int main(int argc, char **argv)
(93)  {
(94)      char c;
(95)      char cmdline[MAXLINE];
(96)      int emit_prompt = 1; /* emit prompt (default) */
(97)
(98)      /* Redirect stderr to stdout (so that driver will get all outpu
      t
(99)       * on the pipe connected to stdout) */
(100)     dup2(1, 2);
(101)
(102)     /* Parse the command line */
(103)     while ((c = getopt(argc, argv, "hvp")) != EOF) {
(104)         switch (c) {
(105)         case 'h':             /* print help message */
(106)             usage();
(107)         break;
(108)         case 'v':             /* emit additional diagnostic info *
      /
(109)             verbose = 1;
(110)         break;
(111)         case 'p':             /* don't print a prompt */
(112)             emit_prompt = 0;  /* handy for automatic testing */
(113)         break;
(114)     default:
(115)             usage();
(116)     }
(117)     }
(118)
(119)     /* Install the signal handlers */
(120)
(121)     /* These are the ones you will need to implement */
(122)     Signal(SIGINT,  sigint_handler);   /* ctrl-c */
```

```
(123)     Signal(SIGTSTP, sigtstp_handler);  /* ctrl-z */
(124)     Signal(SIGCHLD, sigchld_handler);  /* Terminated or stopped ch
      ild */
(125)
(126)     /* This one provides a clean way to kill the shell */
(127)     Signal(SIGQUIT, sigquit_handler);
(128)
(129)     /* Initialize the job list */
(130)     initjobs(jobs);
(131)
(132)     /* Execute the shell's read/eval loop */
(133)     while (1) {
(134)
(135)     /* Read command line */
(136)     if (emit_prompt) {
(137)         printf("%s", prompt);
(138)         fflush(stdout);
(139)     }
(140)     if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
(141)         app_error("fgets error");
(142)     if (feof(stdin)) { /* End of file (ctrl-d) */
(143)         fflush(stdout);
(144)         exit(0);
(145)     }
(146)
(147)     /* Evaluate the command line */
(148)     eval(cmdline);
(149)     fflush(stdout);
(150)     fflush(stdout);
(151)     }
(152)
(153)     exit(0); /* control never reaches here */
(154) }
(155)
(156) /*
(157)  * eval - Evaluate the command line that the user has just typed i
      n
(158)  *
(159)  * If the user has requested a built-in command (quit, jobs, bg or
       fg)
(160)  * then execute it immediately. Otherwise, fork a child process an
      d
(161)  * run the job in the context of the child. If the job is running
      in
```

```
(162)  * the foreground, wait for it to terminate and then return.  Note
       :
(163)  * each child process must have a unique process group ID so that
       our
(164)  * background children don't receive SIGINT (SIGTSTP) from the ker
       nel
(165)  * when we type ctrl-c (ctrl-z) at the keyboard.
(166)  */
(167) void eval(char *cmdline)
(168) {
(169)     /* $begin handout */
(170)     char *argv[MAXARGS]; /* argv for execve() */
(171)     int bg;              /* should the job run in bg or fg? */
(172)     pid_t pid;           /* process id */
(173)     sigset_t mask;       /* signal mask */
(174)
(175)     /* Parse command line */
(176)     bg = parseline(cmdline, argv);
(177)     if (argv[0] == NULL)
(178)     return;   /* ignore empty lines */
(179)
(180)     if (!builtin_cmd(argv)) {
(181)
(182)         /*
(183)      * This is a little tricky. Block SIGCHLD, SIGINT, and SIGTSTP
(184)      * signals until we can add the job to the job list. This
(185)      * eliminates some nasty races between adding a job to the job
(186)      * list and the arrival of SIGCHLD, SIGINT, and SIGTSTP signal
       s.
(187)      */
(188)
(189)     if (sigemptyset(&mask) < 0)
(190)         unix_error("sigemptyset error");
(191)     if (sigaddset(&mask, SIGCHLD))
(192)         unix_error("sigaddset error");
(193)     if (sigaddset(&mask, SIGINT))
(194)         unix_error("sigaddset error");
(195)     if (sigaddset(&mask, SIGTSTP))
(196)         unix_error("sigaddset error");
(197)     if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0)
(198)         unix_error("sigprocmask error");
(199)
(200)     /* Create a child process */
(201)     if ((pid = fork()) < 0)
(202)         unix_error("fork error");
```

```
(203)
(204)     /*
(205)      * Child  process
(206)      */
(207)
(208)     if (pid == 0) {
(209)         /* Child unblocks signals */
(210)         sigprocmask(SIG_UNBLOCK, &mask, NULL);
(211)
(212)         /* Each new job must get a new process group ID
(213)            so that the kernel doesn't send ctrl-c and ctrl-z
(214)            signals to all of the shell's jobs */
(215)         if (setpgid(0, 0) < 0)
(216)         unix_error("setpgid error");
(217)
(218)         /* Now load and run the program in the new job */
(219)         if (execve(argv[0], argv, environ) < 0) {
(220)         printf("%s: Command not found\n", argv[0]);
(221)         exit(0);
(222)         }
(223)     }
(224)
(225)     /*
(226)      * Parent process
(227)      */
(228)
(229)     /* Parent adds the job, and then unblocks signals so that
(230)        the signals handlers can run again */
(231)     addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
(232)     sigprocmask(SIG_UNBLOCK, &mask, NULL);
(233)
(234)     if (!bg)
(235)         waitfg(pid);
(236)     else
(237)         printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
(238)     }
(239)     /* $end handout */
(240)     return;
(241) }
(242)
(243) /*
(244)  * parseline - Parse the command line and build the argv array.
(245)  *
(246)  * Characters enclosed in single quotes are treated as a single
```

```
(247)  * argument.  Return true if the user has requested a BG job, fals
       e if
(248)  * the user has requested a FG job.
(249)  */
(250) int parseline(const char *cmdline, char **argv)
(251) {
(252)     static char array[MAXLINE]; /* holds local copy of command lin
       e */
(253)     char *buf = array;          /* ptr that traverses command line
       */
(254)     char *delim;                /* points to first space delimiter
       */
(255)     int argc;                   /* number of args */
(256)     int bg;                     /* background job? */
(257)
(258)     strcpy(buf, cmdline);
(259)     buf[strlen(buf)-1] = ' ';  /* replace trailing '\n' with space
       */
(260)     while (*buf && (*buf == ' ')) /* ignore leading spaces */
(261)     buf++;
(262)
(263)     /* Build the argv list */
(264)     argc = 0;
(265)     if (*buf == '\'') {
(266)     buf++;
(267)     delim = strchr(buf, '\'');
(268)     }
(269)     else {
(270)     delim = strchr(buf, ' ');
(271)     }
(272)
(273)     while (delim) {
(274)     argv[argc++] = buf;
(275)     *delim = '\0';
(276)     buf = delim + 1;
(277)     while (*buf && (*buf == ' ')) /* ignore spaces */
(278)            buf++;
(279)
(280)     if (*buf == '\'') {
(281)         buf++;
(282)         delim = strchr(buf, '\'');
(283)     }
(284)     else {
(285)         delim = strchr(buf, ' ');
(286)     }
```

```
(287)     }
(288)     argv[argc] = NULL;
(289)
(290)     if (argc == 0)  /* ignore blank line */
(291)     return 1;
(292)
(293)     /* should the job run in the background? */
(294)     if ((bg = (*argv[argc-1] == '&')) != 0) {
(295)     argv[--argc] = NULL;
(296)     }
(297)     return bg;
(298) }
(299)
(300) /*
(301)  * builtin_cmd - If the user has typed a built-in command then exe
      cute
(302)  *    it immediately.
(303)  */
(304) int builtin_cmd(char **argv)
(305) {
(306)     if (!strcmp(argv[0], "quit"))//如果命令行是内置 quit 命令,则直接执
      行退出
(307)         exit(0);
(308)     else if (!strcmp(argv[0], "&"))
(309)         return 1;//返回 1，标志着是内置命令
(310)     else if (!strcmp(argv[0], "bg")) {//判断命令行是否是 fg 或者 bg,若
      是则调用函数 do_bgfg(argv)，执行相关工作，并返回 1，标记为内置命令
(311)         do_bgfg(argv);
(312)         return 1;//返回 1，标志着是内置命令
(313)     }
(314)     else if (!strcmp(argv[0], "fg")) {
(315)         do_bgfg(argv);
(316)         return 1;//返回 1，标志着是内置命令
(317)     }
(318)     else if (!strcmp(argv[0], "jobs")) {//如果命令行是内置 jobs 命令,
      即显示当前暂停的进程,则调用函数 listjobs(jobs),并返回 1，标记为内置命令
(319)         listjobs(jobs);
(320)         return 1;//返回 1，标志着是内置命令
(321)     }
(322)     return 0;     /* not a builtin command */ //否则，返回 0，意味着
      非内置命令，此返回值便于直接判断是否是内置命令
(323) }
(324)
(325) /*
(326)  * do_bgfg - Execute the builtin bg and fg commands
```

```
(327)  */
(328) void do_bgfg(char **argv)
(329) {
(330)     /* $begin handout */
(331)     struct job_t *jobp=NULL;
(332)
(333)     /* Ignore command if no argument */
(334)     if (argv[1] == NULL) {
(335)     printf("%s command requires PID or %%jobid argument\n", argv[0
      ]);
(336)     return;
(337)     }
(338)
(339)     /* Parse the required PID or %JID arg */
(340)     if (isdigit(argv[1][0])) {
(341)     pid_t pid = atoi(argv[1]);
(342)     if (!(jobp = getjobpid(jobs, pid))) {
(343)         printf("(%d): No such process\n", pid);
(344)         return;
(345)     }
(346)     }
(347)     else if (argv[1][0] == '%') {
(348)     int jid = atoi(&argv[1][1]);
(349)     if (!(jobp = getjobjid(jobs, jid))) {
(350)         printf("%s: No such job\n", argv[1]);
(351)         return;
(352)     }
(353)     }
(354)     else {
(355)     printf("%s: argument must be a PID or %%jobid\n", argv[0]);
(356)     return;
(357)     }
(358)
(359)     /* bg command */
(360)     if (!strcmp(argv[0], "bg")) {
(361)     if (kill(-(jobp->pid), SIGCONT) < 0)
(362)         unix_error("kill (bg) error");
(363)     jobp->state = BG;
(364)     printf("[%d] (%d) %s", jobp->jid, jobp->pid, jobp->cmdline);
(365)     }
(366)
(367)     /* fg command */
(368)     else if (!strcmp(argv[0], "fg")) {
(369)     if (kill(-(jobp->pid), SIGCONT) < 0)
(370)         unix_error("kill (fg) error");
```

```
(371)     jobp->state = FG;
(372)     waitfg(jobp->pid);
(373)     }
(374)     else {
(375)     printf("do_bgfg: Internal error\n");
(376)     exit(0);
(377)     }
(378)     /* $end handout */
(379)     return;
(380) }
(381)
(382) /*
(383)  * waitfg - Block until process pid is no longer the foreground pr
      ocess
(384)  */
(385) void waitfg(pid_t pid)  //传入的参数为前台进程 pid
(386) {
(387)     while (pid == fgpid(jobs)) //循环阻塞信号，直到 pid 不再是前台进程
(388)     {
(389)         sleep(1);
(390)     }
(391)     return;
(392) }
(393)
(394) /*****************
(395)  * Signal handlers
(396)  *****************/
(397)
(398) /*
(399)  * sigchld_handler - The kernel sends a SIGCHLD to the shell whene
      ver
(400)  *     a child job terminates (becomes a zombie), or stops because
       it
(401)  *     received a SIGSTOP or SIGTSTP signal. The handler reaps all
(402)  *     available zombie children, but doesn't wait for any other
(403)  *     currently running children to terminate.
(404)  */
(405) void sigchld_handler(int sig)
(406) {
(407)     int status;
(408)     int olderrno=errno;//为防止错误的信号处理函数修改，用
      errnoolderrno 保存原来的 errno
(409)     pid_t pid;//保存引起进程终止的子进程 pid
(410)     sigset_t mask,prev;//mask_all 保存所有的信号，pre_all 保存原来信号
      阻塞向量中的所有信号
```

```
(411)     sigfillset(&mask);//将每个信号添加到 mask_all 中
(412)
(413)     while((pid = waitpid(-1,&status,WUNTRACED|WNOHANG))>0) {//立即
      返回，若等待集合中的子进程中都没有停止或终止
(414)         if(WIFSTOPPED(status))
(415)         {           //若引起进程返回的原因是子进程状态是停止的
(416)             struct job_t * job = getjobpid(jobs,pid);//根据获取的
      pid 查询前台任务
(417)             if(job&&job->state!=ST)//若 job 存在且还未收到改变其状态为
      ST 的处理
(418)                 printf("Job [%d] (%d) stopped by signal %d\n", pid
      2jid(pid), pid, WSTOPSIG(status));//终端输出打印
(419)         }
(420)
(421)         if(WIFSIGNALED(status))
(422)         {           //若引起进程返回的原因是子进程未捕捉信号终止
(423)             struct job_t * job = getjobpid(jobs,pid);//获取此子进程
      pid 所在的 job
(424)             if(job)
(425)             {       //若存在此 job
(426)                 printf("Job [%d] (%d) terminated by signal %d\n",
      pid2jid(pid), pid, WTERMSIG(status));//终端打印输出
(427)                 sigprocmask(SIG_BLOCK, &mask, &prev);//阻塞所有信
      号，因为下面要对全局数据结构进行访问
(428)                 deletejob(jobs, pid);//将 pid 所指的 job 从 job 结构数组
      中删除
(429)                 sigprocmask(SIG_SETMASK, &prev, NULL);//解除阻塞
(430)             }
(431)         }
(432)         if(WIFEXITED(status))
(433)         {           //若引起进程返回的原因是子进程调用 exit 或者 return
      正常终止
(434)             sigprocmask(SIG_BLOCK,&mask,&prev);//阻塞所有信号，因为下
      面要对全局数据结构进行访问
(435)             deletejob(jobs, pid);//将 pid 所指的 job 从 job 结构数组中删
      除
(436)             sigprocmask(SIG_SETMASK,&prev,NULL);//解除阻塞
(437)         }
(438)     }
(439)
(440)     errno=olderrno;//恢复原来的 errno
(441)     return;
(442) }
(443)
(444) /*
```

```
(445)  * sigint_handler - The kernel sends a SIGINT to the shell whenver
       the
(446)  *    user types ctrl-c at the keyboard.  Catch it and send it alo
       ng
(447)  *    to the foreground job.
(448)  */
(449) void sigint_handler(int sig)
(450) {
(451)     int olderrno = errno;//保存原来的 errno
(452)     sigset_t mask,pre;//mask 用于保存所有的信号，pre 用于保存原来信号阻
       塞向量中的所有信号
(453)     sigfillset(&mask);//将每个信号添加到 mask 中
(454)
(455)     sigprocmask(SIG_BLOCK,&mask,&pre);//阻塞所有信号,因为下面要对全局
       数据结构进行访问
(456)     pid_t pid = fgpid(jobs);//获取当前前台作业的 pid
(457)     if (pid > 0)
(458)     {
(459)         kill(-pid,SIGINT);//根据获取的 pid 查询前台任务
(460)         printf("Job [%d] (%d) terminated by signal %d\n",pid2jid(p
       id),pid,sig);//打印输出在终端
(461)         deletejob(jobs,pid);//将 pid 所指的 job 从 job 结构数组中删除
(462)     }
(463)     sigprocmask(SIG_SETMASK,&pre,NULL);//解除阻塞
(464)     errno = olderrno;//恢复原来的 errno
(465)     return;
(466) }
(467)
(468) /*
(469)  * sigtstp_handler - The kernel sends a SIGTSTP to the shell whene
       ver
(470)  *     the user types ctrl-z at the keyboard. Catch it and suspend
       the
(471)  *     foreground job by sending it a SIGTSTP.
(472)  */
(473) void sigtstp_handler(int sig)
(474) {
(475)     int olderrno = errno;//保存原来的 errno
(476)     sigset_t mask,pre;  //mask 用于保存所有的信号，pre 用于保存原来信号
       阻塞向量中的所有信号
(477)     sigfillset(&mask);  //将每个信号添加到 mask 中
(478)
(479)     sigprocmask(SIG_BLOCK,&mask,&pre);//阻塞所有信号,因为下面要对全局
       数据结构进行访问
(480)     pid_t pid = fgpid(jobs);//获取当前前台作业的 pid
```

```
(481)     if (pid > 0)
(482)     {
(483)         struct job_t * job = getjobpid(jobs,pid);//根据获取的 pid 查
      询前台任务
(484)         job->state=ST;//将前台任务状态设置为 ST,即停止
(485)         kill(-pid,SIGTSTP);//向前台进程组发送 SIGSTP 的信号
(486)         printf("Job [%d] (%d) stopped by signal %d\n",pid2jid(pid)
      ,pid,sig);//打印输出
(487)     }
(488)
(489)     sigprocmask(SIG_SETMASK,&pre,NULL);//解除阻塞
(490)     errno = olderrno;   //恢复原来的 errno
(491)     return;
(492) }
(493)
(494) /*********************
(495)  * End signal handlers
(496)  *********************/
(497)
(498) /***********************************************
(499)  * Helper routines that manipulate the job list
(500)  **********************************************/
(501)
(502) /* clearjob - Clear the entries in a job struct */
(503) void clearjob(struct job_t *job) {
(504)     job->pid = 0;
(505)     job->jid = 0;
(506)     job->state = UNDEF;
(507)     job->cmdline[0] = '\0';
(508) }
(509)
(510) /* initjobs - Initialize the job list */
(511) void initjobs(struct job_t *jobs) {
(512)     int i;
(513)
(514)     for (i = 0; i < MAXJOBS; i++)
(515)     clearjob(&jobs[i]);
(516) }
(517)
(518) /* maxjid - Returns largest allocated job ID */
(519) int maxjid(struct job_t *jobs)
(520) {
(521)     int i, max=0;
(522)
(523)     for (i = 0; i < MAXJOBS; i++)
```

```
(524)     if (jobs[i].jid > max)
(525)         max = jobs[i].jid;
(526)     return max;
(527) }
(528)
(529) /* addjob - Add a job to the job list */
(530) int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline
      )
(531) {
(532)     int i;
(533)
(534)     if (pid < 1)
(535)     return 0;
(536)
(537)     for (i = 0; i < MAXJOBS; i++) {
(538)     if (jobs[i].pid == 0) {
(539)         jobs[i].pid = pid;
(540)         jobs[i].state = state;
(541)         jobs[i].jid = nextjid++;
(542)         if (nextjid > MAXJOBS)
(543)         nextjid = 1;
(544)         strcpy(jobs[i].cmdline, cmdline);
(545)         if(verbose){
(546)             printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].
      pid, jobs[i].cmdline);
(547)             }
(548)             return 1;
(549)     }
(550)     }
(551)     printf("Tried to create too many jobs\n");
(552)     return 0;
(553) }
(554)
(555) /* deletejob - Delete a job whose PID=pid from the job list */
(556) int deletejob(struct job_t *jobs, pid_t pid)
(557) {
(558)     int i;
(559)
(560)     if (pid < 1)
(561)     return 0;
(562)
(563)     for (i = 0; i < MAXJOBS; i++) {
(564)     if (jobs[i].pid == pid) {
(565)         clearjob(&jobs[i]);
(566)         nextjid = maxjid(jobs)+1;
```

```
(567)          return 1;
(568)      }
(569)      }
(570)      return 0;
(571)  }
(572)
(573)  /* fgpid - Return PID of current foreground job, 0 if no such job
       */
(574)  pid_t fgpid(struct job_t *jobs) {
(575)      int i;
(576)
(577)      for (i = 0; i < MAXJOBS; i++)
(578)      if (jobs[i].state == FG)
(579)          return jobs[i].pid;
(580)      return 0;
(581)  }
(582)
(583)  /* getjobpid  - Find a job (by PID) on the job list */
(584)  struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
(585)      int i;
(586)
(587)      if (pid < 1)
(588)      return NULL;
(589)      for (i = 0; i < MAXJOBS; i++)
(590)      if (jobs[i].pid == pid)
(591)          return &jobs[i];
(592)      return NULL;
(593)  }
(594)
(595)  /* getjobjid  - Find a job (by JID) on the job list */
(596)  struct job_t *getjobjid(struct job_t *jobs, int jid)
(597)  {
(598)      int i;
(599)
(600)      if (jid < 1)
(601)      return NULL;
(602)      for (i = 0; i < MAXJOBS; i++)
(603)      if (jobs[i].jid == jid)
(604)          return &jobs[i];
(605)      return NULL;
(606)  }
(607)
(608)  /* pid2jid - Map process ID to job ID */
(609)  int pid2jid(pid_t pid)
(610)  {
```

```
(611)     int i;
(612)
(613)     if (pid < 1)
(614)     return 0;
(615)     for (i = 0; i < MAXJOBS; i++)
(616)     if (jobs[i].pid == pid) {
(617)             return jobs[i].jid;
(618)         }
(619)     return 0;
(620) }
(621)
(622) /* listjobs - Print the job list */
(623) void listjobs(struct job_t *jobs)
(624) {
(625)     int i;
(626)
(627)     for (i = 0; i < MAXJOBS; i++) {
(628)     if (jobs[i].pid != 0) {
(629)         printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
(630)         switch (jobs[i].state) {
(631)         case BG:
(632)             printf("Running ");
(633)             break;
(634)         case FG:
(635)             printf("Foreground ");
(636)             break;
(637)         case ST:
(638)             printf("Stopped ");
(639)             break;
(640)         default:
(641)             printf("listjobs: Internal error: job[%d].state=%d ",
(642)                i, jobs[i].state);
(643)         }
(644)         printf("%s", jobs[i].cmdline);
(645)     }
(646)     }
(647) }
(648) /*****************************
(649)  * end job list helper routines
(650)  *****************************/
(651)

(652) /***********************
(653)  * Other helper routines
(654)  ***********************/
```

```
(655)
(656) /*
(657)  * usage - print a help message
(658)  */
(659) void usage(void)
(660) {
(661)     printf("Usage: shell [-hvp]\n");
(662)     printf("   -h   print this message\n");
(663)     printf("   -v   print additional diagnostic information\n");
(664)     printf("   -p   do not emit a command prompt\n");
(665)     exit(1);
(666) }
(667)
(668) /*
(669)  * unix_error - unix-style error routine
(670)  */
(671) void unix_error(char *msg)
(672) {
(673)     fprintf(stdout, "%s: %s\n", msg, strerror(errno));
(674)     exit(1);
(675) }
(676)
(677) /*
(678)  * app_error - application-style error routine
(679)  */
(680) void app_error(char *msg)
(681) {
(682)     fprintf(stdout, "%s\n", msg);
(683)     exit(1);
(684) }
(685)
(686) /*
(687)  * Signal - wrapper for the sigaction function
(688)  */
(689) handler_t *Signal(int signum, handler_t *handler)
(690) {
(691)     struct sigaction action, old_action;
(692)
(693)     action.sa_handler = handler;
(694)     sigemptyset(&action.sa_mask); /* block sigs of type being hand
      led */
(695)     action.sa_flags = SA_RESTART; /* restart syscalls if possible
      */
(696)
(697)     if (sigaction(signum, &action, &old_action) < 0)
```

```
(698)      unix_error("Signal error");
(699)      return (old_action.sa_handler);
(700)  }
(701)
(702)  /*
(703)   * sigquit_handler - The driver program can gracefully terminate t
       he
(704)   *    child shell by sending it a SIGQUIT signal.
(705)   */
(706)  void sigquit_handler(int sig)
(707)  {
(708)      printf("Terminating after receipt of SIGQUIT signal\n");
(709)      exit(1);
(710)  }
(711)
```

第 4 章 TinyShell 测试

总分 15 分

4.1 测试方法

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

4.2 测试结果评价

tsh 与 tshref 的输出在以下两个方面可以不同：

（1）pid

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

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

4.3 自测试结果

填写以下各个测试用例的测试结果，每个测试用例 1 分。

4.3.1 测试用例 trace01.txt

<table>
<tr><td colspan="2">tsh 测试结果</td></tr>
<tr><td colspan="2">fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
trace01.txt - Properly terminate on EOF.
#</td></tr>
<tr><td colspan="2">tshref 测试结果</td></tr>
<tr><td colspan="2">fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
trace01.txt - Properly terminate on EOF.
#</td></tr>
<tr><td>测试结论</td><td>相同</td></tr>
</table>

4.3.2 测试用例 trace02.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

测试结论	相同

4.3.3 测试用例 trace03.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

测试结论	相同

4.3.4 测试用例 trace04.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (4201) ./myspin 1 &
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (4207) ./myspin 1 &
```

测试结论	相同

4.3.5 测试用例 trace05.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (4269) ./myspin 2 &
tsh> ./myspin 3 &
[2] (4271) ./myspin 3 &
tsh> jobs
[1] (4269) Running ./myspin 2 &
[2] (4271) Running ./myspin 3 &
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (4289) ./myspin 2 &
tsh> ./myspin 3 &
[2] (4291) ./myspin 3 &
tsh> jobs
[1] (4289) Running ./myspin 2 &
[2] (4291) Running ./myspin 3 &
```

测试结论	相同

4.3.6 测试用例 trace06.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (4301) terminated by signal 2
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (4307) terminated by signal 2
```

测试结论	相同

4.3.7 测试用例 trace07.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (4315) ./myspin 4 &
tsh> ./myspin 5
Job [2] (4317) terminated by signal 2
tsh> jobs
[1] (4315) Running ./myspin 4 &
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (4324) ./myspin 4 &
tsh> ./myspin 5
Job [2] (4326) terminated by signal 2
tsh> jobs
[1] (4324) Running ./myspin 4 &
```

测试结论	相同

4.3.8 测试用例 trace08.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (5106) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5108) stopped by signal 20
tsh> jobs
[1] (5106) Running ./myspin 4 &
[2] (5108) Stopped ./myspin 5
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ ma
ke rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (5127) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5129) stopped by signal 20
tsh> jobs
[1] (5127) Running ./myspin 4 &
[2] (5129) Stopped ./myspin 5
```

测试结论	相同

4.3.9 测试用例 trace09.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (5116) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5118) stopped by signal 20
tsh> jobs
[1] (5116) Running ./myspin 4 &
[2] (5118) Stopped ./myspin 5
tsh> bg %2
[2] (5118) ./myspin 5
tsh> jobs
[1] (5116) Running ./myspin 4 &
[2] (5118) Running ./myspin 5
```

tshref 测试结果：

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (5150) ./myspin 4 &
tsh> ./myspin 5
Job [2] (5152) stopped by signal 20
tsh> jobs
[1] (5150) Running ./myspin 4 &
[2] (5152) Stopped ./myspin 5
tsh> bg %2
[2] (5152) ./myspin 5
tsh> jobs
[1] (5150) Running ./myspin 4 &
[2] (5152) Running ./myspin 5
```

测试结论	相同

4.3.10 测试用例 trace10.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (5163) ./myspin 4 &
tsh> fg %1
Job [1] (5163) stopped by signal 20
tsh> jobs
[1] (5163) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (5173) ./myspin 4 &
tsh> fg %1
Job [1] (5173) stopped by signal 20
tsh> jobs
[1] (5173) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

测试结论	相同

4.3.11 测试用例 trace11.txt

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

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test11
./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (5479) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSI
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5474 pts/0    S+     0:00 make test11
   5475 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
   5476 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
   5477 pts/0    S+     0:00 ./tsh -p
   5482 pts/0    R      0:00 /bin/ps a
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/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] (5488) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSIO
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/X
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5483 pts/0    S+     0:00 make rtest11
   5484 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
   5485 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p
   5486 pts/0    S+     0:00 ./tshref -p
   5491 pts/0    R      0:00 /bin/ps a
```

测试结论	相同

4.3.12 测试用例 trace12.txt

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

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (5511) stopped by signal 20
tsh> jobs
[1] (5511) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSI
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5506 pts/0    S+     0:00 make test12
   5507 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
   5508 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
   5509 pts/0    S+     0:00 ./tsh -p
   5511 pts/0    T      0:00 ./mysplit 4
   5512 pts/0    T      0:00 ./mysplit 4
   5515 pts/0    R      0:00 /bin/ps a
```

tshref 测试结果

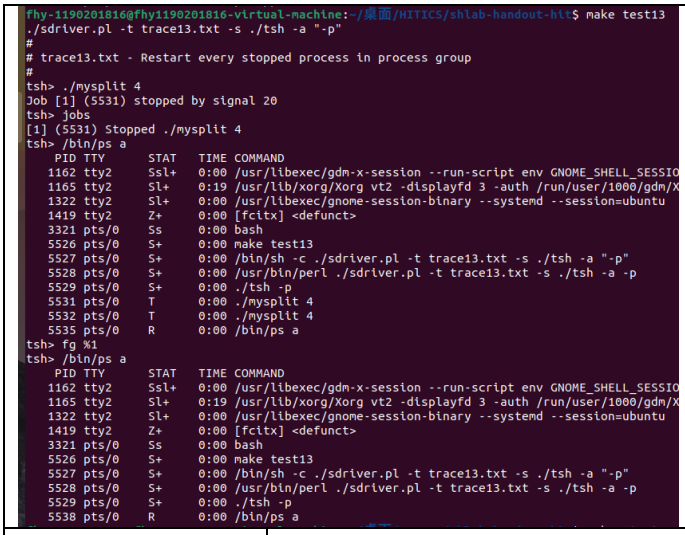
```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make rtest12
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (5521) stopped by signal 20
tsh> jobs
[1] (5521) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xa
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5516 pts/0    S+     0:00 make rtest12
   5517 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
   5518 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
   5519 pts/0    S+     0:00 ./tshref -p
   5521 pts/0    T      0:00 ./mysplit 4
   5522 pts/0    T      0:00 ./mysplit 4
   5525 pts/0    R      0:00 /bin/ps a
```

测试结论	相同

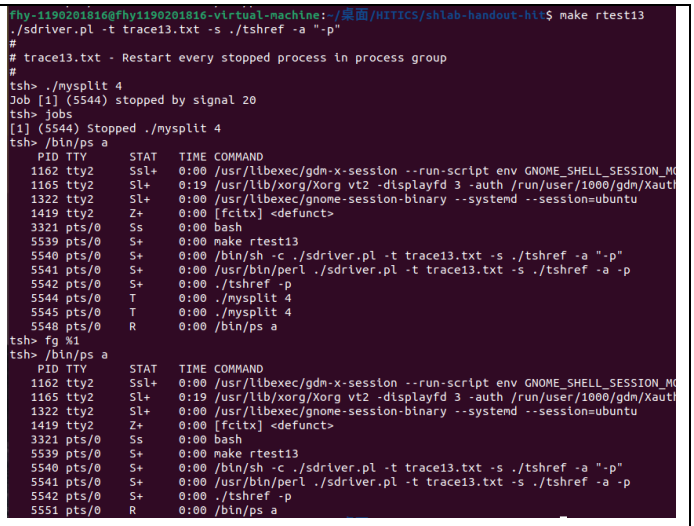
4.3.13 测试用例 trace13.txt

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

tsh 测试结果	tshref 测试结果



```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (5531) stopped by signal 20
tsh> jobs
[1] (5531) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSIO
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/X
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5526 pts/0    S+     0:00 make test13
   5527 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   5528 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   5529 pts/0    S+     0:00 ./tsh -p
   5531 pts/0    T      0:00 ./mysplit 4
   5532 pts/0    T      0:00 ./mysplit 4
   5535 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSIO
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/X
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5526 pts/0    S+     0:00 make test13
   5527 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
   5528 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
   5529 pts/0    S+     0:00 ./tsh -p
   5538 pts/0    R      0:00 /bin/ps a
```



```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/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] (5544) stopped by signal 20
tsh> jobs
[1] (5544) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MO
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauth
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5539 pts/0    S+     0:00 make rtest13
   5540 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
   5541 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
   5542 pts/0    S+     0:00 ./tshref -p
   5544 pts/0    T      0:00 ./mysplit 4
   5545 pts/0    T      0:00 ./mysplit 4
   5548 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1162 tty2     Ssl+   0:00 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MO
   1165 tty2     Sl+    0:19 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauth
   1322 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --session=ubuntu
   1419 tty2     Z+     0:00 [fcitx] <defunct>
   3321 pts/0    Ss     0:00 bash
   5539 pts/0    S+     0:00 make rtest13
   5540 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
   5541 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
   5542 pts/0    S+     0:00 ./tshref -p
   5551 pts/0    R      0:00 /bin/ps a
```

测试结论	相同

4.3.14 测试用例 trace14.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test14
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (5559) ./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] (5559) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (5559) ./myspin 4 &
tsh> jobs
[1] (5559) Running ./myspin 4 &
```

tshref 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make rtest14
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (5578) ./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] (5578) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (5578) ./myspin 4 &
tsh> jobs
[1] (5578) Running ./myspin 4 &
```

测试结论	相同

4.3.15 测试用例 trace15.txt

tsh 测试结果

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/shlab-handout-hit$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (5599) terminated by signal 2
tsh> ./myspin 3 &
[1] (5601) ./myspin 3 &
tsh> ./myspin 4 &
[2] (5603) ./myspin 4 &
tsh> jobs
[1] (5601) Running ./myspin 3 &
[2] (5603) Running ./myspin 4 &
tsh> fg %1
Job [1] (5601) stopped by signal 20
tsh> jobs
[1] (5601) Stopped ./myspin 3 &
[2] (5603) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (5601) ./myspin 3 &
tsh> jobs
[1] (5601) Running ./myspin 3 &
[2] (5603) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

tshref 测试结果：

```
fhy-1190201816@fhy1190201816-virtual-machine:~/桌面/HITICS/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] (5620) terminated by signal 2
tsh> ./myspin 3 &
[1] (5622) ./myspin 3 &
tsh> ./myspin 4 &
[2] (5624) ./myspin 4 &
tsh> jobs
[1] (5622) Running ./myspin 3 &
[2] (5624) Running ./myspin 4 &
tsh> fg %1
Job [1] (5622) stopped by signal 20
tsh> jobs
[1] (5622) Stopped ./myspin 3 &
[2] (5624) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (5622) ./myspin 3 &
tsh> jobs
[1] (5622) Running ./myspin 3 &
[2] (5624) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

测试结论:相同

第 5 章 评测得分

总分 20 分

实验程序统一测试的评分（教师评价）：

（1）正确性得分：_____（满分 10）

（2）性能加权得分：_____（满分 10）

第 6 章 总结

5.1 请总结本次实验的收获

理解了现代计算机系统进程与并发的基本知识

掌握了 linux 异常控制流和信号机制的基本原理和相关系统函数

掌握了 shell 的基本原理和实现方法

深入了理解 Linux 信号响应可能导致的并发冲突及解决方法

5.2 请给出对本次实验内容的建议

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.