哈爾濱工業大學

实验报告

实 验（七）

题　　目　TinyShell

　　微壳

专　　业　计算学部

学　　号　1190200708

班　　级　1903008

学　　生　熊峰

指 导 教 师　吴锐

实 验 地 点　G709

实 验 日 期　2021.05.31

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

X86-64 CPU；3.60GHz；16G RAM；256G SSD；1T SSD

1.2.2 软件环境

Win 10
Ubuntu 20.04.2 LTS
WSL2

1.2.3 开发工具

Visual Studio 2019；Vim；GCC；GDB；Code::Blocks; CLion 2020.3.1 x64;EDB

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

Kill 命令：

kill －l：列出信号

kill －SIGKILL 17130： 杀死 pid 为 17130 的进程

kill -9 17130 ：杀死 pid 为 17130 的进程，或者：

kill -9 -17130：杀死进程组 17130 中的每个进程

killall -9 pname： 杀死名字为 pname 的进程

进程状态：

D 不可中断睡眠 (通常是在 IO 操作) 收到信号不唤醒和不可运行, 进程必须等待直到有中断发生

R 正在运行或可运行（在运行队列排队中）

S 可中断睡眠 (休眠中, 受阻, 在等待某个条件的形成或接受到信号)

T 已停止的 进程收到 SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU 信号后停止运行

W 正在换页(2.6.内核之前有效)

X 死进程 (未开启)

Z 僵尸进程 a defunct (”zombie”) process

< 高优先级(not nice to other users)

N 低优先级(nice to other users)

L 页面锁定在内存（实时和定制的 IO）

s 一个信息头

l 多线程（使用 CLONE_THREAD，像 NPTL 的 pthreads 的那样）

+ 在前台进程组

ps t /ps aux /ps

t <终端机编号 n> 列终端 n 的程序的状况。

a 显示现行终端机下的所有程序，包括其他用户的程序。

u 以用户为主的格式来显示程序状况。

x 显示所有程序，不以终端来区分。

Linux>sleep 2000 |more|sort|grep hit &

Linux>ps -f a

Linux>ps aj

作业 ：jobs、 fg %n 、 bg%n

jobs 显示当前暂停的进程

bg %n 使第 n 个任务在后台运行(%前有空格)

fg %n 使第 n 个任务在前台运行

bg, fg 不带%n 表示对最后一个进程操作

ctrl+c: 终止前台作业(进程组的每个进程)

ctrl+z: 停止前台作业(进程组的每个进程)，随后可用 bg 恢复后台运行， fg 恢复前台运行。

第 2 章 实验预习

总分 20 分

2.1 进程的概念、创建和回收方法（5 分）

进程的概念：

进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。进程是一个可执行的、具有独立功能的程序关于某个数据集合的一次执行过程，也是操作系统进行资源分配和调度的基本单位。进程是操作系统动态执行的基本单元。

进程的创建：

父进程通过调用 fork 函数创建一个新的运行的子进程，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：子进程有不同于父进程的 PID。

进程在执行过程中可能创建多个新的进程。创建进程称为父进程，而新的进程称为子进程。每个新进程可以再创建其他进程，从而形成进程树。

进程的回收方法：

当进程终止时，它仍然消耗系统资源，父进程执行回收，成为僵死进程。父进程收到子进程的退出状态，内核删掉僵死子进程。

2.2 信号的机制、种类（5 分）

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。

信号的种类：

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop$ kill -l
 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      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

发送方法：

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。

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进程。

用系统调用 kill 发送信号。

阻塞方法：

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。

处理程序：

处理程序尽可能简单：简单设置全局标志并立即返回。

在处理程序中只调用异步信号安全的函数：printf, sprintf, malloc, and exit are not safe!

保存和恢复 errno：确保其他处理程序不会覆盖当前的 errno。

阻塞所有信号保护对共享全局数据结构的访问，避免可能的冲突。

用 volatile 声明全局变量：强迫编译器从内存中读取引用的值。

用 sig_atomic_t 声明标志：原子型标志: 只适用于单个的读或者写，不适用 flag++或 flag=flag+10 这样的更新(e.g. flag = 1, not flag++)，采用这种方式声明的标

志不需要类似其他全局变量的保护。

2.4 什么是 shell，功能和处理流程（5 分）

shell 是一个交互型应用级程序，代表用户运行其他程序。

功能：shell 执行一些列的读/求值步骤，读写步骤读取用户的命令行，求值的步骤解析命令，代表用户运行。

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。然后 shell 在搜索路径里寻找这些应用程序。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：
解析并执行命令。当用户的命令是 tsh 内置的 4 个命令：quit，jobs，fg，bg 时，将立即执行这些功能。否则，将会 fork 一个子进程，并在子进程中执行命令。若进程在前台执行，那么就等待改作业运行终止后返回。

参　　数：
Char *cmdline 即用户在 shell 中输入的命令的字符串的指针。

处理流程：
1. 使用 parseline 解析用户输入的命令。若 argv[0]为空则直接返回。
2. 使用 builtin_cmd()函数判断命令是否为内置命令，若不是，则继续执行。
3. 在函数内部加上阻塞列表，将 SIGCHLD,SIGINT，SIGTSTP 信号加入阻塞列表。
4. 创建子进程，阻塞 SIGCHLD 信号，并在子进程调用 sigprocmask()函数解除 SIGCHLD 信号，修改进程组 pgid，运行 execve，并通过 exit 退出。
5. 在父进程中，使用 addjob 函数添加 job，然后接触阻塞信号，以便信号处理程序可以再次运行。
6. 判断该进程是前台进程还是后台进程，若是前台进程，则调用 waitfg 函数，阻塞进程，直到进程 pid 不再是前台进程。
7. 否则输出当前进程的信息。

要点分析：
1. 每个子进程都需要一个新的进程组 pgid，防止在键盘上输入 Crtl+C(Crtl+Z)时，我们的后台子进程从内核接收 SIGINT 等信号。
2. 在 fork 子程序前，要通过阻塞列表阻塞 SIGCHLD、SIGINT、SIGTSTP 信号，防止添加任务时信号出现竞争。

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

函数功能：检查用户输入的命令是否为 tsh 的内置命令，若是则执行命令。

参　　数：char **argv，即用户输入的命令。

处理流程：
通过 strcmp 命令比较 argv[0]与内置命令，若是 quit 则直接推出，若是 jobs 则打印 jobs，并返回 1，若是 bg 或 fg 则调用 do_bgfg 函数，并返回 1，若是&则直接返回 1.

要点分析：
用户若直接回车，则可能导致 argv 变量为空指针。

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

函数功能：执行内置的 bg 或 fg 命令

参　　数：char **argv，即用户输入的命令行，并通过空格分隔。

处理流程：
1. 判断 fg 或 bg 后是否有参数，若没有参数，则打印提示并返回
2. 若 fg 或 bg 后是数字，获取进程号，并使用 getjobjid 得到 job
3. 若 fg 或 bg 后跟的是%加上数字，则可以通过 jid 和 getjobjid 获取 jobp
4. 判断命令为 bg 还是 fg，若是 bg 则发送 SIGCONT 信号给进程组 PID 的每个进程，设置任务的状态为 BG，打印任务的 jid，pid 和命令行。
5. 若是 fg 则发送 SIGCONT 给进程组的每个进程，并将状态设置为 FG，然后调用 waitfg 函数等待前台进程的结束。

要点分析：
1. 此时的 SIGCONT 信号应该对 job 所在的进程组发送。
2. 需要判断命令后的参数是数字还是%

3.1.4 void waitfg(pid_t pid) 函数（5 分）

函数功能：阻塞直到进程 pid 不再是前台进程。

参　　数：pid 即需要等待的进程 pid

处理流程：使用 sigprocmask 阻塞信号，再调用 fgpid 获取前台进程 pid，若前台进程的 pid 与参数相同，则等待并且循环。

要点分析：在 while 循环中，若使用 pause 函数，则程序必须等待较长时间。

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

函数功能：内核因某个子进程终止，发送 SIGCHLD 信号，handler 函数将所有僵死的子进程回收，但不会等待其他正在运行的子程序终止。

参　　数：形参是 sig

处理流程：

1. 首先将将 oldererrno 赋值为 olderrno，并将信号添加到 mask 阻塞列表中
2. 采用非挂起的方式调用 waitpid 函数，while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)，尽可能的回收子进程。
3. 在循环中阻塞信号，并使用 getjobpid()函数，通过 pid 找到 pid 的 job。
4. 通过 waitpid 存放的 status 的信息，判断子进程的退出状态。若使返回的子进程当前是停止的，则 WIFSTOPPED(status)为真，将由 pid 找的 job 的状态改为 ST，并将 job 的 jid 和 pid 以及导致子进程停止的信号的编号输出。
5. 若子进程因一个未被捕获的信号终止，则 WIFSIGNALED(status)返回真，将由 pid 找的 job、pid 及导致子进程终止的信息编号输出，再通过 deletejob()信号将其回收。
6. 清空缓冲区，解除阻塞，并恢复 errno 返回。

要点分析：
在 while 循环中使用 waitpid()函数可以尽可能多的回收僵尸进程。
调用 deletejob()函数时，由于 jobs 是全局变量，因此需要阻塞信号阻止发生修改。

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

```c
/*
 * 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>
```

```
13. #include <sys/wait.h>
14. #include <errno.h>
15.
16. /* Misc manifest constants */
17. #define MAXLINE    1024   /* max line size */
18. #define MAXARGS     128   /* max args on a command line */
19. #define MAXJOBS      16   /* max jobs at any point in time */
20. #define MAXJID    1<<16   /* max job ID */
21.
22. /* Job states */
23. #define UNDEF 0 /* undefined */
24. #define FG 1    /* running in foreground */
25. #define BG 2    /* running in background */
26. #define ST 3    /* stopped */
27.
28. /*
29.  * Jobs states: FG (foreground), BG (background), ST (stopped)
30.  * Job state transitions and enabling actions:
31.  *     FG -> ST  : ctrl-z
32.  *     ST -> FG  : fg command
33.  *     ST -> BG  : bg command
34.  *     BG -> FG  : fg command
35.  * At most 1 job can be in the FG state.
36.  */
37.
38. /* Global variables */
39. extern char **environ;      /* defined in libc */
40. char prompt[] = "tsh> ";    /* command line prompt (DO NOT CHANGE) */
41. int verbose = 0;            /* if true, print additional output */
42. int nextjid = 1;            /* next job ID to allocate */
43. char sbuf[MAXLINE];         /* for composing sprintf messages */
44.
45. struct job_t {              /* The job struct */
46.     pid_t pid;              /* job PID */
47.     int jid;                /* job ID [1, 2, ...] */
48.     int state;              /* UNDEF, BG, FG, or ST */
49.     char cmdline[MAXLINE];  /* command line */
50. };
51. struct job_t jobs[MAXJOBS]; /* The job list */
52. /* End global variables */
53.
54.
55. /* Function prototypes */
56.
57. /* Here are the functions that you will implement */
58. void eval(char *cmdline);
59. int builtin_cmd(char **argv);
60. void do_bgfg(char **argv);
61. void waitfg(pid_t pid);
62.
63. void sigchld_handler(int sig);
64. void sigtstp_handler(int sig);
65. void sigint_handler(int sig);
66.
67. /* Here are helper routines that we've provided for you */
68. int parseline(const char *cmdline, char **argv);
69. void sigquit_handler(int sig);
70.
71. void clearjob(struct job_t *job);
72. void initjobs(struct job_t *jobs);
73. int maxjid(struct job_t *jobs);
```

```
74. int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);
75. int deletejob(struct job_t *jobs, pid_t pid);
76. pid_t fgpid(struct job_t *jobs);
77. struct job_t *getjobpid(struct job_t *jobs, pid_t pid);
78. struct job_t *getjobjid(struct job_t *jobs, int jid);
79. int pid2jid(pid_t pid);
80. void listjobs(struct job_t *jobs);
81.
82. void usage(void);
83. void unix_error(char *msg);
84. void app_error(char *msg);
85. typedef void handler_t(int);
86. handler_t *Signal(int signum, handler_t *handler);
87.
88. /*
89.  * main - The shell's main routine
90.  */
91. int main(int argc, char **argv)
92. {
93.     char c;
94.     char cmdline[MAXLINE];
95.     int emit_prompt = 1; /* emit prompt (default) */
96.
97.     /* Redirect stderr to stdout (so that driver will get all output
98.      * on the pipe connected to stdout) */
99.     dup2(1, 2);
100.
101.     /* Parse the command line */
102.     while ((c = getopt(argc, argv, "hvp")) != EOF) {
103.         switch (c) {
104.         case 'h':             /* print help message */
105.             usage();
106.         break;
107.         case 'v':             /* emit additional diagnostic info */
108.             verbose = 1;
109.         break;
110.         case 'p':             /* don't print a prompt */
111.             emit_prompt = 0;  /* handy for automatic testing */
112.         break;
113.     default:
114.             usage();
115.     }
116.     }
117.
118.     /* Install the signal handlers */
119.
120.     /* These are the ones you will need to implement */
121.     Signal(SIGINT,  sigint_handler);   /* ctrl-c */
122.     Signal(SIGTSTP, sigtstp_handler);  /* ctrl-z */
123.     Signal(SIGCHLD, sigchld_handler);  /* Terminated or stopped child */
124.
125.     /* This one provides a clean way to kill the shell */
126.     Signal(SIGQUIT, sigquit_handler);
127.
128.     /* Initialize the job list */
129.     initjobs(jobs);
130.
131.     /* Execute the shell's read/eval loop */
132.     while (1) {
133.
134.     /* Read command line */
```

```
135.     if (emit_prompt) {
136.         printf("%s", prompt);
137.         fflush(stdout);
138.     }
139.     if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
140.         app_error("fgets error");
141.     if (feof(stdin)) { /* End of file (ctrl-d) */
142.         fflush(stdout);
143.         exit(0);
144.     }
145.
146.     /* Evaluate the command line */
147.     eval(cmdline);
148.     fflush(stdout);
149.     fflush(stdout);
150.     }
151.
152.     exit(0); /* control never reaches here */
153. }
154.
155. /*
156.  * eval - Evaluate the command line that the user has just typed in
157.  *
158.  * If the user has requested a built-in command (quit, jobs, bg or fg)
159.  * then execute it immediately. Otherwise, fork a child process and
160.  * run the job in the context of the child. If the job is running in
161.  * the foreground, wait for it to terminate and then return.  Note:
162.  * each child process must have a unique process group ID so that our
163.  * background children don't receive SIGINT (SIGTSTP) from the kernel
164.  * when we type ctrl-c (ctrl-z) at the keyboard.
165. */
166. void eval(char *cmdline)
167. {
168.     /* $begin handout */
169.     char *argv[MAXARGS]; /* argv for execve() */
170.     int bg;              /* should the job run in bg or fg? */
171.     pid_t pid;           /* process id */
172.     sigset_t mask;       /* signal mask */
173.
174.     /* Parse command line */
175.     bg = parseline(cmdline, argv);
176.     if (argv[0] == NULL)
177.     return;   /* ignore empty lines */
178.
179.     if (!builtin_cmd(argv)) {
180.
181.         /*
182.      * This is a little tricky. Block SIGCHLD, SIGINT, and SIGTSTP
183.      * signals until we can add the job to the job list. This
184.      * eliminates some nasty races between adding a job to the job
185.      * list and the arrival of SIGCHLD, SIGINT, and SIGTSTP signals.
186.      */
187.
188.     if (sigemptyset(&mask) < 0)
189.         unix_error("sigemptyset error");
190.     if (sigaddset(&mask, SIGCHLD))
191.         unix_error("sigaddset error");
192.     if (sigaddset(&mask, SIGINT))
193.         unix_error("sigaddset error");
194.     if (sigaddset(&mask, SIGTSTP))
195.         unix_error("sigaddset error");
```

```
196.     if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0)
197.         unix_error("sigprocmask error");
198.
199.     /* Create a child process */
200.     if ((pid = fork()) < 0)
201.         unix_error("fork error");
202.
203.     /*
204.      * Child  process
205.      */
206.
207.     if (pid == 0) {
208.         /* Child unblocks signals */
209.         sigprocmask(SIG_UNBLOCK, &mask, NULL);
210.
211.         /* Each new job must get a new process group ID
212.            so that the kernel doesn't send ctrl-c and ctrl-z
213.            signals to all of the shell's jobs */
214.         if (setpgid(0, 0) < 0)
215.         unix_error("setpgid error");
216.
217.         /* Now load and run the program in the new job */
218.         if (execve(argv[0], argv, environ) < 0) {
219.         printf("%s: Command not found\n", argv[0]);
220.         exit(0);
221.         }
222.     }
223.
224.     /*
225.      * Parent process
226.      */
227.
228.     /* Parent adds the job, and then unblocks signals so that
229.        the signals handlers can run again */
230.     addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
231.     sigprocmask(SIG_UNBLOCK, &mask, NULL);
232.
233.     if (!bg)
234.         waitfg(pid);
235.     else
236.         printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
237.     }
238.     /* $end handout */
239.     return;
240. }
241.
242. /*
243.  * parseline - Parse the command line and build the argv array.
244.  *
245.  * Characters enclosed in single quotes are treated as a single
246.  * argument.  Return true if the user has requested a BG job, false if
247.  * the user has requested a FG job.
248.  */
249. int parseline(const char *cmdline, char **argv)
250. {
251.     static char array[MAXLINE]; /* holds local copy of command line */
252.     char *buf = array;          /* ptr that traverses command line */
253.     char *delim;                /* points to first space delimiter */
254.     int argc;                   /* number of args */
255.     int bg;                     /* background job? */
256.
```

```
257.     strcpy(buf, cmdline);
258.     buf[strlen(buf)-1] = ' ';  /* replace trailing '\n' with space */
259.     while (*buf && (*buf == ' ')) /* ignore leading spaces */
260.     buf++;
261.
262.     /* Build the argv list */
263.     argc = 0;
264.     if (*buf == '\'') {
265.     buf++;
266.     delim = strchr(buf, '\'');
267.     }
268.     else {
269.     delim = strchr(buf, ' ');
270.     }
271.
272.     while (delim) {
273.     argv[argc++] = buf;
274.     *delim = '\0';
275.     buf = delim + 1;
276.     while (*buf && (*buf == ' ')) /* ignore spaces */
277.            buf++;
278.
279.     if (*buf == '\'') {
280.         buf++;
281.         delim = strchr(buf, '\'');
282.     }
283.     else {
284.         delim = strchr(buf, ' ');
285.     }
286.     }
287.     argv[argc] = NULL;
288.
289.     if (argc == 0)  /* ignore blank line */
290.     return 1;
291.
292.     /* should the job run in the background? */
293.     if ((bg = (*argv[argc-1] == '&')) != 0) {
294.     argv[--argc] = NULL;
295.     }
296.     return bg;
297. }
298.
299. /*
300.  * builtin_cmd - If the user has typed a built-in command then execute
301.  *    it immediately.
302.  */
303. int builtin_cmd(char **argv)
304. {
305.     if (!strcmp(argv[0], "quit"))
306.     {
307.         // 退出
308.         exit(0);
309.     }
310.     else if (!strcmp(argv[0], "jobs"))
311.     {
312.         // 调用 listjobs 打印 jobs
313.         listjobs(jobs);
314.         return 1;
315.     }
316.     else if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg"))
317.     {
```

```
318.         // 执行内置的 bg 和 fg 命令
319.         do_bgfg(argv);
320.         return 1;
321.     }
322.     if(!strcmp(argv[0], "&")) {
323.         return 1;
324.     }
325.     return 0;     /* not a builtin command */
326. }
327.
328. /*
329.  * do_bgfg - Execute the builtin bg and fg commands
330.  */
331. void do_bgfg(char **argv)
332. {
333.     /* $begin handout */
334.     struct job_t *jobp=NULL;
335.
336.     /* Ignore command if no argument */
337.     if (argv[1] == NULL) {
338.     printf("%s command requires PID or %%jobid argument\n", argv[0]);
339.     return;
340.     }
341.
342.     /* Parse the required PID or %JID arg */
343.     if (isdigit(argv[1][0])) {
344.     pid_t pid = atoi(argv[1]);
345.     if (!(jobp = getjobpid(jobs, pid))) {
346.         printf("(%d): No such process\n", pid);
347.         return;
348.     }
349.     }
350.     else if (argv[1][0] == '%') {
351.     int jid = atoi(&argv[1][1]);
352.     if (!(jobp = getjobjid(jobs, jid))) {
353.         printf("%s: No such job\n", argv[1]);
354.         return;
355.     }
356.     }
357.     else {
358.     printf("%s: argument must be a PID or %%jobid\n", argv[0]);
359.     return;
360.     }
361.
362.     /* bg command */
363.     if (!strcmp(argv[0], "bg")) {
364.     if (kill(-(jobp->pid), SIGCONT) < 0)
365.         unix_error("kill (bg) error");
366.     jobp->state = BG;
367.     printf("[%d] (%d) %s", jobp->jid, jobp->pid, jobp->cmdline);
368.     }
369.
370.     /* fg command */
371.     else if (!strcmp(argv[0], "fg")) {
372.     if (kill(-(jobp->pid), SIGCONT) < 0)
373.         unix_error("kill (fg) error");
374.     jobp->state = FG;
375.     waitfg(jobp->pid);
376.     }
377.     else {
378.     printf("do_bgfg: Internal error\n");
```

```
379.     exit(0);
380.     }
381.     /* $end handout */
382.     return;
383. }
384.
385. /*
386.  * waitfg - Block until process pid is no longer the foreground process
387.  */
388. void waitfg(pid_t pid)
389. {
390.     while(pid == fgpid(jobs)){
391.         sleep(1);
392.     }
393.     return;
394. }
395.
396. /*****************
397.  * Signal handlers
398.  *****************/
399.
400. /*
401.  * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
402.  *     a child job terminates (becomes a zombie), or stops because it
403.  *     received a SIGSTOP or SIGTSTP signal. The handler reaps all
404.  *     available zombie children, but doesn't wait for any other
405.  *     currently running children to terminate.
406.  */
407. void sigchld_handler(int sig)
408. {
409.     // 保存 errno
410.     int olderrno = errno;
411.     pid_t pid;
412.     int status;
413.     sigset_t mask_all, prev;
414.     sigfillset(&mask_all);  // 设置全阻塞
415.     while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0)
416.     {
417.         // WNOHANG | WUNTRACED 是立即返回
418.         // 用 WIFEXITED(status)，WIFSIGNALED(status)，WIFSTOPPED(status)捕获终止
419.         // 被停止的子进程的退出状态。
420.         if (WIFEXITED(status))
421.         {
422.             sigprocmask(SIG_BLOCK, &mask_all, &prev);
423.             deletejob(jobs, pid);
424.             sigprocmask(SIG_SETMASK, &prev, NULL);
425.         }
426.         // 子进程终止引起的返回,判断是否是前台进程
427.         // 并且判断该信号是否是未捕获的信号
428.         else if (WIFSIGNALED(status))
429.         {
430.             struct job_t* job = getjobpid(jobs, pid);
431.             sigprocmask(SIG_BLOCK, &mask_all, &prev);
432.             printf("Job [%d] (%d) terminated by signal %d\n", job->jid, job->pid, WTERMSIG
433.             deletejob(jobs, pid);
434.             sigprocmask(SIG_SETMASK, &prev, NULL);
435.         }
436.         else
437.         {
438.             struct job_t* job = getjobpid(jobs, pid);
439.             sigprocmask(SIG_BLOCK, &mask_all, &prev);
```

```
440.             printf("Job [%d] (%d) stopped by signal %d\n", job->jid, job->pid, WSTOPSIG(st
441.             job->state= ST;
442.             sigprocmask(SIG_SETMASK, &prev, NULL);
443.         }
444.     }
445.     // 恢复 olderrno
446.     errno = olderrno;
447.     return;
448. }
449.
450. /*
451.  * sigint_handler - The kernel sends a SIGINT to the shell whenver the
452.  *    user types ctrl-c at the keyboard.  Catch it and send it along
453.  *    to the foreground job.
454.  */
455. void sigint_handler(int sig)
456. {
457.     int olderrno = errno;
458.     sigset_t mask_all, mask_prev;
459.     pid_t curr_fg_pid;
460.     sigfillset(&mask_all);
461.     // 访问全局结构体数组，阻塞信号
462.     sigprocmask(SIG_BLOCK, &mask_all, &mask_prev);
463.     curr_fg_pid = fgpid(jobs);
464.     sigprocmask(SIG_SETMASK, &mask_prev, NULL);
465.     if(curr_fg_pid != 0){
466.         kill(-curr_fg_pid, SIGINT);
467.     }
468.     errno = olderrno;
469.     return;
470. }
471.
472. /*
473.  * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
474.  *     the user types ctrl-z at the keyboard. Catch it and suspend the
475.  *     foreground job by sending it a SIGTSTP.
476.  */
477. void sigtstp_handler(int sig)
478. {
479.     int olderrno = errno;
480.     pid_t pid = fgpid(jobs);
481.     if (pid != 0)
482.         kill(-pid, sig);
483.     errno = olderrno;
484.     return;
485. }
486.
487. /*********************
488.  * End signal handlers
489.  *********************/
490.
491. /***********************************************
492.  * Helper routines that manipulate the job list
493.  **********************************************/
494.
495. /* clearjob - Clear the entries in a job struct */
496. void clearjob(struct job_t *job) {
497.     job->pid = 0;
498.     job->jid = 0;
499.     job->state = UNDEF;
500.     job->cmdline[0] = '\0';
```

```
501. }
502.
503. /* initjobs - Initialize the job list */
504. void initjobs(struct job_t *jobs) {
505.     int i;
506.
507.     for (i = 0; i < MAXJOBS; i++)
508.     clearjob(&jobs[i]);
509. }
510.
511. /* maxjid - Returns largest allocated job ID */
512. int maxjid(struct job_t *jobs)
513. {
514.     int i, max=0;
515.
516.     for (i = 0; i < MAXJOBS; i++)
517.     if (jobs[i].jid > max)
518.         max = jobs[i].jid;
519.     return max;
520. }
521.
522. /* addjob - Add a job to the job list */
523. int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline)
524. {
525.     int i;
526.
527.     if (pid < 1)
528.     return 0;
529.
530.     for (i = 0; i < MAXJOBS; i++) {
531.     if (jobs[i].pid == 0) {
532.         jobs[i].pid = pid;
533.         jobs[i].state = state;
534.         jobs[i].jid = nextjid++;
535.         if (nextjid > MAXJOBS)
536.         nextjid = 1;
537.         strcpy(jobs[i].cmdline, cmdline);
538.         if(verbose){
539.             printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].pid, jobs[i].cmdline);
540.             }
541.             return 1;
542.     }
543.     }
544.     printf("Tried to create too many jobs\n");
545.     return 0;
546. }
547.
548. /* deletejob - Delete a job whose PID=pid from the job list */
549. int deletejob(struct job_t *jobs, pid_t pid)
550. {
551.     int i;
552.
553.     if (pid < 1)
554.     return 0;
555.
556.     for (i = 0; i < MAXJOBS; i++) {
557.     if (jobs[i].pid == pid) {
558.         clearjob(&jobs[i]);
559.         nextjid = maxjid(jobs)+1;
560.         return 1;
561.     }
```

```
562.     }
563.     return 0;
564. }
565.
566. /* fgpid - Return PID of current foreground job, 0 if no such job */
567. pid_t fgpid(struct job_t *jobs) {
568.     int i;
569.
570.     for (i = 0; i < MAXJOBS; i++)
571.     if (jobs[i].state == FG)
572.         return jobs[i].pid;
573.     return 0;
574. }
575.
576. /* getjobpid  - Find a job (by PID) on the job list */
577. struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
578.     int i;
579.
580.     if (pid < 1)
581.     return NULL;
582.     for (i = 0; i < MAXJOBS; i++)
583.     if (jobs[i].pid == pid)
584.         return &jobs[i];
585.     return NULL;
586. }
587.
588. /* getjobjid  - Find a job (by JID) on the job list */
589. struct job_t *getjobjid(struct job_t *jobs, int jid)
590. {
591.     int i;
592.
593.     if (jid < 1)
594.     return NULL;
595.     for (i = 0; i < MAXJOBS; i++)
596.     if (jobs[i].jid == jid)
597.         return &jobs[i];
598.     return NULL;
599. }
600.
601. /* pid2jid - Map process ID to job ID */
602. int pid2jid(pid_t pid)
603. {
604.     int i;
605.
606.     if (pid < 1)
607.     return 0;
608.     for (i = 0; i < MAXJOBS; i++)
609.     if (jobs[i].pid == pid) {
610.             return jobs[i].jid;
611.         }
612.     return 0;
613. }
614.
615. /* listjobs - Print the job list */
616. void listjobs(struct job_t *jobs)
617. {
618.     int i;
619.
620.     for (i = 0; i < MAXJOBS; i++) {
621.     if (jobs[i].pid != 0) {
622.         printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
```

```
623.         switch (jobs[i].state) {
624.         case BG:
625.             printf("Running ");
626.             break;
627.         case FG:
628.             printf("Foreground ");
629.             break;
630.         case ST:
631.             printf("Stopped ");
632.             break;
633.         default:
634.             printf("listjobs: Internal error: job[%d].state=%d ",
635.                i, jobs[i].state);
636.         }
637.         printf("%s", jobs[i].cmdline);
638.     }
639.     }
640. }
641. /***************************
642.  * end job list helper routines
643.  ***************************/
644.
645.
646. /***********************
647.  * Other helper routines
648.  ***********************/
649.
650. /*
651.  * usage - print a help message
652.  */
653. void usage(void)
654. {
655.     printf("Usage: shell [-hvp]\n");
656.     printf("   -h   print this message\n");
657.     printf("   -v   print additional diagnostic information\n");
658.     printf("   -p   do not emit a command prompt\n");
659.     exit(1);
660. }
661.
662. /*
663.  * unix_error - unix-style error routine
664.  */
665. void unix_error(char *msg)
666. {
667.     fprintf(stdout, "%s: %s\n", msg, strerror(errno));
668.     exit(1);
669. }
670.
671. /*
672.  * app_error - application-style error routine
673.  */
674. void app_error(char *msg)
675. {
676.     fprintf(stdout, "%s\n", msg);
677.     exit(1);
678. }
679.
680. /*
681.  * Signal - wrapper for the sigaction function
682.  */
683. handler_t *Signal(int signum, handler_t *handler)
```

```
684. {
685.     struct sigaction action, old_action;
686.
687.     action.sa_handler = handler;
688.     sigemptyset(&action.sa_mask); /* block sigs of type being handled */
689.     action.sa_flags = SA_RESTART; /* restart syscalls if possible */
690.
691.     if (sigaction(signum, &action, &old_action) < 0)
692.     unix_error("Signal error");
693.     return (old_action.sa_handler);
694. }
695.
696. /*
697.  * sigquit_handler - The driver program can gracefully terminate the
698.  *    child shell by sending it a SIGQUIT signal.
699.  */
700. void sigquit_handler(int sig)
701. {
702.     printf("Terminating after receipt of SIGQUIT signal\n");
703.     exit(1);
704. }
```

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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab7/shlab-handout-hit$ make test01 ./sdriver.pl -t trace01.txt -s ./tsh -a "-p" # # trace01.txt - Properly terminate on EOF. #	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab7/shlab-handout-hit$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. #	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab7/shlab-handout-hit$ make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (96) ./myspin 1 &	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (102) ./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

baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (108) ./myspin 2 & tsh> ./myspin 3 & [2] (110) ./myspin 3 & tsh> jobs [1] (108) Running ./myspin 2 & [2] (110) Running ./myspin 3 &	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (117) ./myspin 2 & tsh> ./myspin 3 & [2] (119) ./myspin 3 & tsh> jobs [1] (117) Running ./myspin 2 & [2] (119) 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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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 [1] (126) terminated by signal 2	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (132) 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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (138) ./myspin 4 & tsh> ./myspin 5 Job [2] (140) terminated by signal 2 tsh> jobs [1] (138) Running ./myspin 4 &	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (147) ./myspin 4 & tsh> ./myspin 5 Job [2] (149) terminated by signal 2 tsh> jobs [1] (147) 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

baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (156) ./myspin 4 & tsh> ./myspin 5 Job [2] (158) stopped by signal 20 tsh> jobs [1] (156) Running ./myspin 4 & [2] (158) Stopped ./myspin 5	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (165) ./myspin 4 & tsh> ./myspin 5 Job [2] (167) stopped by signal 20 tsh> jobs [1] (165) Running ./myspin 4 & [2] (167) 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

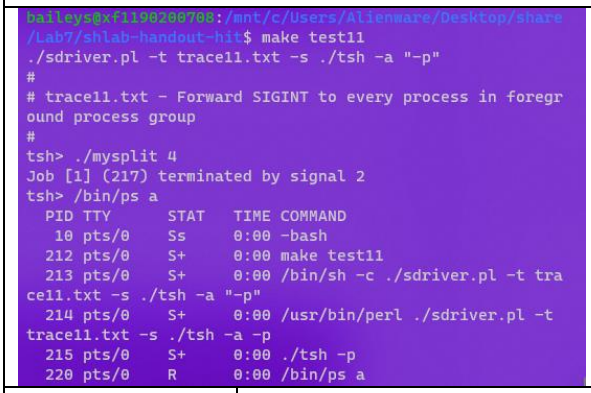
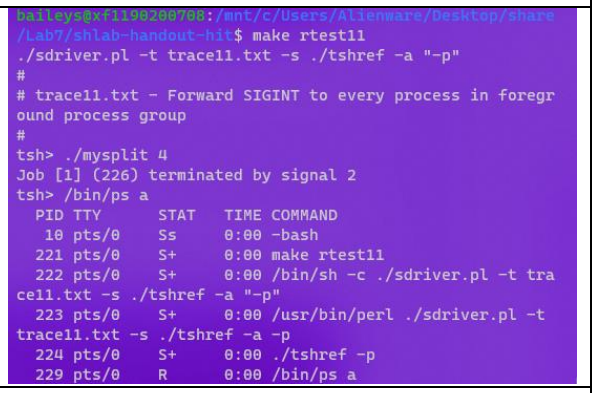
tsh 测试结果	tshref 测试结果
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/Lab7/shlab-handout-hit$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (174) ./myspin 4 & tsh> ./myspin 5 Job [2] (176) stopped by signal 20 tsh> jobs [1] (174) Running ./myspin 4 & [2] (176) Stopped ./myspin 5 tsh> bg %2 [2] (176) ./myspin 5 tsh> jobs [1] (174) Running ./myspin 4 & [2] (176) Running ./myspin 5	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (185) ./myspin 4 & tsh> ./myspin 5 Job [2] (187) stopped by signal 20 tsh> jobs [1] (185) Running ./myspin 4 & [2] (187) Stopped ./myspin 5 tsh> bg %2 [2] (187) ./myspin 5 tsh> jobs [1] (185) Running ./myspin 4 & [2] (187) 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
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (197) ./myspin 4 & tsh> fg %1 Job [1] (197) stopped by signal 20 tsh> jobs [1] (197) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (207) ./myspin 4 & tsh> fg %1 Job [1] (207) stopped by signal 20 tsh> jobs [1] (207) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs
测试结论	相同/不同，原因分析如下：相同

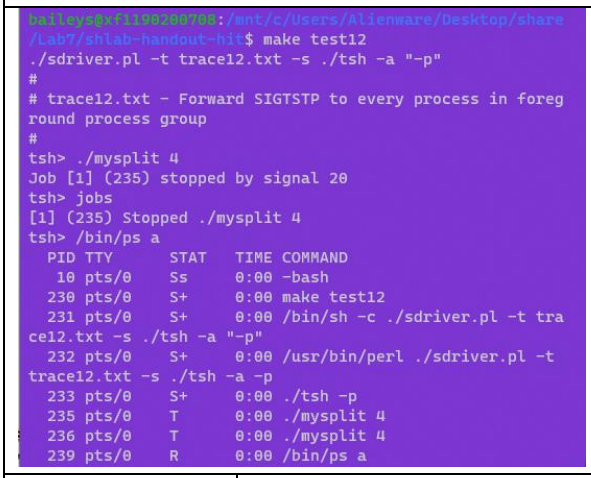
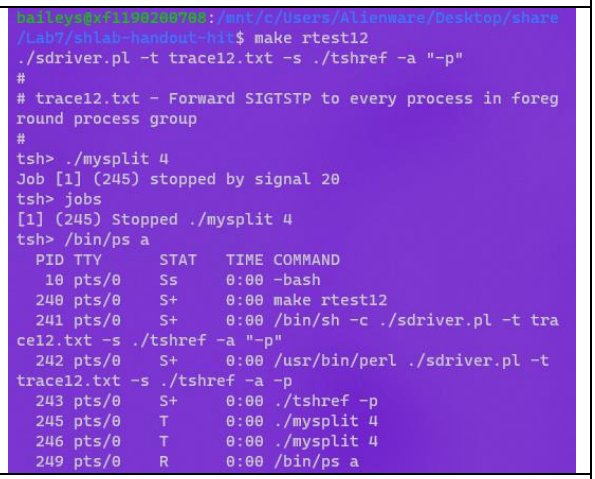
4.3.11 测试用例 trace11.txt

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
	
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

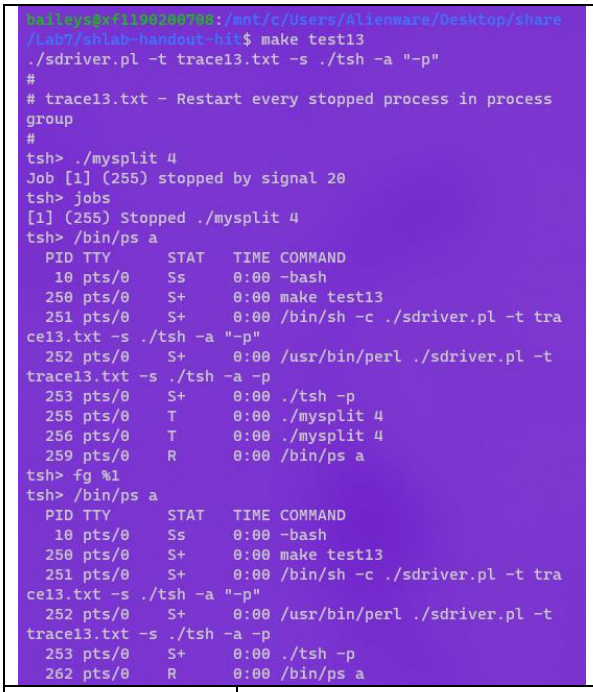
测试中 ps 指令的输出内容较多，仅记录和本实验密切相关的 tsh、mysplit 等进程的部分信息即可。

tsh 测试结果	tshref 测试结果
	
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



```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share
/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
Job [1] (255) stopped by signal 20
tsh> jobs
[1] (255) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
   10 pts/0    Ss     0:00 -bash
  250 pts/0    S+     0:00 make test13
  251 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t tra
ce13.txt -s ./tsh -a "-p"
  252 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t
trace13.txt -s ./tsh -a -p
  253 pts/0    S+     0:00 ./tsh -p
  255 pts/0    T      0:00 ./mysplit 4
  256 pts/0    T      0:00 ./mysplit 4
  259 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
   10 pts/0    Ss     0:00 -bash
  250 pts/0    S+     0:00 make test13
  251 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t tra
ce13.txt -s ./tsh -a "-p"
  252 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t
trace13.txt -s ./tsh -a -p
  253 pts/0    S+     0:00 ./tsh -p
  262 pts/0    R      0:00 /bin/ps a
```

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share
/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] (268) stopped by signal 20
tsh> jobs
[1] (268) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
   10 pts/0    Ss     0:00 -bash
  263 pts/0    S+     0:00 make rtest13
  264 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t tra
ce13.txt -s ./tshref -a "-p"
  265 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t
trace13.txt -s ./tshref -a -p
  266 pts/0    S+     0:00 ./tshref -p
  268 pts/0    T      0:00 ./mysplit 4
  269 pts/0    T      0:00 ./mysplit 4
  272 pts/0    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
   10 pts/0    Ss     0:00 -bash
  263 pts/0    S+     0:00 make rtest13
  264 pts/0    S+     0:00 /bin/sh -c ./sdriver.pl -t tra
ce13.txt -s ./tshref -a "-p"
  265 pts/0    S+     0:00 /usr/bin/perl ./sdriver.pl -t
trace13.txt -s ./tshref -a -p
  266 pts/0    S+     0:00 ./tshref -p
  275 pts/0    R      0:00 /bin/ps a
```

测试结论	相同/不同，原因分析如下：相同

4.3.14 测试用例 trace14.txt

tsh 测试结果	tshref 测试结果

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share
/Lab7/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] (283) ./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] (283) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (283) ./myspin 4 &
tsh> jobs
[1] (283) Running ./myspin 4 &
```

```
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share
/Lab7/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] (302) ./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] (302) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (302) ./myspin 4 &
tsh> jobs
[1] (302) Running ./myspin 4 &
```

测试结论	相同/不同，原因分析如下：相同

4.3.15 测试用例 trace15.txt

tsh 测试结果	tshref 测试结果
baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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 [1] (321) terminated by signal 2 tsh> ./myspin 3 & [1] (323) ./myspin 3 & tsh> ./myspin 4 & [2] (325) ./myspin 4 & tsh> jobs [1] (323) Running ./myspin 3 & [2] (325) Running ./myspin 4 & tsh> fg %1 Job [1] (323) stopped by signal 20 tsh> jobs [1] (323) Stopped ./myspin 3 & [2] (325) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (323) ./myspin 3 & tsh> jobs [1] (323) Running ./myspin 3 & [2] (325) Running ./myspin 4 & tsh> fg %1 tsh> quit	baileys@xf1190200708:/mnt/c/Users/Alienware/Desktop/share/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] (341) terminated by signal 2 tsh> ./myspin 3 & [1] (343) ./myspin 3 & tsh> ./myspin 4 & [2] (345) ./myspin 4 & tsh> jobs [1] (343) Running ./myspin 3 & [2] (345) Running ./myspin 4 & tsh> fg %1 Job [1] (343) stopped by signal 20 tsh> jobs [1] (343) Stopped ./myspin 3 & [2] (345) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (343) ./myspin 3 & tsh> jobs [1] (343) Running ./myspin 3 & [2] (345) 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

了解掌握了信号和信号的处理的过程
对信号发送方法、阻塞方法、处理程序的设置方法有了更深的体会
了解了 shell 的工作原理、功能、处理流程

5.2 请给出对本次实验内容的建议

实验 ddl 较紧张

注：本章为酌情加分项。

参考文献

[1] RANDALE.BRYANT, DAVIDR.O‘HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.