

哈尔滨工业大学

实验报告

实验（六）

题 目 TinyShell
 微壳

专 业 计算机类

学 号 1160300901

班 级 1603009

学 生 孙月晴

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2017/12/19

计算机科学与技术学院

目 录

第 1 章 实验基本信息.....	- 4 -
1.1 实验目的.....	- 4 -
1.2 实验环境与工具.....	- 4 -
1.2.1 硬件环境.....	- 4 -
1.2.2 软件环境.....	- 4 -
1.2.3 开发工具.....	- 4 -
1.3 实验预习.....	- 4 -
第 2 章 实验预习.....	- 5 -
2.1 进程的概念、创建和回收方法（5 分）.....	- 5 -
2.2 信号的机制、种类（5 分）.....	- 5 -
2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）.....	- 6 -
2.4 什么是 SHELL，功能和处理流程（5 分）.....	- 7 -
第 3 章 TINY SHELL 的设计与实现.....	- 9 -
3.1.1 VOID EVAL(CHAR *CMDLINE)函数（10 分）.....	- 9 -
3.1.2 INT BUILTIN_CMD(CHAR **ARGV)函数（5 分）.....	- 11 -
3.1.3 VOID DO_BGFG(CHAR **ARGV) 函数（5 分）.....	- 12 -
3.1.4 VOID WAITFG(PID_T PID) 函数（5 分）.....	- 13 -
3.1.5 VOID SIGCHLD_HANDLER(INT SIG) 函数（10 分）.....	- 13 -
第 4 章 TINY SHELL 测试.....	- 28 -
4.1 测试方法.....	- 28 -
4.2 测试结果评价.....	- 28 -
4.3 自测试结果.....	- 28 -
4.3.1 测试用例 trace01.txt 的输出截图（1 分）.....	- 28 -
4.3.2 测试用例 trace02.txt 的输出截图（1 分）.....	- 28 -
4.3.3 测试用例 trace03.txt 的输出截图（1 分）.....	- 29 -
4.3.4 测试用例 trace04.txt 的输出截图（1 分）.....	- 29 -
4.3.5 测试用例 trace05.txt 的输出截图（1 分）.....	- 29 -
4.3.6 测试用例 trace06.txt 的输出截图（1 分）.....	- 30 -
4.3.7 测试用例 trace07.txt 的输出截图（1 分）.....	- 30 -
4.3.8 测试用例 trace08.txt 的输出截图（1 分）.....	- 30 -
4.3.9 测试用例 trace09.txt 的输出截图（1 分）.....	- 31 -
4.3.10 测试用例 trace10.txt 的输出截图（1 分）.....	- 31 -
4.3.11 测试用例 trace11.txt 的输出截图（1 分）.....	- 31 -
4.3.12 测试用例 trace12.txt 的输出截图（1 分）.....	- 32 -
4.3.13 测试用例 trace13.txt 的输出截图（1 分）.....	- 32 -

4.3.14 测试用例 <i>trace14.txt</i> 的输出截图 (1 分)	- 33 -
4.3.15 测试用例 <i>trace15.txt</i> 的输出截图 (1 分)	- 33 -
4.4 自测试评分.....	- 33 -
第 5 章 总结.....	- 34 -
5.1 请总结本次实验的收获.....	- 34 -
5.2 请给出对本次实验内容的建议.....	- 34 -
参考文献.....	- 35 -

第 1 章 实验基本信息

1.1 实验目的

1. 理解现代计算机系统进程与并发的基本知识
2. 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
3. 掌握 shell 的基本原理和实现方法
4. 深入理解 Linux 信号响应可能导致的并发冲突及解决方法
5. 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

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

shlab-handout-hit.tar 实验包

1.3 实验预习

1. 进程的概念、创建和回收方法
2. 信号的机制、种类
3. 信号的发送方法、阻塞方法、处理程序的设置方法
4. shell 的定义，功能和处理流程

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：一个执行中程序的实例

创建进程：父进程通过调用 `fork()` 函数创建一个新的运行的子进程

回收进程：

1. 父进程通过 `wait` 函数回收子进程，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. 父进程通过 `waitpid` 函数回收子进程，等待特定进程，挂起当前进程直到指定进程终止才返回，有多种选项。

2.2 信号的机制、种类（5 分）

信号的机制：一个信号就是一条小消息，它通知进程系统中发生了一个某种类型的事件。内核给一个进程发送软中断信号的方法，是在进程所在的进程表项的信号域设置对应于该信号的位。这里要补充的是，如果信号发送给一个正在睡眠的进程，那么要看该进程进入睡眠的优先级，如果进程睡眠在可被中断的优先级上，则唤醒进程；否则仅设置进程表中信号域相应的位，而不唤醒进程。这一点比较重要，因为进程检查是否收到信号的时机是：一个进程在即将从内核态返回到用户态时；或者，在一个进程要进入或离开一个适当的低调度优先级睡眠状态时。

内核处理一个进程收到的信号的时机是在一个进程从内核态返回用户态时。所以，当一个进程在内核态下运行时，软中断信号并不立即起作用，要等到将返回用户态时才处理。进程只有处理完信号才会返回用户态，进程在用户态下不会有未处理完的信号。

内核处理一个进程收到的软中断信号是在该进程的上下文中，因此，进程必须处于运行状态。当进程接收到一个它忽略的信号时，进程丢弃该信号，就象没有收到该信号似的继续运行。如果进程收到一个要捕捉的信号，那么进程从内核态返回用户态时执行用户定义的函数。而且执行用户定义的函数的方法很巧妙，内核是在用户栈上创建一个新的层，该层中将返回地址的值设置成用户定义的处理函数的地址，这样进程从内核返回弹出栈顶时就返回到用户定义的函数处，从函数返回再弹出栈顶时，才返回原先进入内核的地方。

种类：

序号	名称	默认行为	相应事件
1	SIGHUP	终止	终端线挂断
2	SIGINT	终止	来自键盘的中断
3	SIGQUIT	终止	来自键盘的退出
4	SIGILL	终止	非法指令
5	SIGTRAP	终止并转储内存 ^①	跟踪陷阱
6	SIGABRT	终止并转储内存 ^①	来自 abort 函数的终止信号
7	SIGBUS	终止	总线错误
8	SIGFPE	终止并转储内存 ^①	浮点异常
9	SIGKILL	终止 ^②	杀死程序
10	SIGUSR1	终止	用户定义的信号 1
11	SIGSEGV	终止并转储内存 ^①	无效的内存引用 (段故障)
12	SIGUSR2	终止	用户定义的信号 2
13	SIGPIPE	终止	向一个没有读用户的管道做写操作
14	SIGALRM	终止	来自 alarm 函数的定时器信号
15	SIGTERM	终止	软件终止信号
16	SIGSTKFLT	终止	协处理器上的栈故障
17	SIGCHLD	忽略	一个子进程停止或者终止
18	SIGCONT	忽略	继续进程如果该进程停止
19	SIGSTOP	停止直到下一个 SIGCONT ^②	不是来自终端的停止信号
20	SIGTSTP	停止直到下一个 SIGCONT	来自终端的停止信号
21	SIGTTIN	停止直到下一个 SIGCONT	后台进程从终端读
22	SIGTTOU	停止直到下一个 SIGCONT	后台进程向终端写
23	SIGURG	忽略	套接字上的紧急情况
24	SIGXCPU	终止	CPU 时间限制超出
25	SIGXFSZ	终止	文件大小限制超出
26	SIGVTALRM	终止	虚拟定时器期满
27	SIGPROF	终止	剖析定时器期满
28	SIGWINCH	忽略	窗口大小变化
29	SIGIO	终止	在某个描述符上可执行 I/O 操作
30	SIGPWR	终止	电源故障

2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分）

信号的发送方法:

1. 用 /bin/kill 程序发送信号， /bin/kill 程序可以向另外的进程或进程组发送任意的信号，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。例如，从键盘上输入 ctrl-c (ctrl-z) 会导致内核发送一个 SIGINT (SIGTSTP) 信号到前台进程组中的每个作业；

3. 用 kill 函数发送信号。int kill(pid_t pid,int sig);

如果 pid 大于零，那么 kill 函数发送信号号码 sig 给进程 pid；如果 pid 等于零，那么 kill 发送信号 sig 给调用进程所在进程组中的每个进程，包括调用进程自己；如果 pid 小于零，kill 发送信号 sig 给进程组|pid|（pid 的绝对值）中的每

个进程。

4. 用 alarm 函数发送信号。Unsigned int alarm(unsigned int secs);

alarm 函数安排内核在 secs 秒后发送一个 SIGALRM 信号给调用进程。如果 secs 为零，那么不会调度安排新的闹钟（alarm）。在任何情况下，对 alarm 的调用都将取消任何待处理的闹钟，并且返回任何待处理的闹钟在被发送前还剩下的秒数；如果没有任何待处理的闹钟就返回零。

信号的阻塞方法：Linux 提供阻塞信号的隐式和显式的机制

1. 隐式阻塞机制。内核默认阻塞任何当前处理程序正在处理信号类型的待处理的信号。

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

处理程序的设置方法：通过把处理程序的地址传递给 signal 函数从而改变默认行为，这叫做设置信号处理程序。

进程可以使用 signal 函数修改和信号 signalnum 相关联的默认行为

handler_t *signal(int signalnum, handler_t *handler)

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

1. 如果 handler 是 SIG_IGN，那么忽略类型为 signalnum 的信号；
2. 如果 handler 是 SIG_DFL，那么类型为 signalnum 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，这个函数被称为信号处理程序，只要接收到一个类型为 signalnum 的信号，就会调用这个程序。

2.4 什么是 shell，功能和处理流程（5 分）

1. **定义：**Shell 是一个交互型的应用级程序，它代表用户运行其他程序。

2. **功能：**Shell 会打印出提示符，等待来自 stdin 的输入，根据输入执行特定地操作，这样就产生了一种错觉，似乎输入的文字（命令行）控制了程序的执行。

3. **处理流程：**命令行是一串 ASCII 字符由空格分隔。字符串的第一个单词是一个可执行程序，或者是 shell 的内置命令。命令行的其余部分是命令的参数。如果第一个单词是内置命令，shell 会立即在当前进程中执行。否则，shell 会新建一个子进程，然后再子进程中执行程序。新建的子进程又叫做作业。通常，作业可以由 Unix 管道连接的多个子进程组成。

如果命令行以 & 符号结尾，那么作业将在后台运行，这意味着在打印提示符并等待下一个命令之前，shell 不会等待作业终止。否则，作业在前台运行，这意味着 shell 在作业终止前不会执行下一条命令行。因此，在任何时候，最多可以在一个作业中运行在前台。但是，任意数量的作业可以在后台运行。例如，键入命令行：sh> jobs，会让 shell 运行内置命令 jobs。

键入命令行 `sh> /bin/ls -l -d` 会导致 shell 在前台运行 `ls` 程序。根据约定, shell 会执行程序的 `main` 函数 `int main(int argc, char *argv[])`

`argc` 和 `argv` 会接收到下面的值:

```
argc == 3,  
argv[0] == "/bin/ls",  
argv[1] == "-l",  
argv[2] == "-d".
```

下面以 `&` 结尾的命令行会在后台执行 `ls` 程序

```
sh> /bin/ls -l -d &
```

Unix shell 支持作业控制的概念, 允许用户在前台和后台之间来回移动作业, 并更改进程的状态(运行, 停止或终止)。在作业运行时, 键入 `ctrl-c` 会将 `SIGINT` 信号传递到前台作业中的每个进程。`SIGINT` 的默认动作是终止进程。类似地, 键入 `ctrl-z` 会导致 `SIGTSTP` 信号传递给所有前台进程。`SIGTSTP` 的默认操作是停止进程, 直到它被 `SIGCONT` 信号唤醒为止。Unix shell 还支持作业控制的各种内置命令。例如:

`jobs`: 列出运行和停止的后台作业。

`bg <job>`: 将停止的后台作业更改为正在运行的后台作业。

`fg <job>`: 将停止或运行的后台作业更改为在前台运行。

`kill <job>`: 终止作业。

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，评估用户刚输入的命令行。

参 数：cmdline

处理流程：

1. 判断是否为内部函数，如果是内部函数，直接执行 builtin_cmd 函数就行。如果不是内部函数，需要 Fork 出一个新的进程去执行相应的函数。
2. 函数框架提供了 parseline 命令行解析工具，即可以通过该解析函数来得到命令行参数。如果不是内部函数，首先要先将 SIGCHLD 信号阻塞住，以防出现竞争条件。
3. 子进程解决信号阻塞，并执行相关函数。
4. 父进程要判断子进程是前台进程还是后台进程，如果是前台进程，则调用 waitpid 来等待前台进程，如果是后台进程，则打印出相关进程信息。同时，把新添加的进程利用 addjob 添加到工作组中。

要点分析：

1. 如果用户请求了一个内置的命令 (quit, jobs, bg 或者 fg)，那么立即执行它。否则，派生一个子进程并在子进程中运行这个 job。如果作业在前台运行，等待它终止，然后返回。
2. 每个子进程必须有自己独一无二的进程组 id，通过在 fork() 之后的子进程中 Setpgid(0,0) 实现，这样当向前台程序发送 ctrl+c 或 ctrl+z 命令时，才不会影响到后台程序。如果没有这一步，则所有的子进程与当前的 tsh shell 进程为同一个进程组，发送信号时，前后台的子进程均会收到。
3. 在 fork() 新进程前后要阻塞 SIGCHLD 信号，防止出现竞争，这是经典的同步错误，如果不阻塞会出现子进程先结束从 jobs 中删除，然后再执行到主进程 addjob 的竞争问题。

代码实现：

```
void eval(char *cmdline)
{
    /* $begin handout */
    char *argv[MAXARGS]; /* argv for execve() */
    int bg;               /* should the job run in bg or fg? */
    pid_t pid;            /* process id */
    sigset_t mask;        /* signal mask */

    /* Parse command line */
    bg = parseline(cmdline, argv);
```

```
if (argv[0] == NULL)
return;    /* ignore empty lines */

if (!builtin_cmd(argv)) {

    /*
    * This is a little tricky. Block SIGCHLD, SIGINT, and SIGTSTP
    * signals until we can add the job to the job list. This
    * eliminates some nasty races between adding a job to the job
    * list and the arrival of SIGCHLD, SIGINT, and SIGTSTP signals.
    * 阻止 SIGCHLD, SIGINT 和 SIGTSTP 信号，直到我们可以将作业添加到作业列表。
    */

    if (sigemptyset(&mask) < 0)
        unix_error("sigemptyset error");
    if (sigaddset(&mask, SIGCHLD))
        unix_error("sigaddset error");
    if (sigaddset(&mask, SIGINT))
        unix_error("sigaddset error");
    if (sigaddset(&mask, SIGTSTP))
        unix_error("sigaddset error");
    if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0)
        unix_error("sigprocmask error");

    /* Create a child process */
    if ((pid = fork()) < 0)
        unix_error("fork error");

    /*
    * Child process
    */

    if (pid == 0) {
        /* Child unblocks signals 取消阻塞信号 */
        sigprocmask(SIG_UNBLOCK, &mask, NULL);

        /* Each new job must get a new process group ID
        so that the kernel doesn't send ctrl-c and ctrl-z
        signals to all of the shell's jobs
        每个新作业必须得到一个新的进程组 ID，
        以便内核不会向所有 shell 的作业发送 ctrl-c 和 ctrl-z 信号*/
        if (setpgid(0, 0) < 0)
            unix_error("setpgid error");

        /* Now load and run the program in the new job
        * 在新的作业中加载并运行程序
        */
        if (execve(argv[0], argv, environ) < 0) {
            printf("%s: Command not found\n", argv[0]);
            exit(0);
        }
    }
}
```

```

/*
 * Parent process
 */

/* Parent adds the job, and then unblocks signals so that
   the signals handlers can run again
 * 父进程添加作业，然后解除阻塞信号，以便信号处理程序可以再次运行
 */
addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
sigprocmask(SIG_UNBLOCK, &mask, NULL);

if (!bg)
    waitfg(pid);
else
    printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
}
/* Send handout */
return;
}

```

3. 1.2 int builtin_cmd(char **argv) 函数 (5 分)

函数功能：该函数检查第一个命令行参数是否是一个内置的 shell 命令，即识别并解释内置命令: quit, fg, bg, 和 jobs.

参 数：argv

处理流程：如果第一个命令行参数是一个内置的 shell 命令，它就立即解释这个命令，并返回 1，否则返回 0。

要点分析：

1. 当命令行参数为 quit 时，直接终止 shell
2. 当命令行参数为 jobs 时，调用 listjobs 函数，显示 job 列表
3. 当命令行参数为 bg 或 fg 时，调用 do_bgfg 函数，执行内置的 bg 和 fg 命令
4. 当命令行参数为&时，忽略单个的&直接返回 1；
5. 不是内置命令时返回 0

代码实现：

```

int builtin_cmd(char **argv)
{
    if(!strcmp(argv[0], "quit")) /* quit 命令，直接终止 shell */
        exit(0);
    if(!strcmp(argv[0], "jobs")){
        listjobs(jobs);
        return 1;
    }
    if(!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")){
        do_bgfg(argv);
        return 1;
    }
    if(!strcmp(argv[0], "&"))
        return 1;
    return 0; /* 不是内置命令 */
}

```

}

3. 1.3 void do_bgfg(char **argv) 函数 (5 分)

函数功能: 实现内置命令 bg 和 fg

参 数: argv

处理流程:

1. 先做命令错误判断, fg 和 bg 后面是否有参数, 参数是否符合%+数字或者数字, 所表示的进程是否为正在运行的进程;
2. 做完这些判断之后, 根据如果是%号, 说明取的是工作组号, 如果直接是数字说明取的是进程号, 根据工作组号和进程号获取对应的 job 结构体;
3. 接下来如果是 bg, 说明要恢复成后台进程, 即改变 job 的 state; 如果是 fg, 说明要恢复成前台进程, 即改变 job 的 state, 然后调用 waitfg; 等前台进程运行结束。

要点分析:

做命令判断后, 看 fg 和 bg 后面的参数, 直接是数字则是工程组号, 如果是%说明取的是进程号, 然后如果是 bg 则恢复成后台进程, 如果是 fg 恢复成前台进程。

代码实现:

```
void do_bgfg(char **argv)
{
    /* $begin handout */
    struct job_t *jobp=NULL;

    /* Ignore command if no argument */
    if (argv[1] == NULL) {
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }

    /* Parse the required PID or %JID arg */
    if (isdigit(argv[1][0])) { /* 如果是数字, 取的是进程号 */
        pid_t pid = atoi(argv[1]);
        if (!(jobp = getjobpid(jobs, pid))) {
            printf("(%d): No such process\n", pid);
            return;
        }
    }
    else if (argv[1][0] == '%') { /* 如果是%, 取的是工程组号 */
        int jid = atoi(&argv[1][1]);
        if (!(jobp = getjobjid(jobs, jid))) {
            printf("%s: No such job\n", argv[1]);
            return;
        }
    }
    else {
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }

    /* bg command */
}
```

```

if (!strcmp(argv[0], "bg")) { /*如果是 bg, 恢复成后台进程 */
if (kill(-(jobp->pid), SIGCONT) < 0)
    unix_error("kill (bg) error");
jobp->state = BG;
printf("[%d] (%d) %s", jobp->jid, jobp->pid, jobp->cmdline);
}

/* fg command */
else if (!strcmp(argv[0], "fg")) { /*如果是 fg, 恢复成前台进程 */
if (kill(-(jobp->pid), SIGCONT) < 0)
    unix_error("kill (fg) error");
jobp->state = FG;
waitfg(jobp->pid);
}
else {
printf("do_bgfg: Internal error\n");
exit(0);
}
/* $end handout */
return;
}

```

3. 1.4 void waitfg(pid_t pid) 函数 (5 分)

函数功能: 等待一个前台作业结束

参 数: pid

处理流程: 主进程 sleep 直到前台进程结束。

要点分析: 前台进程与后台进程的唯一区别是 shell 会等待前台进程, 因此前台进程只有一个。waitfg 实现了这一等待的功能。最显而易见的选择是用 waitpid 等待前台进程的结束。最佳选择是使用 sleep 函数, 只要前台进程仍然是需要等待的进程, 主进程就 sleep。那么 sleep 多长时间呢, sleep(0)是最佳的选择, 0 表示进程会让其它进程来执行, 如果没有其它的进程在执行会继续执行。这样总会有进程再执行, 而不会出现 CPU 空转的情况。

```

void waitfg(pid_t pid)
{
    while(pid == fgpid(jobs)){
        sleep(0);
    }
    return;
}

```

3. 1.5 void sigchld_handler(int sig) 函数 (10 分)

函数功能: 捕获 SIGCHLD 信号

参 数: sig

处理流程: 父进程用 waitpid 作为 while 循环的测试条件, 等待它所有的子进程终止。因为第一个参数是-1, 所以对 waitpid 的调用会阻塞, 直到任意一个子进程终止, 在每个子进程终止时, 对 waitpid 的调用会返回, 返回值是该子进程的非零的

PID

要点分析：内核发送 SIGCHLD 当孩子作业终止（变成僵尸），或由于收到 SIGSTOP 或 SIGTSTP 信号停止。该处理程序收集所有可用的僵尸孩子，但不等待任何其他正在运行的孩子终止。

```
void sigchld_handler(int sig)//处理子进程
{
    pid_t pid;
    while ((pid = waitpid(-1, NULL, WNOHANG)) > 0) {

        deletejob(jobs, pid);
    }
    return;
}
```

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
 *
 * <孙月晴-1160300901>
 */
#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 同一时刻最大工作量*/
#define MAXJID 1<<16 /* max job ID */

/* Job states */
#define UNDEF 0 /* undefined 未定义的*/
#define FG 1 /* running in foreground 在前台运行*/
#define BG 2 /* running in background 在后台运行*/
#define ST 3 /* stopped 停止*/

/*
 * Jobs states: FG (foreground), BG (background), ST (stopped)
 * Job state transitions and enabling actions:
 */
```

```

*      FG -> ST   : ctrl-z
*      ST -> FG   : fg command
*      ST -> BG   : bg command
*      BG -> FG   : fg command
* At most 1 job can be in the FG state.
*/

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

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

/* Function prototypes */

/* Here are the functions that you will implement */
void eval(char *cmdline);
//这个函数解析了以空格分割的命令行参数，并构造最终会传递给 execve 的 argv 参数
int builtin_cmd(char **argv);
//这里所谓的内置外壳命令是 quit, jobs, bg, fg。如果第一个不是内置命令，则返回 0
void do_bgfg(char **argv);

void waitfg(pid_t pid); //等待处理 PID
void sigchld_handler(int sig); //处理 SIGCHLD 信号--这个信号处理程序回收子进程--如果子进程
//是僵尸进程
//或者收到 stop 之类的信号时，调用
void sigtstp_handler(int sig); //处理 SIGTSTP 信号---停止直到下一个 SIGCONT---键盘输入
CTRL+z 会触发这个信号
void sigint_handler(int sig); //处理 SIGINT 信号--在键盘上输入 CTRL+C 会导致一个 SIGINT 信
//号被发送到外壳。
//外壳捕获该信号，然后发送 SIGINT 信号到这个前台进程组中的每个进程

/* Here are helper routines that we've provided for you */
int parseline(const char *cmdline, char **argv);
/*解析命令，并构建 argv 数组，括号中的单引号字符被视为一个单独的参数。如果用户请求
后台工作返回 true，
请求前台工作，返回 false。即最后一个参数是"&"，表示后台运行，否则前台
*/
void sigquit_handler(int sig);

```

//驱动程序可以优雅地终止,通过发送一个 SIGQUIT 信号子 shell。

```

void clearjob(struct job_t *job);
void initjobs(struct job_t *jobs);
int maxjid(struct job_t *jobs); //返回最大分配的 job id
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);
int deletejob(struct job_t *jobs, pid_t pid);
pid_t fgpid(struct job_t *jobs); //返回现在在前台工作的任务的 pid, 如果没有返回 0
struct job_t *getjobpid(struct job_t *jobs, pid_t pid); //在工作组里, 寻找指定 pid 的进程。如果没有返回 NULL
struct job_t *getjobjid(struct job_t *jobs, int jid);
int pid2jid(pid_t pid); //得到线程的进程编号
void listjobs(struct job_t *jobs);

void usage(void);
void unix_error(char *msg); //unix 方面的错误处理
void app_error(char *msg); //应用方面的错误处理
typedef void handler_t(int); // 定义了一个类型——handler_t, 表示返回值为 void 型
// (参数为 int 型) 的函数 (的) 指针。 它可以用来声明一个或多个函数指针。
handler_t *Signal(int signum, handler_t *handler); //信号包装处理

/*
 * main - The shell's main routine
 */
int main(int argc, char **argv)
{
    char c;
    char cmdline[MAXLINE];
    int emit_prompt = 1; /* emit prompt (default) */

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

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

    /* Install the signal handlers */

```



```

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

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

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

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

/* Read command line */
if (emit_prompt) {
    printf("%s", prompt);
    fflush(stdout);
    //清除读写缓冲区, 如果成功刷新,fflush 返回0。指定的流没有缓冲区或者只读打开时
    //也返回0 值。返回EOF 指出一个错
}
if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
    app_error("fgets error");
if (feof(stdin)) { /* End of file (ctrl-d) */
    fflush(stdout);
    exit(0);
}

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

exit(0); /* control never reaches here 一般不会到达这里*/
}

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

```

```

pid_t pid;           /* process id */
sigset_t mask;       /* signal mask */

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

if (!builtin_cmd(argv)) {

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

    if (sigemptyset(&mask) < 0) //sigemptyset() 用来将参数 set(mask) 信号集初始化并清空。
        unix_error("sigemptyset error");
    if (sigaddset(&mask, SIGCHLD)) //sigaddset() 用来将参数 signal(sigchld) 代表的信号加入至
参数 set(mask) 信号集里
        unix_error("sigaddset error");
    if (sigaddset(&mask, SIGINT))
        unix_error("sigaddset error");
    if (sigaddset(&mask, SIGTSTP))
        unix_error("sigaddset error");
    if (sigprocmask(SIG_BLOCK, &mask, NULL) < 0) //用于改变进程的当前阻塞信号集,也可以
用来检测当前进程的信号掩码
        unix_error("sigprocmask error");

    /* Create a child process */
    if ((pid = fork()) < 0)
        unix_error("fork error");

    /*
     * Child process
     */

    if (pid == 0) {
        /* Child unblocks signals */
        sigprocmask(SIG_UNBLOCK, &mask, NULL);

        /* Each new job must get a new process group ID
         so that the kernel doesn't send ctrl-c and ctrl-z
         signals to all of the shell's jobs */
        if (setpgid(0, 0) < 0)
            unix_error("setpgid error");

        /* Now load and run the program in the new job */
        if (execve(argv[0], argv, environ) < 0) {
            printf("%s: Command not found\n", argv[0]);
            exit(0);
        }
    }
}

```

```

    }

    /*
    * Parent process
    */

    /* Parent adds the job, and then unblocks signals so that
    the signals handlers can run again */
    addjob(jobs, pid, (bg == 1 ? BG : FG), cmdline);
    sigprocmask(SIG_UNBLOCK, &mask, NULL);

    if (!bg)
        waitfg(pid);
    else
        printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
    }
    /* Send handout */
    return;
}

/*
* parseline - Parse the command line and build the argv array.
*
* Characters enclosed in single quotes are treated as a single
* argument.  Return true if the user has requested a BG job, false if
* the user has requested a FG job.
*/
int parseline(const char *cmdline, char **argv)
{
    static char array[MAXLINE]; /* holds local copy of command line */
    char *buf = array;          /* ptr that traverses command line */
    char *delim;                 /* points to first space delimiter */
    int argc;                    /* number of args */
    int bg;                      /* background job? */

    strcpy(buf, cmdline);
    buf[strlen(buf)-1] = ' '; /* replace trailing '\n' with space */
    while (*buf && (*buf == ' ')) /* ignore leading spaces */
        buf++;

    /* Build the argv list */
    argc = 0;
    if (*buf == '\n') {
        buf++;
        delim = strchr(buf, '\n');
    }
    else {
        delim = strchr(buf, ' ');
    }

    while (delim) {
        argv[argc++] = buf;
        *delim = '\0';
        buf = delim + 1;
    }
}

```

```

while (*buf && (*buf == ' ')) /* ignore spaces */
    buf++;

if (*buf == '\\') {
    buf++;
    delim = strchr(buf, '\\');
}
else {
    delim = strchr(buf, ' ');
}
}
argv[argc] = NULL;

if (argc == 0) /* ignore blank line */
return 1;

/* should the job run in the background? */
if ((bg = (*argv[argc-1] == '&')) != 0) {
    argv[--argc] = NULL;
}
return bg;
}

/*
 * builtin_cmd - If the user has typed a built-in command then execute
 * it immediately.
 */
int builtin_cmd(char **argv) {
    if (!strcmp(argv[0], "quit")) /* quit 命令，直接终止 shell */
        exit(0);
    if (!strcmp(argv[0], "jobs")) {
        listjobs(jobs);
        return 1;
    }
    if (!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg")) {
        do_bgfg(argv);
        return 1;
    }
    if (!strcmp(argv[0], "&"))
        return 1;
    return 0; /* 不是内置命令 */
}

/*
 * do_bgfg - Execute the builtin bg and fg commands
 */
void do_bgfg(char **argv)
{
    /* $begin handout */
    struct job_t *jobp=NULL;

    /* Ignore command if no argument */
    if (argv[1] == NULL) {

```

```

printf("%s command requires PID or %%jobid argument\n", argv[0]);
return;
}

/* Parse the required PID or %JID arg */
if (isdigit(argv[1][0])) {
    pid_t pid = atoi(argv[1]);
    if (!jobp = getjobpid(jobs, pid)) {
        printf("(%d): No such process\n", pid);
        return;
    }
}
else if (argv[1][0] == '%') {
    int jid = atoi(&argv[1][1]);
    if (!jobp = getjobjid(jobs, jid)) {
        printf("%s: No such job\n", argv[1]);
        return;
    }
}
else {
    printf("%s: argument must be a PID or %%jobid\n", argv[0]);
    return;
}

/* bg command */
if (!strcmp(argv[0], "bg")) {
    if (kill(-(jobp->pid), SIGCONT) < 0)
        unix_error("kill (bg) error");
    jobp->state = BG;
    printf("[%d] (%d) %s", jobp->jid, jobp->pid, jobp->cmdline);
}

/* fg command */
else if (!strcmp(argv[0], "fg")) {
    if (kill(-(jobp->pid), SIGCONT) < 0)
        unix_error("kill (fg) error");
    jobp->state = FG;
    waitfg(jobp->pid);
}
else {
    printf("do_bgfg: Internal error\n");
    exit(0);
}

/* Send handout */
return;
}

/*
 * waitfg - Block until process pid is no longer the foreground process
 */

void waitfg(pid_t pid)
{
    while(pid==fgpid(jobs)){

```

```

    sleep(0);
    }
    return;
}
/*****
 * Signal handlers
 *****/

/*
 * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
 *                   a child job terminates (becomes a zombie), or stops because it
 *                   received a SIGSTOP or SIGTSTP signal. The handler reaps all
 *                   available zombie children, but doesn't wait for any other
 *                   currently running children to terminate.
 */
void sigchld_handler(int sig)//处理子进程
{
    pid_t pid;
    while ((pid = waitpid(-1, NULL, WNOHANG)) > 0) {

        deletejob(jobs, pid);
    }
    return;
}

/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenever the
 *                  user types ctrl-c at the keyboard. Catch it and send it along
 *                  to the foreground job.
 */
void sigint_handler(int sig)//处理int 信号
{
    pid_t pid;
    int jid;
    pid = fgpid(jobs);
    jid = pid2jid(pid);
    if (kill(-pid, SIGINT) < 0) {
        fprintf(stdout, "kill error");
        return;
    }
    else
        printf("JID [%d] (%d) terminated by signal %d\n", jid, pid, SIGINT);

    return;
}

/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *                   the user types ctrl-z at the keyboard. Catch it and suspend the
 *                   foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig)//处理 SIGTSTP 信号
{

```

```

    pid_t pid;
    int jid;
    pid = fgpid(jobs);
    if (kill(-pid, SIGTSTP) < 0) {
        fprintf(stdout, "kill error");
        return;
    }
    jid = pid2jid(pid);
    jobs[jid-1].state = ST;
    printf("JID [%d] (%d) stopped by signal %d\n", jid, pid, SIGTSTP);

    return;
}

/*****
 * End signal handlers
 *****/

/*****
 * Helper routines that manipulate the job list
 *****/

/* clearjob - Clear the entries in a job struct */
void clearjob(struct job_t *job) { //清空工作列表
    job->pid = 0;
    job->jid = 0;
    job->state = UNDEF;
    job->cmdline[0] = '\0';
}

/* initjobs - Initialize the job list */
void initjobs(struct job_t *jobs) { //初始化
    int i;

    for (i = 0; i < MAXJOBS; i++)
        clearjob(&jobs[i]);
}

/* maxjid - Returns largest allocated job ID */
int maxjid(struct job_t *jobs) //返回最大的已分配的工作ID
{
    int i, max=0;

    for (i = 0; i < MAXJOBS; i++)
        if (jobs[i].jid > max)
            max = jobs[i].jid;
    return max;
}

/* addjob - Add a job to the job list */
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline)
{ //添加一个任务 (工作)
    int i;

```

```

    if (pid < 1) // 错误处理, 一般 PID 都大于 0
    return 0;

    for (i = 0; i < MAXJOBS; i++) { // 找到一个空的 jobs 组
    if (jobs[i].pid == 0) {
        jobs[i].pid = pid;
        jobs[i].state = state;
        jobs[i].jid = nextjid++;
        if (nextjid > MAXJOBS)
            nextjid = 1;
        strcpy(jobs[i].cmdline, cmdline);
        if (verbose) {
            printf("Added job [%d] %d %s\n", jobs[i].jid, jobs[i].pid, jobs[i].cmdline);
        }
        return 1;
    }
    }
    printf("Tried to create too many jobs\n");
    return 0;
}

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

    if (pid < 1)
    return 0;

    for (i = 0; i < MAXJOBS; i++) {
    if (jobs[i].pid == pid) {
        clearjob(&jobs[i]);
        nextjid = maxjid(jobs)+1;
        return 1;
    }
    }
    return 0;
}

/* fgpid - Return PID of current foreground job, 0 if no such job */
pid_t fgpid(struct job_t *jobs) {
    int i;

    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].state == FG)
        return jobs[i].pid;
    return 0;
}

/* getjobpid - Find a job (by PID) on the job list */
struct job_t *getjobpid(struct job_t *jobs, pid_t pid) {
    int i;

```



```

    if (pid < 1)
    return NULL;
    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].pid == pid)
        return &jobs[i];
    return NULL;
}

/* getjobjid - Find a job (by JID) on the job list */
struct job_t *getjobjid(struct job_t *jobs, int jid)
{
    int i;

    if (jid < 1)
    return NULL;
    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].jid == jid)
        return &jobs[i];
    return NULL;
}

/* pid2jid - Map process ID to job ID */
int pid2jid(pid_t pid)
{
    int i;

    if (pid < 1)
    return 0;
    for (i = 0; i < MAXJOBS; i++)
    if (jobs[i].pid == pid) {
        return jobs[i].jid;
    }
    return 0;
}

/* listjobs - Print the job list */
void listjobs(struct job_t *jobs)
{
    int i;

    for (i = 0; i < MAXJOBS; i++) {
    if (jobs[i].pid != 0) {
        printf("[%d] (%d) ", jobs[i].jid, jobs[i].pid);
        switch (jobs[i].state) {
        case BG:
            printf("Running ");
            break;
        case FG:
            printf("Foreground ");
            break;
        case ST:
            printf("Stopped ");
            break;
        }
    }
    }
}

```

```

    default:
        printf("listjobs: Internal error: job[%d].state=%d ",
            i, jobs[i].state);
    }
    printf("%s", jobs[i].cmdline);
}
}
}
/*****
 * end job list helper routines
 *****/

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

/*
 * usage - print a help message
 */
void usage(void)
{
    printf("Usage: shell [-hvp]\n");
    printf("  -h   print this message\n");
    printf("  -v   print additional diagnostic information\n");
    printf("  -p   do not emit a command prompt\n");
    exit(1);
}

/*
 * unix_error - unix-style error routine
 */
void unix_error(char *msg)
{
    fprintf(stdout, "%s: %s\n", msg, strerror(errno));
    exit(1);
}

/*
 * app_error - application-style error routine
 */
void app_error(char *msg)
{
    fprintf(stdout, "%s\n", msg);
    exit(1);
}

/*
 * Signal - wrapper for the sigaction function
 */
handler_t *Signal(int signum, handler_t *handler)
{
    struct sigaction action, old_action;

```

```

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

    if (sigaction(signum, &action, &old_action) < 0)
        unix_error("Signal error");
    return (old_action.sa_handler);
}

/*
 * sigquit_handler - The driver program can gracefully terminate the
 *                   child shell by sending it a SIGQUIT signal.
 */
void sigquit_handler(int sig)
{
    printf("Terminating after receipt of SIGQUIT signal\n");
    exit(1);
}

```

第 4 章 TinyShell 测试

总分 15 分

4.1 测试方法

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

4.2 测试结果评价

tsh 与 tshref 的输出在一下两个方面可以不同:

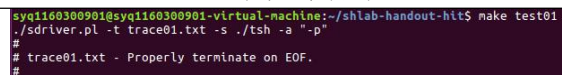
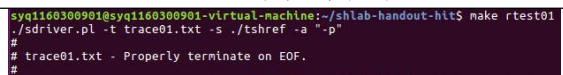
(1) PID

(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

4.3.1 测试用例 trace01.txt 的输出截图 (1 分)

tsh 测试结果		tshref 测试结果	
			
测试结论	相同		

4.3.2 测试用例 trace02.txt 的输出截图 (1 分)

tsh 测试结果	tshref 测试结果
----------	-------------

<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test02 ./sdriver.pl -t trace02.txt -s ./tsh -a "-p" # # trace02.txt - Process builtin quit command. </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest02 ./sdriver.pl -t trace02.txt -s ./tshref -a "-p" # # trace02.txt - Process builtin quit command. </pre>
测试结论	相同

4.3.3 测试用例 trace03.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test03 ./sdriver.pl -t trace03.txt -s ./tsh -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest03 ./sdriver.pl -t trace03.txt -s ./tshref -a "-p" # # trace03.txt - Run a foreground job. # tsh> quit </pre>
测试结论	相同

4.3.4 测试用例 trace04.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test04 ./sdriver.pl -t trace04.txt -s ./tsh -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (5701) ./myspin 1 & </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest04 ./sdriver.pl -t trace04.txt -s ./tshref -a "-p" # # trace04.txt - Run a background job. # tsh> ./myspin 1 & [1] (5707) ./myspin 1 & </pre>
测试结论	相同

4.3.5 测试用例 trace05.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
----------	-------------

<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test05 ./sdriver.pl -t trace05.txt -s ./tsh -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (5713) ./myspin 2 & tsh> ./myspin 3 & [2] (5715) ./myspin 3 & tsh> jobs [1] (5713) Running ./myspin 2 & [2] (5715) Running ./myspin 3 & </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest05 ./sdriver.pl -t trace05.txt -s ./tshref -a "-p" # # trace05.txt - Process jobs builtin command. # tsh> ./myspin 2 & [1] (5722) ./myspin 2 & tsh> ./myspin 3 & [2] (5724) ./myspin 3 & tsh> jobs [1] (5722) Running ./myspin 2 & [2] (5724) Running ./myspin 3 & </pre>
测试结论	相同

4.3.6 测试用例 trace06.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test06 ./sdriver.pl -t trace06.txt -s ./tsh -a "-p" # # trace06.txt - Forward SIGINT to foreground job. # tsh> ./myspin 4 JID [1] (5733) terminated by signal 2 </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5741) terminated by signal 2 </pre>
测试结论	相同

4.3.7 测试用例 trace07.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5747) ./myspin 4 & tsh> ./myspin 5 JID [2] (5749) terminated by signal 2 tsh> jobs [1] (5747) Running ./myspin 4 & </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5756) ./myspin 4 & tsh> ./myspin 5 Job [2] (5758) terminated by signal 2 tsh> jobs [1] (5756) Running ./myspin 4 & </pre>
测试结论	相同

4.3.8 测试用例 trace08.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
----------	-------------

<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5765) ./myspin 4 & tsh> ./myspin 5 JID [2] (5767) stopped by signal 20 tsh> jobs [1] (5765) Running ./myspin 4 & [2] (5767) Stopped ./myspin 5 </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5774) ./myspin 4 & tsh> ./myspin 5 Job [2] (5776) stopped by signal 20 tsh> jobs [1] (5774) Running ./myspin 4 & [2] (5776) Stopped ./myspin 5 </pre>
测试结论	相同

4.3.9 测试用例 trace09.txt 的输出截图（1 分）

<p style="text-align: center;">tsh 测试结果</p> <pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test09 ./sdriver.pl -t trace09.txt -s ./tsh -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (5785) ./myspin 4 & tsh> ./myspin 5 JID [2] (5787) stopped by signal 20 tsh> jobs [1] (5785) Running ./myspin 4 & [2] (5787) Stopped ./myspin 5 tsh> bg %2 [2] (5787) ./myspin 5 tsh> jobs [1] (5785) Running ./myspin 4 & [2] (5787) Running ./myspin 5 </pre>	<p style="text-align: center;">tshref 测试结果</p> <pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest09 ./sdriver.pl -t trace09.txt -s ./tshref -a "-p" # # trace09.txt - Process bg builtin command # tsh> ./myspin 4 & [1] (5796) ./myspin 4 & tsh> ./myspin 5 Job [2] (5798) stopped by signal 20 tsh> jobs [1] (5796) Running ./myspin 4 & [2] (5798) Stopped ./myspin 5 tsh> bg %2 [2] (5798) ./myspin 5 tsh> jobs [1] (5796) Running ./myspin 4 & [2] (5798) Running ./myspin 5 </pre>
测试结论	相同

4.3.10 测试用例 trace10.txt 的输出截图（1 分）

<p style="text-align: center;">tsh 测试结果</p> <pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make test10 ./sdriver.pl -t trace10.txt -s ./tsh -a "-p" # # trace10.txt - Process fg builtin command. # tsh> ./myspin 4 & [1] (5807) ./myspin 4 & tsh> fg %1 JID [1] (5807) stopped by signal 20 tsh> jobs [1] (5807) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs </pre>	<p style="text-align: center;">tshref 测试结果</p> <pre> syq1160300901@syq1160300901-virtual-machine:~/shlab-handout-hit\$ make rtest10 ./sdriver.pl -t trace10.txt -s ./tshref -a "-p" # # trace10.txt - Process fg builtin command. # tsh> ./myspin 4 & [1] (5817) ./myspin 4 & tsh> fg %1 Job [1] (5817) stopped by signal 20 tsh> jobs [1] (5817) Stopped ./myspin 4 & tsh> fg %1 tsh> jobs </pre>
测试结论	相同

4.3.11 测试用例 trace11.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
----------	-------------

<pre>sq1160300901@sq1160300901-virtual-machine:~/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 J10 [1] (5829) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:16 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5604 pts/4 S+ 0:00 make test11 5825 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p" 5826 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p 5827 pts/4 S+ 0:00 ./tsh -p 5832 pts/4 R 0:00 /bin/ps a</pre>	<pre>sq1160300901@sq1160300901-virtual-machine:~/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] (5838) terminated by signal 2 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:16 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5604 pts/4 S+ 0:00 make rtest11 5834 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p" 5835 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p 5840 pts/4 S+ 0:00 ./tshref -p 5841 pts/4 R 0:00 /bin/ps a</pre>
--	--

测试结论 相同/不同，原因分析如下：

4.3.12 测试用例 trace12.txt 的输出截图（1 分）

<p>tsh 测试结果</p> <pre>sq1160300901@sq1160300901-virtual-machine:~/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 J10 [1] (5847) stopped by signal 20 tsh> jobs [1] (5847) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:18 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5842 pts/4 S+ 0:00 make test12 5843 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p" 5844 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p 5845 pts/4 S+ 0:00 ./tsh -p 5847 pts/4 T 0:00 ./mysplit 4 5848 pts/4 T 0:00 ./mysplit 4 5851 pts/4 R 0:00 /bin/ps a</pre>	<p>tshref 测试结果</p> <pre>sq1160300901@sq1160300901-virtual-machine:~/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] (5857) stopped by signal 20 tsh> jobs [1] (5857) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:18 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5852 pts/4 S+ 0:00 make rtest12 5853 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p" 5854 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p 5855 pts/4 S+ 0:00 ./tshref -p 5857 pts/4 T 0:00 ./mysplit 4 5858 pts/4 T 0:00 ./mysplit 4 5861 pts/4 R 0:00 /bin/ps a</pre>
---	--

测试结论 相同/不同，原因分析如下：

4.3.13 测试用例 trace13.txt 的输出截图（1 分）

<p>tsh 测试结果</p> <pre>sq1160300901@sq1160300901-virtual-machine:~/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 J10 [1] (5873) stopped by signal 20 tsh> jobs [1] (5873) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:20 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5868 pts/4 S+ 0:00 make test13 5869 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 5870 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 5871 pts/4 S+ 0:00 ./tsh -p 5873 pts/4 T 0:00 ./mysplit 4 5874 pts/4 R 0:00 /bin/ps a tsh> fg %1 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:20 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5868 pts/4 S+ 0:00 make test13 5869 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p" 5870 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p 5871 pts/4 S+ 0:00 ./tshref -p 5873 pts/4 T 0:00 ./mysplit 4 5874 pts/4 R 0:00 /bin/ps a</pre>	<p>tshref 测试结果</p> <pre>sq1160300901@sq1160300901-virtual-machine:~/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] (5886) stopped by signal 20 tsh> jobs [1] (5886) Stopped ./mysplit 4 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:21 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5881 pts/4 S+ 0:00 make rtest13 5882 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 5883 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 5884 pts/4 S+ 0:00 ./tshref -p 5886 pts/4 T 0:00 ./mysplit 4 5887 pts/4 T 0:00 ./mysplit 4 5890 pts/4 R 0:00 /bin/ps a tsh> fg %1 tsh> /bin/ps a PID TTY STAT TIME COMMAND root/:0 -nolisten tcp vt7 -novtswtch 1415 tty1 Ss+ 0:00 /sbin/agetty --noclear tty1 linux 5549 pts/4 Ss 0:00 bash 5587 pts/4 T 0:00 make test16 5588 pts/4 T 0:00 /bin/sh -c ./sdriver.pl -t trace16.txt -s ./tsh -a "-p" 5589 pts/4 T 0:00 /usr/bin/perl ./sdriver.pl -t trace16.txt -s ./tsh -a -p 5590 pts/4 Z 0:03 [tsh] <defunct> 5601 pts/4 S 1:21 ./tsh -p 5603 pts/4 T 0:00 ./mystop 2 5881 pts/4 S+ 0:00 make rtest13 5882 pts/4 S+ 0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p" 5883 pts/4 S+ 0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p 5884 pts/4 S+ 0:00 ./tshref -p 5893 pts/4 R 0:00 /bin/ps a</pre>
---	---

测试结论 相同

4.3.14 测试用例 trace14.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5901) ./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 JID [1] (5901) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (5901) ./myspin 4 & tsh> jobs [1] (5901) Running ./myspin 4 & </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5922) ./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] (5922) stopped by signal 20 tsh> bg %2 %2: No such job tsh> bg %1 [1] (5922) ./myspin 4 & tsh> jobs [1] (5922) Running ./myspin 4 & </pre>
测试结论	相同

4.3.15 测试用例 trace15.txt 的输出截图（1 分）

tsh 测试结果	tshref 测试结果
<pre> syq1160300901@syq1160300901-virtual-machine:~/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 JID [1] (5943) terminated by signal 2 tsh> ./myspin 3 & [1] (5943) ./myspin 3 & tsh> ./myspin 4 & [2] (5945) ./myspin 4 & tsh> jobs [1] (5943) Running ./myspin 3 & [2] (5945) Running ./myspin 4 & tsh> fg %1 JID [1] (5943) stopped by signal 20 tsh> jobs [1] (5943) Stopped ./myspin 3 & [2] (5945) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (5943) ./myspin 3 & tsh> jobs [1] (5943) Running ./myspin 3 & [2] (5945) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>	<pre> syq1160300901@syq1160300901-virtual-machine:~/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] (5961) terminated by signal 2 tsh> ./myspin 3 & [1] (5963) ./myspin 3 & tsh> ./myspin 4 & [2] (5965) ./myspin 4 & tsh> jobs [1] (5963) Running ./myspin 3 & [2] (5965) Running ./myspin 4 & tsh> fg %1 Job [1] (5963) stopped by signal 20 tsh> jobs [1] (5963) Stopped ./myspin 3 & [2] (5965) Running ./myspin 4 & tsh> bg %3 %3: No such job tsh> bg %1 [1] (5963) ./myspin 3 & tsh> jobs [1] (5963) Running ./myspin 3 & [2] (5965) Running ./myspin 4 & tsh> fg %1 tsh> quit </pre>
测试结论	相同

4.4 自测试评分

根据节 4.3 的自测试结果，程序的测试评分为：15

第 5 章 总结

5.1 请总结本次实验的收获

1. 这个实验内容是实现一个 unix 下的简易 shell，需要实现的功能包括 bg, fg 等作业控制命令，还需要实现 ctrl-c, ctrl-z 等信息的处理，由于实现这个 shell 需要进行多进程的编程，这是我第一次进行多进程编程，调试的时候很不熟练，遇到很多问题。
2. 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 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。

5.2 请给出对本次实验内容的建议

虽然 Trace16 报告里没让测试，但这个不知道怎么测试，说是不从终端接收信号，要从别的进程接收信号，不知道怎么弄。

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>.