实验四-Shelllab实验

一.准备知识

1. 进程的概念、状态以及控制进程的几个函数（fork,waitpid,execve）。
2. 信号的概念，会编写正确安全的信号处理程序。
3. shell的概念，理解shell程序是如何利用进程管理和信号去执行一个命令行语句。

二.实验目的

shell lab主要目的是为了熟悉进程控制和信号。具体来说需要比对16个test和rtest文件的输出，实现七个函数：

```
void eval(char *cmdline)：分析命令，并派生子进程执行 主要功能是解析cmdline并运行
int builtin_cmd(char **argv)：解析和执行bulidin命令，包括 quit, fg, bg, and jobs
void do_bgfg(char **argv) 执行bg和fg命令
void waitfg(pid_t pid)：实现阻塞等待前台程序运行结束
void sigchld_handler(int sig)：SIGCHID信号处理函数
void sigint_handler(int sig)：信号处理函数，响应 SIGINT (ctrl-c) 信号
void sigtstp_handler(int sig)：信号处理函数，响应 SIGTSTP (ctrl-z) 信号
```

三.实验环境

ubuntu12.04 (32位)环境

四.实验内容及操作步骤

通过阅读实验指导书我们知道此实验要求我们完成tsh.c中的七个函数从而实现一个简单的shell，能够处理前后台运行程序、能够处理ctrl+z、ctrl+c等信号。

首先我们来看一下tsh.c具体内容。

首先定义了一些宏

```
/* 定义了一些宏 */
#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 */
```

定义了四种进程状态

```
/* 工作状态 */
#define UNDEF 0 /* undefined */
#define FG 1/* 前台状态 */
#define BG 2/* 后台状态 */
#define ST 3/* 挂起状态 */
```

然后定义了job_t的任务的类，并且创建了jobs[]数组

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

接着是需要我们完成的七个函数定义

```
void eval(char *cmdline);//分析命令，并派生子进程执行 主要功能是解析cmdline并运行
int builtin_cmd(char **argv);//解析和执行bulidin命令，包括 quit, fg, bg, and jobs
void do_bgfg(char **argv);//执行bg和fg命令
void waitfg(pid_t pid);//实现阻塞等待前台程序运行结束

void sigchld_handler(int sig);//SIGCHID信号处理函数
void sigtstp_handler(int sig);//信号处理函数，响应 SIGINT (ctrl-c) 信号
void sigint_handler(int sig);//信号处理函数，响应 SIGTSTP (ctrl-z) 信号
```

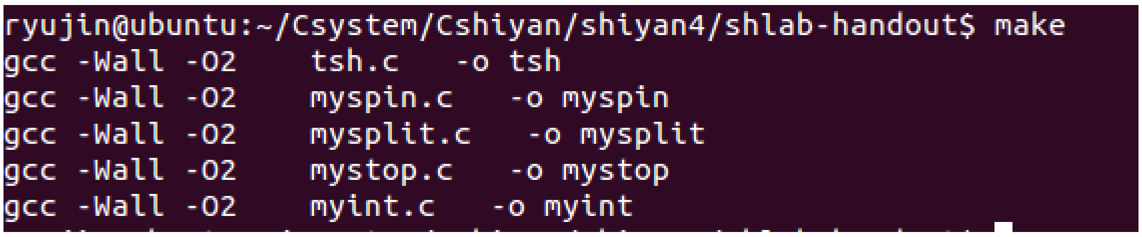
一些辅助的函数

```
int parseline(const char *cmdline, char **argv);   //获取参数列表，返回是否为后台运行命令
void sigquit_handler(int sig);  //处理SIGQUIT信号
void clearjob(struct job_t *job);  //清除job结构体
void initjobs(struct job_t *jobs);  //初始化任务jobs[]
int maxjid(struct job_t *jobs);   //返回jobs链表中最大的jid号。
int addjob(struct job_t *jobs, pid_t pid, int state, char *cmdline);  //向jobs[]添加一个任务
int deletejob(struct job_t *jobs, pid_t pid);   //在jobs[]中删除pid的job
pid_t fgpid(struct job_t *jobs);  //返回当前前台运行job的pid号
struct job_t *getjobpid(struct job_t *jobs, pid_t pid);  //根据pid找到对应的job
struct job_t *getjobjid(struct job_t *jobs, int jid);   //根据jid找到对应的job
int pid2jid(pid_t pid);   //根据pid找到jid
void listjobs(struct job_t *jobs);  //打印jobs
```

接着就是mian函数，作用是在文件中逐行获取命令，并且判断是不是文件结束（EOF），将命令cmdline送入eval函数进行解析。我们需要做的就是逐步完善这个过程

接下来开始实验：

使用make命令编译tsh.c文件（文件有所改变的话需要先使用make clean指令清空）



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make
gcc -Wall -O2    tsh.c   -o tsh
gcc -Wall -O2    myspin.c   -o myspin
gcc -Wall -O2    mysplit.c   -o mysplit
gcc -Wall -O2    mystop.c   -o mystop
gcc -Wall -O2    myint.c   -o myint
```

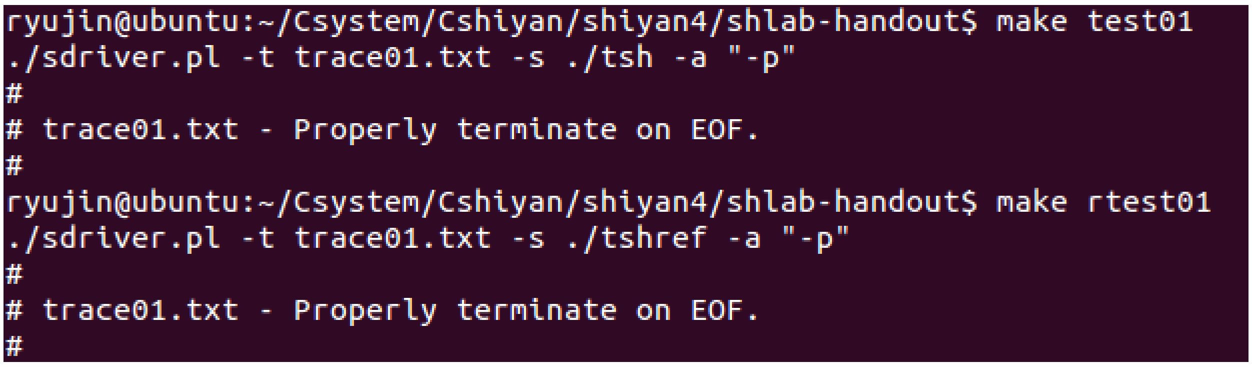
使用make testXX指令比较traceXX.txt文件在编写的shell和reference shell的运行结果；或者也可以使用"./sdriver.pl -t traceXX.txt -s ./tsh -a "-p"

如果在文件名前面加上r，则是执行标准的tshref，或者将tsh变为tshref。通过比对标准tshref和自制tsh的执行结果结果，可以观察tsh的功能是否正确。如果tsh的执行结果和tshref结果一致，说明结果是正确的

五.具体实现

1.trace01

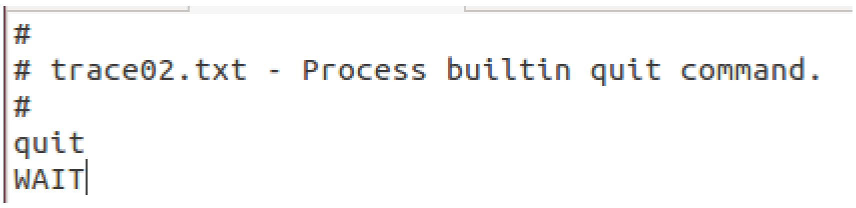
正确终止EOF



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

2.trace02

实现内置的quit



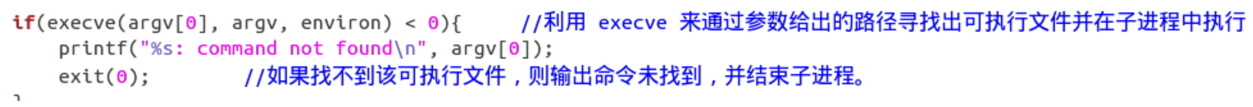
```
#
# trace02.txt - Process builtin quit command.
#
quit
WAIT
```

需编写tsh的内置命令quit

这里需要理解eval（）函数

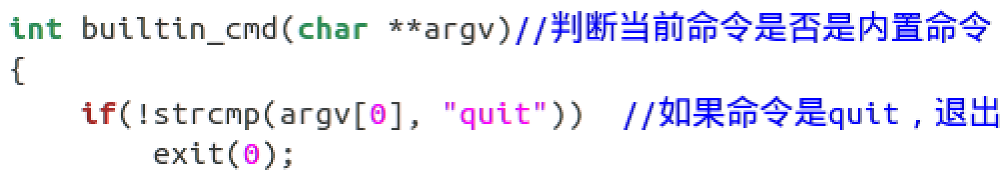
程序会首先执行 eval()，在 eval中进行判断（使用buildin_cmp()函数），如果发现命令不是内置命令，则会调用 fork()函数来新建一个子进程，在子进程中调用 execve()函数通过 argv[0]来寻找路径，并在子进程中运行路径中的可执行文件，如果找不到可执行文件，则说明命令为无效命令，输出命令无效，并用 exit(0)结束该子进程即可。

具体实现：



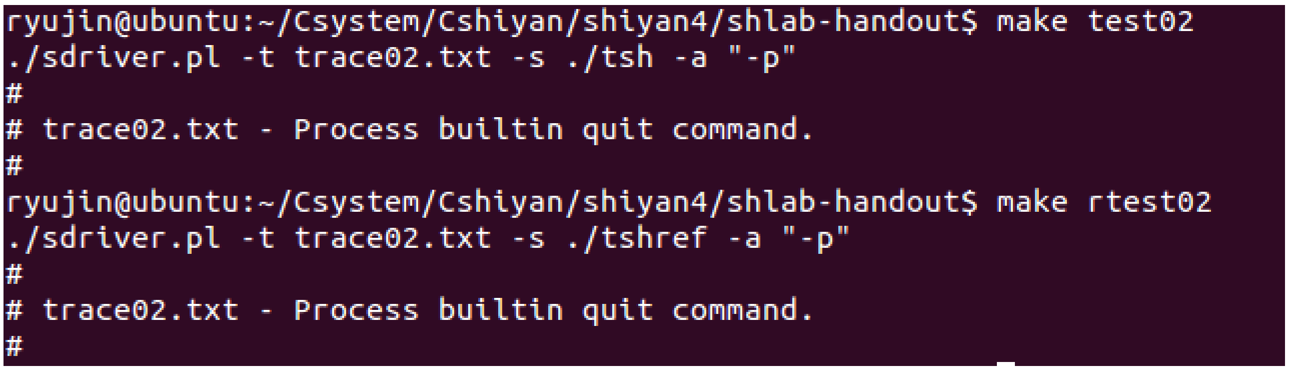
```
if(execve(argv[0], argv, environ) < 0){     //利用 execve 来通过参数给出的路径寻找出可执行文件并在子进程中执行
    printf("%s: command not found\n", argv[0]);
    exit(0);        //如果找不到该可执行文件，则输出命令未找到，并结束子进程。
```

判断是否为内置命令的函数builtin_cmd()



```
int builtin_cmd(char **argv)//判断当前命令是否是内置命令
{
    if(!strcmp(argv[0], "quit"))  //如果命令是quit，退出
        exit(0);
```

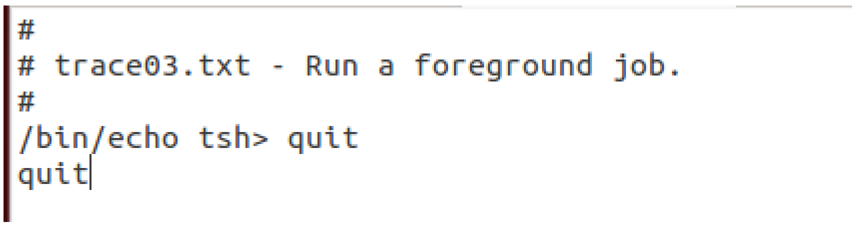
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

3.trace03

运行一个前台job

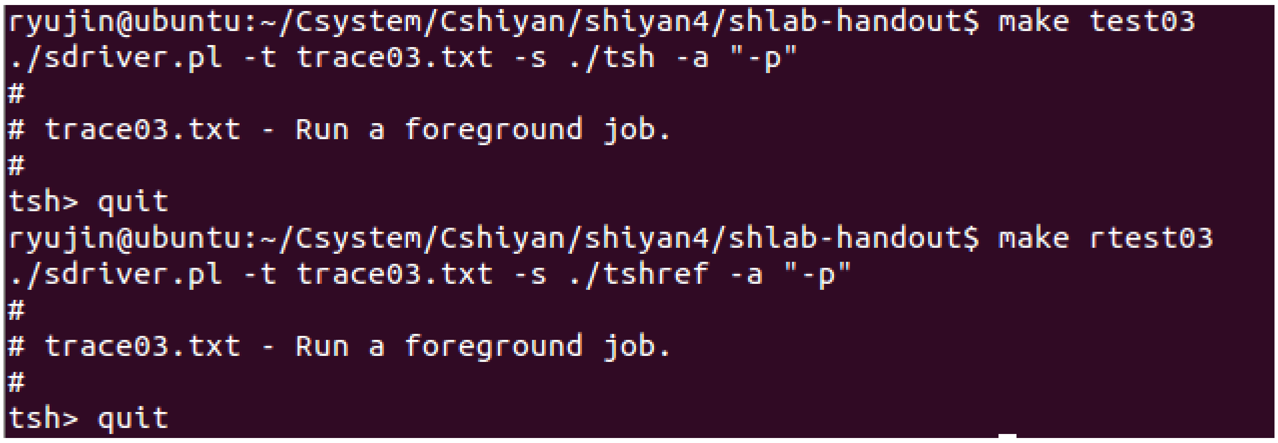


```
#
# trace03.txt - Run a foreground job.
#
/bin/echo tsh> quit
quit
```

/bin/echo就是打开bin目录下的echo文件，echo可以理解为将其后面的内容当作字符串输出

- 首先是/bin/echo tsh> quit 意思是打开 bin 目录下的 echo 可执行文件，在 foregound 开启一个子进程运行它（因为末尾没有&符号，如果有，就是在 backgound 运行）
- 运行 echo 这个进程的过程中，通过 tsh>quit 命令，调用 tsh并执行内置命令 quit，退出 echo 这个子进程
- 最后在 tsh 中执行内置命令 quit，退出 tsh 进程，回到我们的终端。

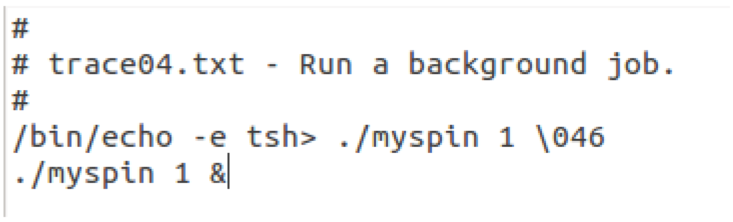
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

4.trace04

运行后台job



```
#
# trace04.txt - Run a background job.
#
/bin/echo -e tsh> ./myspin 1 \046
./myspin 1 &
```

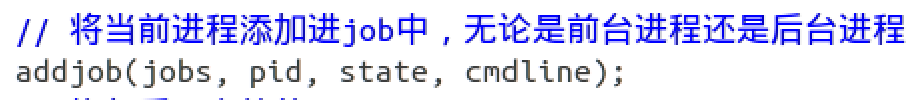
先在前台执行echo命令，等待程序执行完毕回收子进程。&代表是一个后台程序，myspin睡眠1秒，然后停止。因为在后台，所以显示下面一句，如果在前台则无。

解答：

在原有的eval函数基础之上添加将作业添加至后台作业管理的函数使用（addjobs（）），信号的阻塞和取消阻塞。

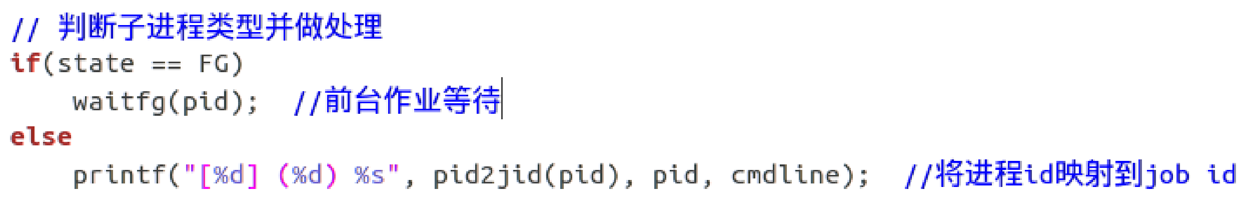
在fork()新进程前要阻塞SIGCHLD信号，防止出现竞争，这是经典的同步错误，如果不阻塞会出现子进程先结束从jobs中删除，然后再执行到主进程addjob的竞争问题。

具体实现：



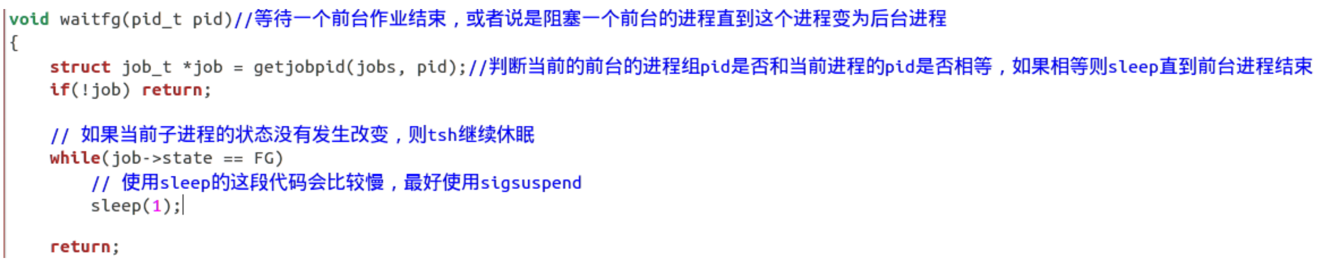
```
// 将当前进程添加进job中，无论是前台进程还是后台进程
addjob(jobs, pid, state, cmdline);
```

在eval中进行判断是否为后台进程：



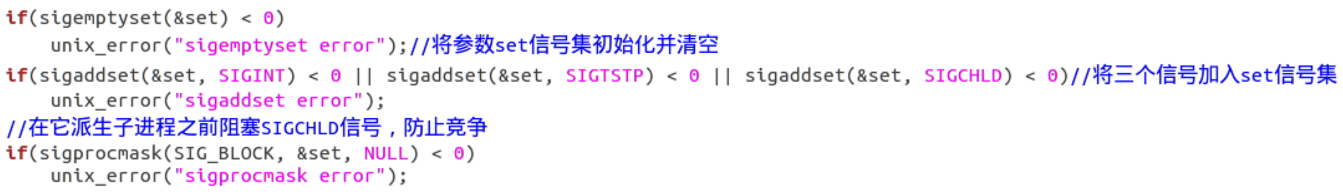
```
// 判断子进程类型并做处理
if(state == FG)
    waitfg(pid);  //前台作业等待
else
    printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);  //将进程id映射到job id
```

等待一个前台作业结束

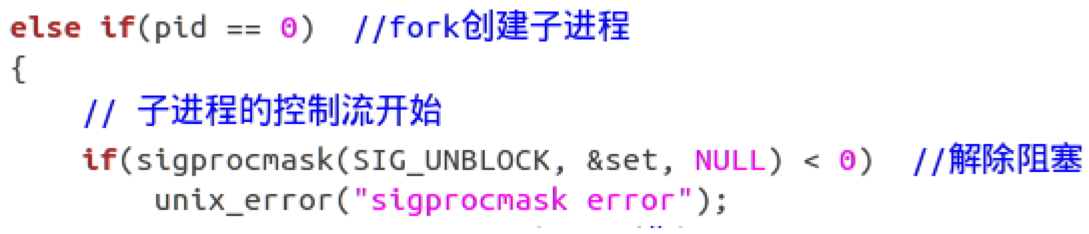


```
void waitfg(pid_t pid)//等待一个前台作业结束，或者说是阻塞一个前台的进程直到这个进程变为后台进程
{
    struct job_t *job = getjobpid(jobs, pid);//判断当前的前台的进程组pid是否和当前进程的pid是否相等，如果相等则sleep直到前台进程结束
    if(!job) return;

    // 如果当前子进程的状态没有发生改变，则tsh继续休眠
    while(job->state == FG)
        // 使用sleep的这段代码会比较慢，最好使用sigsuspend
        sleep(1);

    return;
```

使用sigprocmask()函数显式地阻塞和取消阻塞:

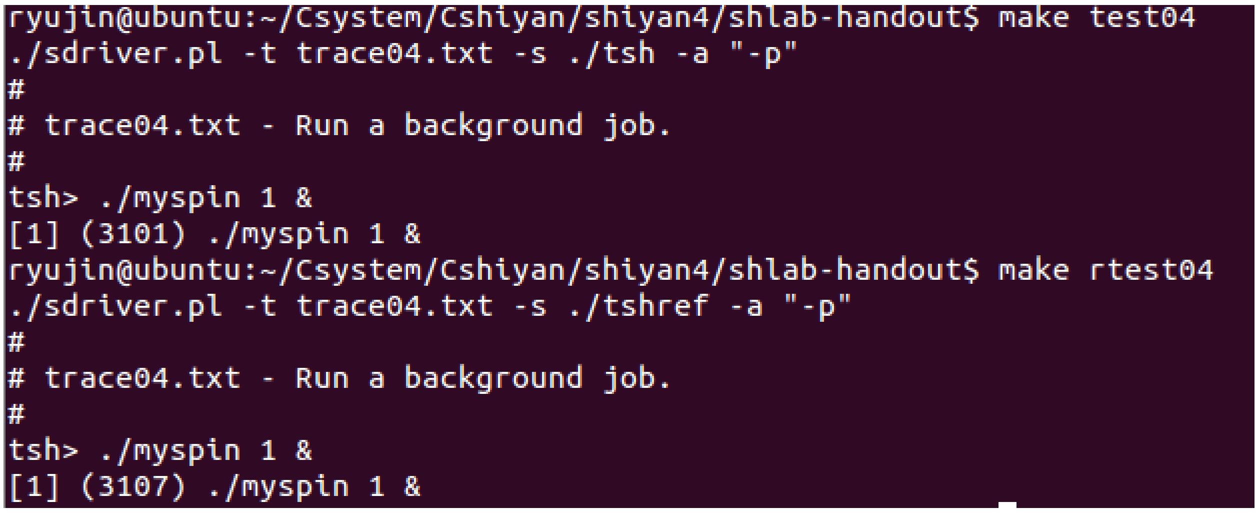


```
if(sigemptyset(&set) < 0)
    unix_error("sigemptyset error");//将参数set信号集初始化并清空
if(sigaddset(&set, SIGINT) < 0 || sigaddset(&set, SIGTSTP) < 0 || sigaddset(&set, SIGCHLD) < 0)//将三个信号加入set信号集
    unix_error("sigaddset error");
//在它派生子进程之前阻塞SIGCHLD信号，防止竞争
if(sigprocmask(SIG_BLOCK, &set, NULL) < 0)
    unix_error("sigprocmask error");
```



```
else if(pid == 0)  //fork创建子进程
{
    // 子进程的控制流开始
    if(sigprocmask(SIG_UNBLOCK, &set, NULL) < 0)  //解除阻塞
        unix_error("sigprocmask error");
```

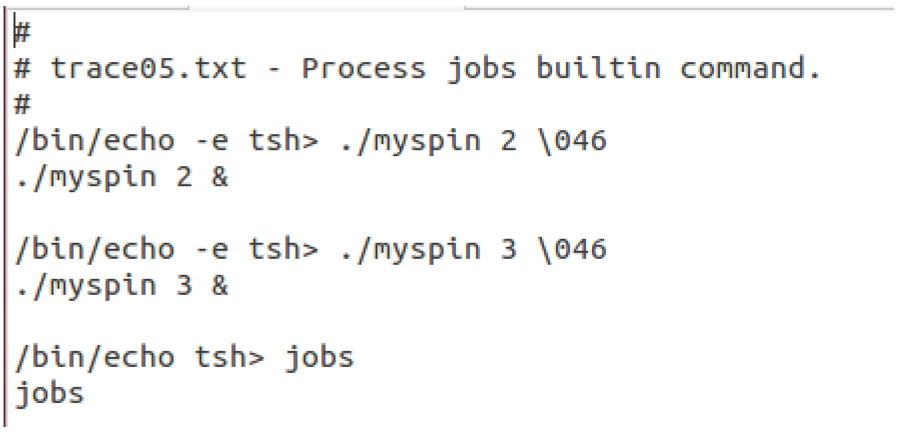
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (3101) ./myspin 1 &
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (3107) ./myspin 1 &
```

5.trace05

处理jobs内置命令



```
#
# trace05.txt - Process jobs builtin command.
#
/bin/echo -e tsh> ./myspin 2 \046
./myspin 2 &

/bin/echo -e tsh> ./myspin 3 \046
./myspin 3 &

/bin/echo tsh> jobs
jobs
```

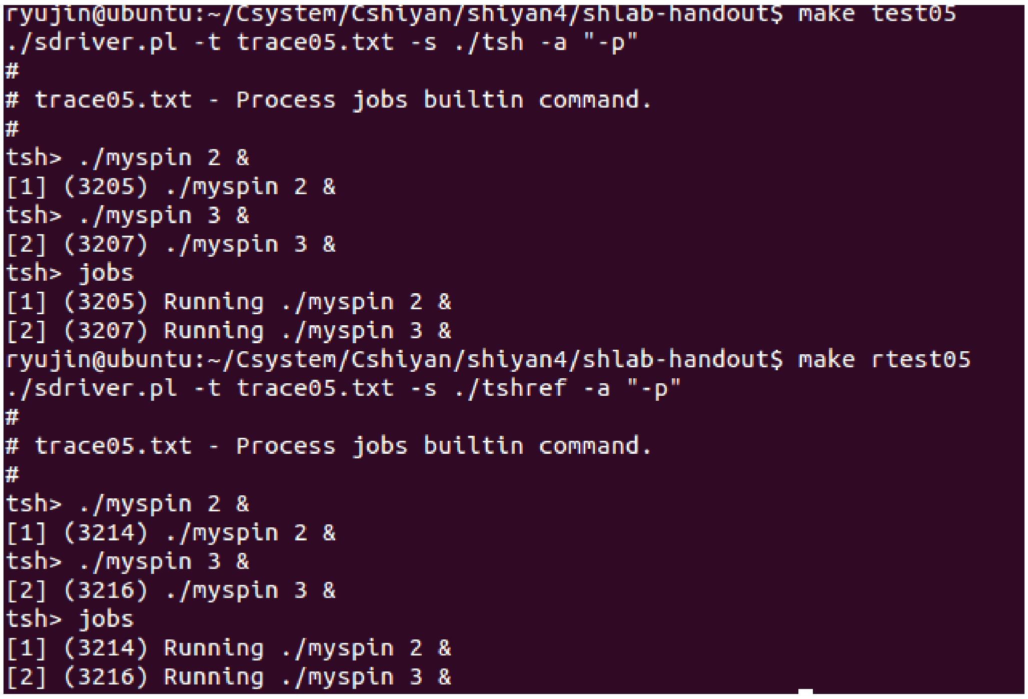
分别运行了前台echo、后台myspin、前台echo、后台myspin，然后需要实现一个内置命令job，功能是显示目前任务列表中的所有任务的所有属性

解答：

直接调用自带的listjobs()方法，就是在原有builtin_cmd函数中添加一个判断函数，如果参数是jobs，则执行listjobs函数的功能（即将所有的作业打印出来）

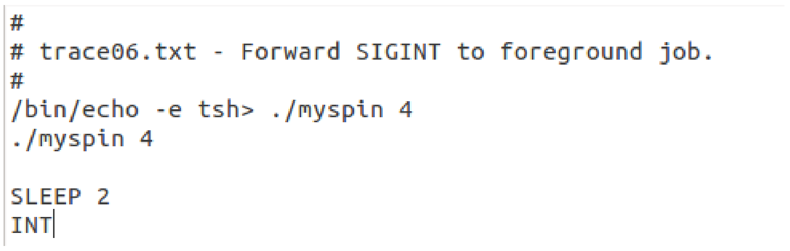
```
else if(!strcmp(argv[0], "jobs"))  //如果命令是jobs，列出正在运行和停止的后台作业
    listjobs(jobs);
```

test

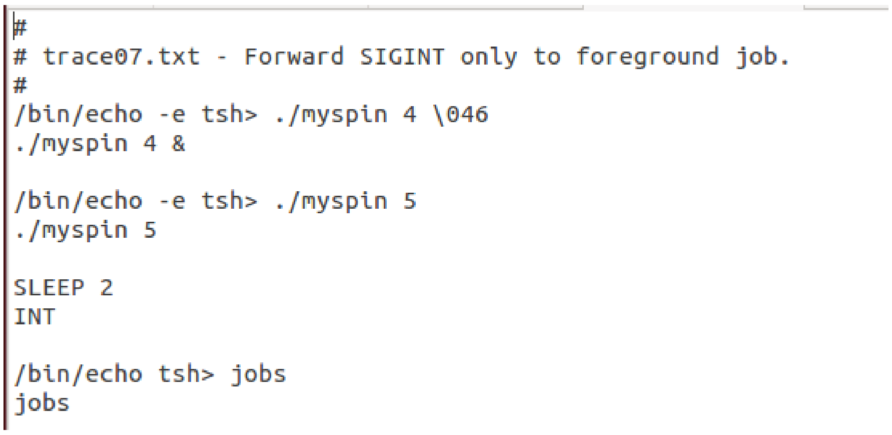


```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (3205) ./myspin 2 &
tsh> ./myspin 3 &
[2] (3207) ./myspin 3 &
tsh> jobs
[1] (3205) Running ./myspin 2 &
[2] (3207) Running ./myspin 3 &
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (3214) ./myspin 2 &
tsh> ./myspin 3 &
[2] (3216) ./myspin 3 &
tsh> jobs
[1] (3214) Running ./myspin 2 &
[2] (3216) Running ./myspin 3 &
```

6.trace06 和 trace07



```
#
# trace06.txt - Forward SIGINT to foreground job.
#
/bin/echo -e tsh> ./myspin 4
./myspin 4

SLEEP 2
INT
```



```
#
# trace07.txt - Forward SIGINT only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
INT

/bin/echo tsh> jobs
jobs
```

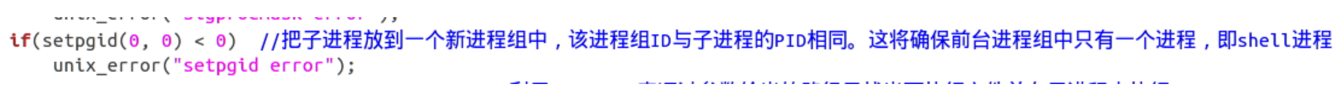
6：将SIGINT转发到前台作业

7：仅仅将SIGINT信号转发到前台作业

接收到了中断信号SIGINT（即CTRL_C)那么结束前台进程

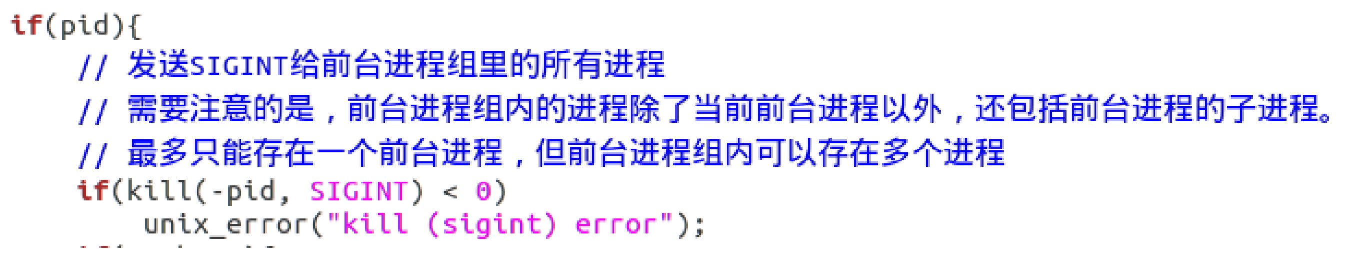
解答：

在eval函数中调用setpgid（0,0），保证ctrl+c只会终止当前的shell进程，而不会影响其他进程



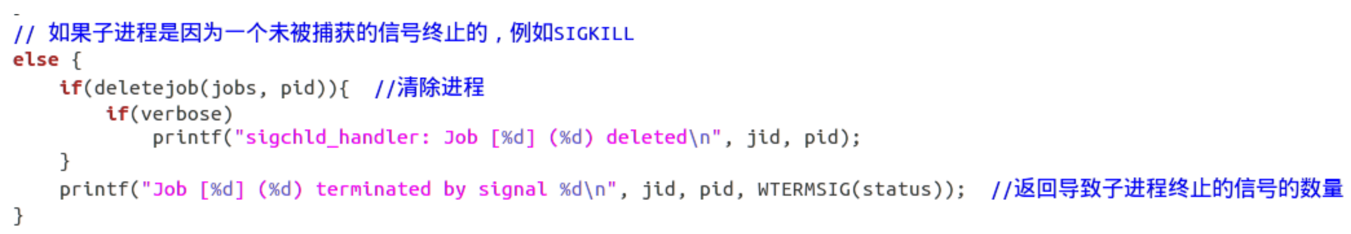
```
if(setpgid(0, 0) < 0)  //把子进程放到一个新进程组中，该进程组ID与子进程的PID相同。这将确保前台进程组中只有一个进程，即shell进程
    unix_error("setpgid error");
```

sigint_handler()实现转发到前台作业的操作



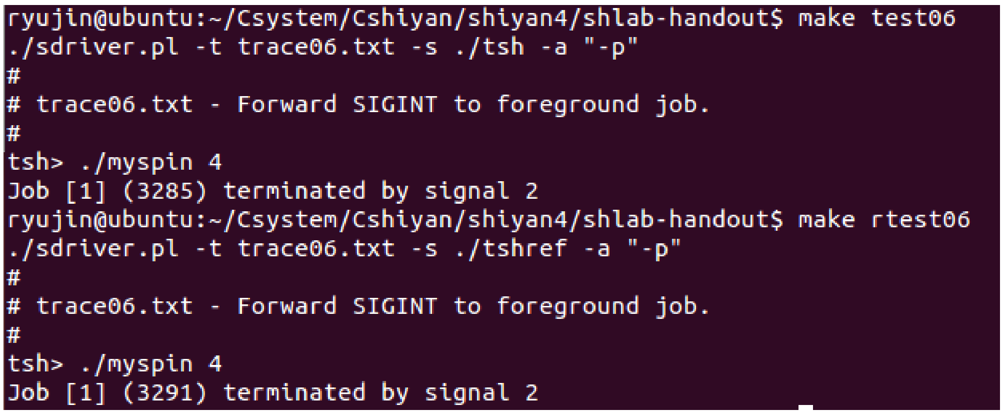
```
if(pid){
    // 发送SIGINT给前台进程组里的所有进程
    // 需要注意的是，前台进程组内的进程除了当前前台进程以外，还包括前台进程的子进程。
    // 最多只能存在一个前台进程，但前台进程组内可以存在多个进程
    if(kill(-pid, SIGINT) < 0)
        unix_error("kill (sigint) error");
```

sigchld_handler()函数，因为收到其他信号如：SIGINT而终止

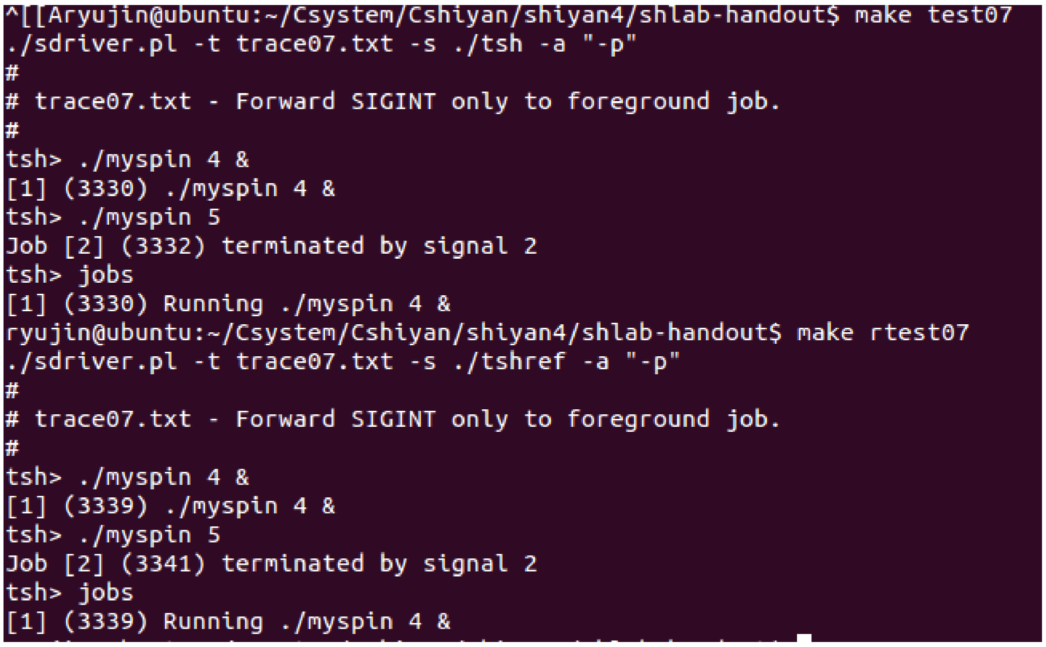


```
// 如果子进程是因为一个未被捕获的信号终止的，例如SIGKILL
else {
    if(deletejob(jobs, pid)){  //清除进程
        if(verbose)
            printf("sigchld_handler: Job [%d] (%d) deleted\n", jid, pid);
    }
    printf("Job [%d] (%d) terminated by signal %d\n", jid, pid, WTERMSIG(status));  //返回导致子进程终止的信号的数量
}
```

test



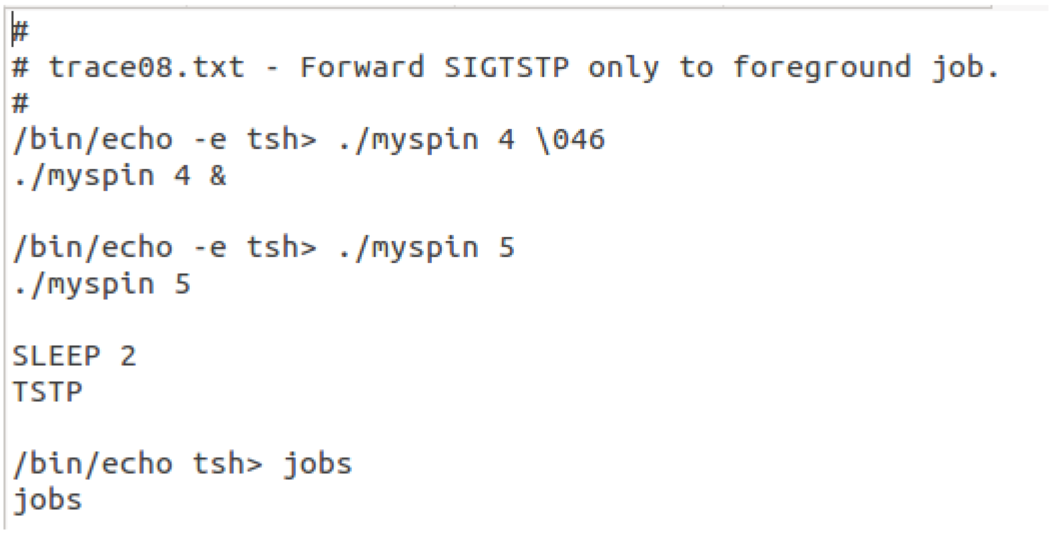
```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (3285) terminated by signal 2
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (3291) terminated by signal 2
```



```
^[[Aryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (3330) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3332) terminated by signal 2
tsh> jobs
[1] (3330) Running ./myspin 4 &
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (3339) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3341) terminated by signal 2
tsh> jobs
[1] (3339) Running ./myspin 4 &
```

7.trace08

仅将SIGTSTP转发到前台作业

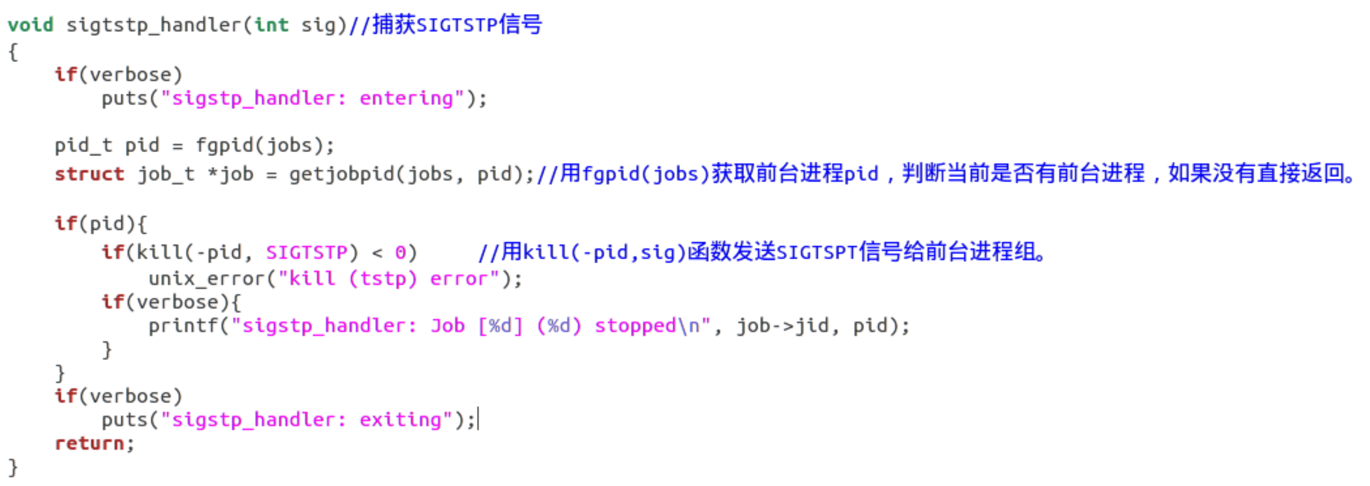


```
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs
```

需要将SIGTSTP转发给前台作业。根据这个信号的作用，也就是该进程会停止直到下一个SIGCONT也就是挂起，让别的程序继续运行。这里也就是运行了后台程序，然后使用jobs来打印出进程的信息。

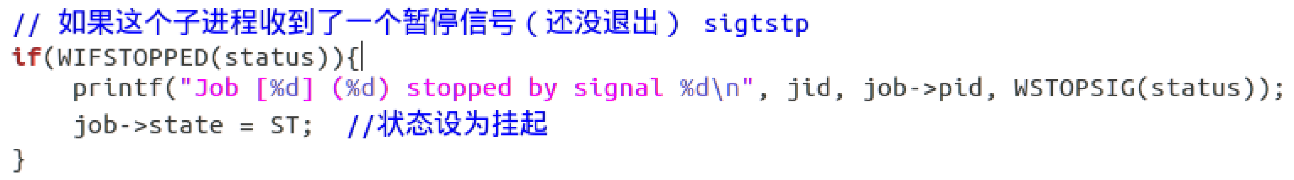
解答：

实现其信号处理函数sigtstp_handler()



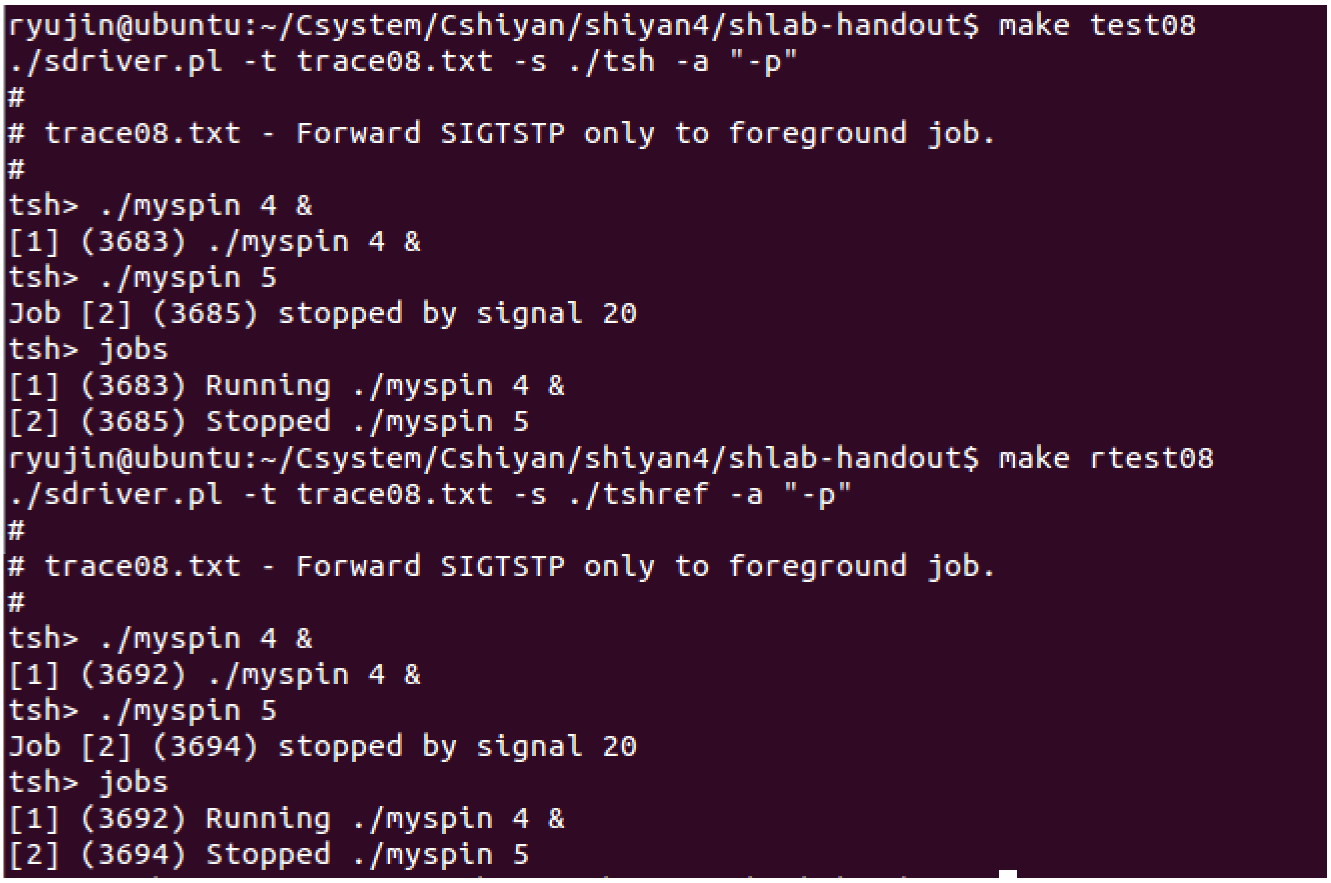
```
void sigtstp_handler(int sig)//捕获SIGTSTP信号
{
    if(verbose)
        puts("sigstp_handler: entering");

    pid_t pid = fgpid(jobs);
    struct job_t *job = getjobpid(jobs, pid);//用fgpid(jobs)获取前台进程pid，判断当前是否有前台进程，如果没有直接返回。

    if(pid){
        if(kill(-pid, SIGTSTP) < 0)     //用kill(-pid,sig)函数发送SIGTSPT信号给前台进程组。
            unix_error("kill (tstp) error");
        if(verbose){
            printf("sigstp_handler: Job [%d] (%d) stopped\n", job->jid, pid);
        }
    }
    if(verbose)
        puts("sigstp_handler: exiting");
    return;
}
```

sigchld_handler()函数增加对于SIGTSTP（ctrl+z）的判断和信息显示



```
// 如果这个子进程收到了一个暂停信号（还没退出） sigtstp
if(WIFSTOPPED(status)){
    printf("Job [%d] (%d) stopped by signal %d\n", jid, job->pid, WSTOPSIG(status));
    job->state = ST;  //状态设为挂起
}
```

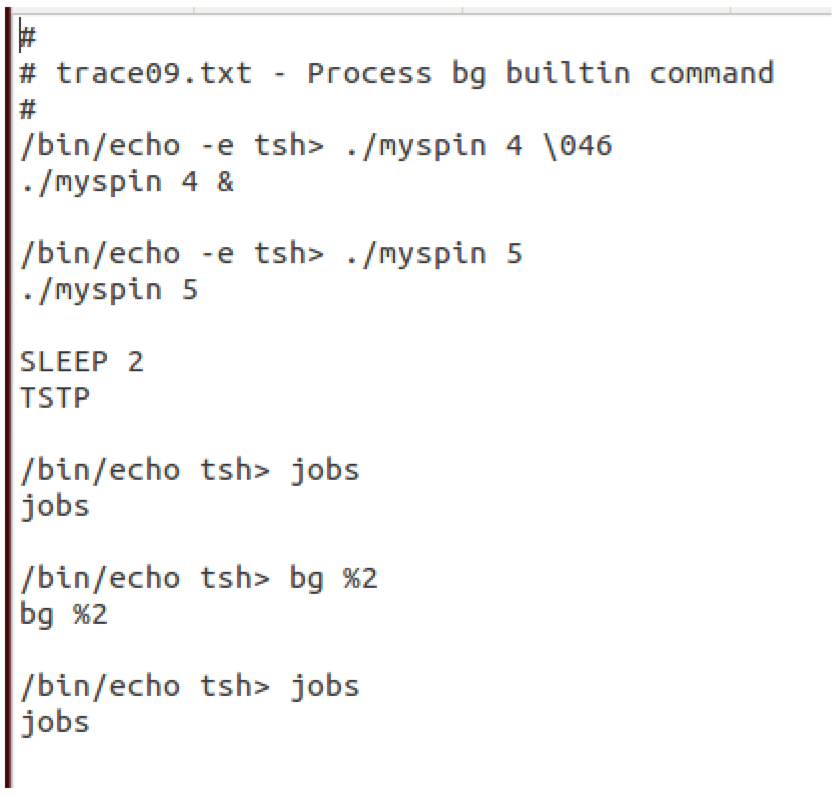
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (3683) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3685) stopped by signal 20
tsh> jobs
[1] (3683) Running ./myspin 4 &
[2] (3685) Stopped ./myspin 5
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (3692) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3694) stopped by signal 20
tsh> jobs
[1] (3692) Running ./myspin 4 &
[2] (3694) Stopped ./myspin 5
```

8.trace09

进程bg内置命令



```
#
# trace09.txt - Process bg builtin command
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo -e tsh> ./myspin 5
./myspin 5

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> bg %2
bg %2

/bin/echo tsh> jobs
jobs
```

在第八关的测试文件之上的一个更加完整的测试，这里也就是在停止后，输出进程信息之后，使用bg命令来唤醒进程2，也就是刚才被挂起的程序，接下来继续使用Jobs命令来输出结果。

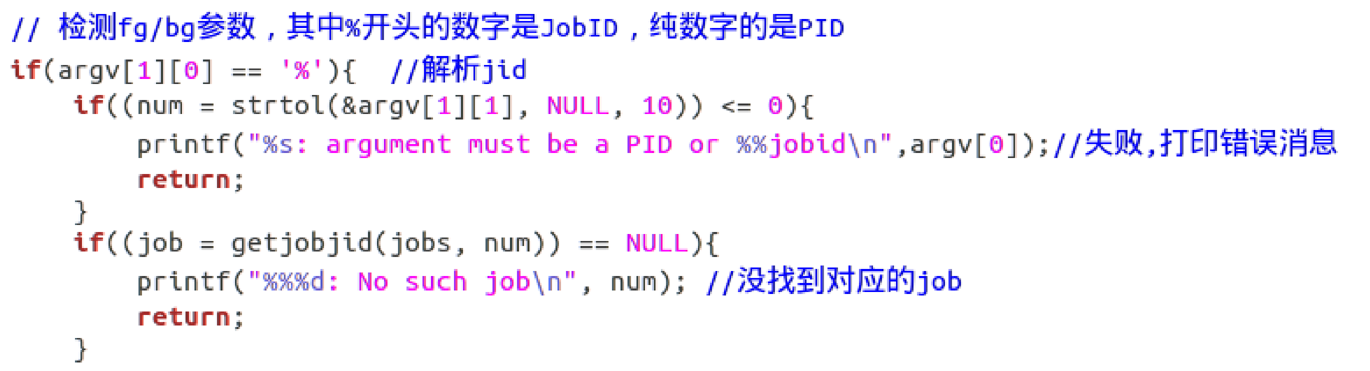
bg <job>:将停止的后台作业更改为正在运行的后台作业。通过发送SIGCONT信号重新启动<job>，然后在后台运行它。<job>参数可以是PID，也可以是JID。ST -> BG

解答：

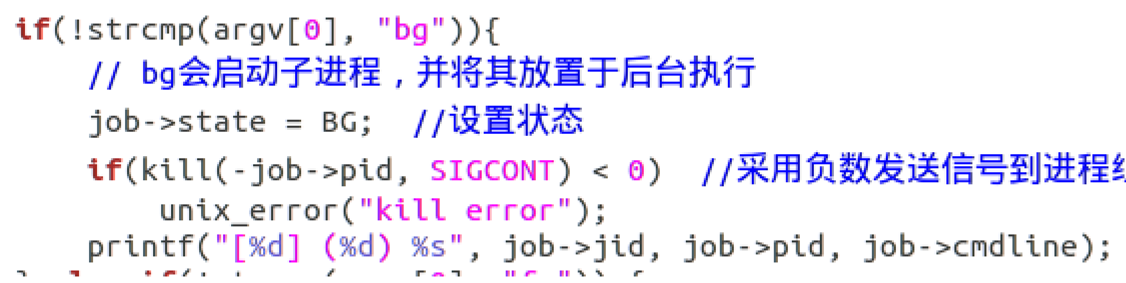
将bg命令添加到识别命令的函数builtin_cmd()中：

```
else if(!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg"))  //如果是bg或者fg命令，执行do_fgbg函数
    do_bgfg(argv);
```

实现其处理函数do_bgfg()

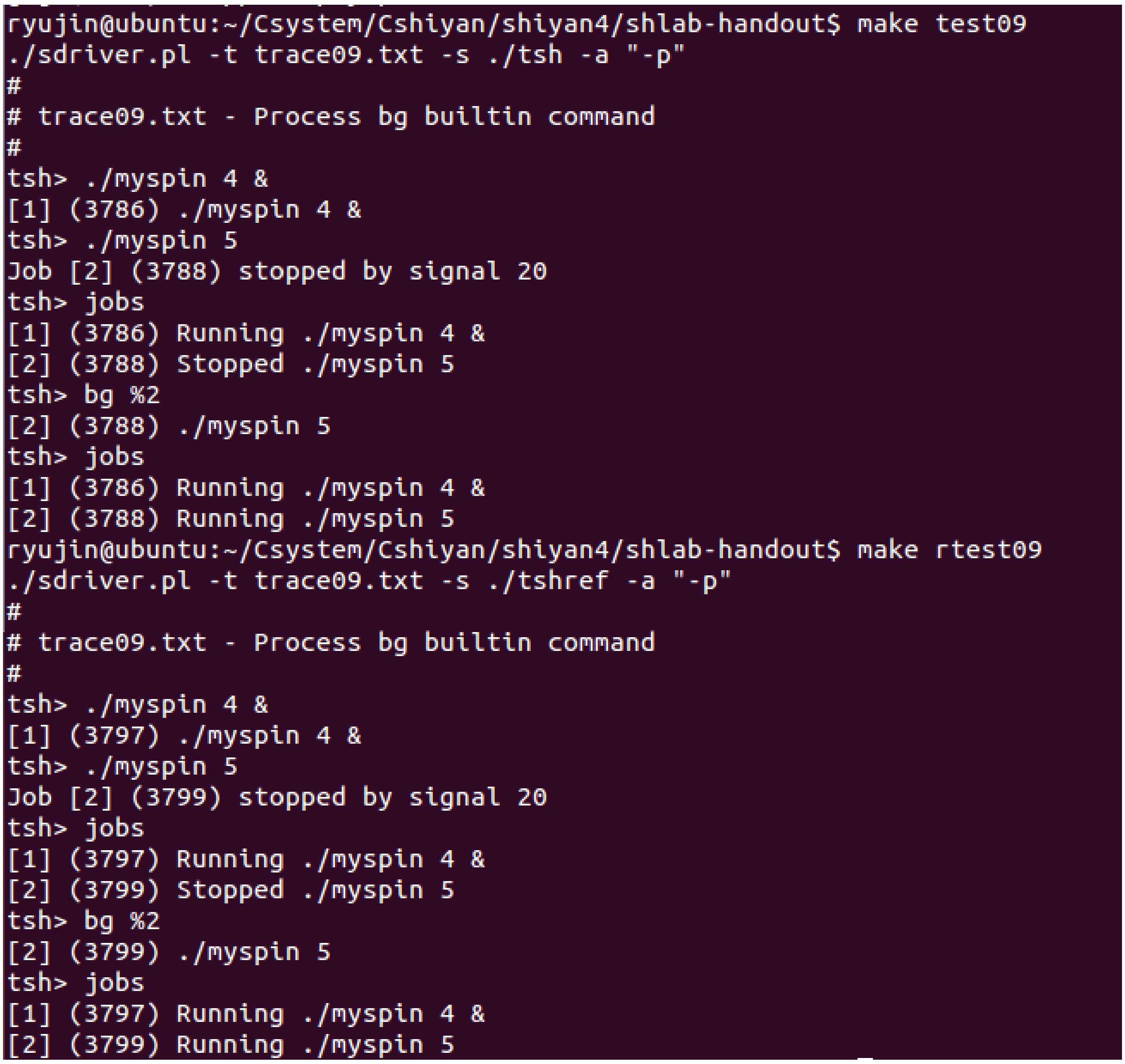


```
// 检测fg/bg参数，其中%开头的数字是JobID，纯数字的是PID
if(argv[1][0] == '%'){  //解析jid
    if((num = strtol(&argv[1][1], NULL, 10)) <= 0){
        printf("%s: argument must be a PID or %%jobid\n",argv[0]);//失败,打印错误消息
        return;
    }
    if((job = getjobjid(jobs, num)) == NULL){
        printf("%%%d: No such job\n", num); //没找到对应的job
        return;
    }
```



```
if(!strcmp(argv[0], "bg")){
    // bg会启动子进程，并将其放置于后台执行
    job->state = BG;  //设置状态
    if(kill(-job->pid, SIGCONT) < 0)  //采用负数发送信号到进程组
        unix_error("kill error");
    printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
```

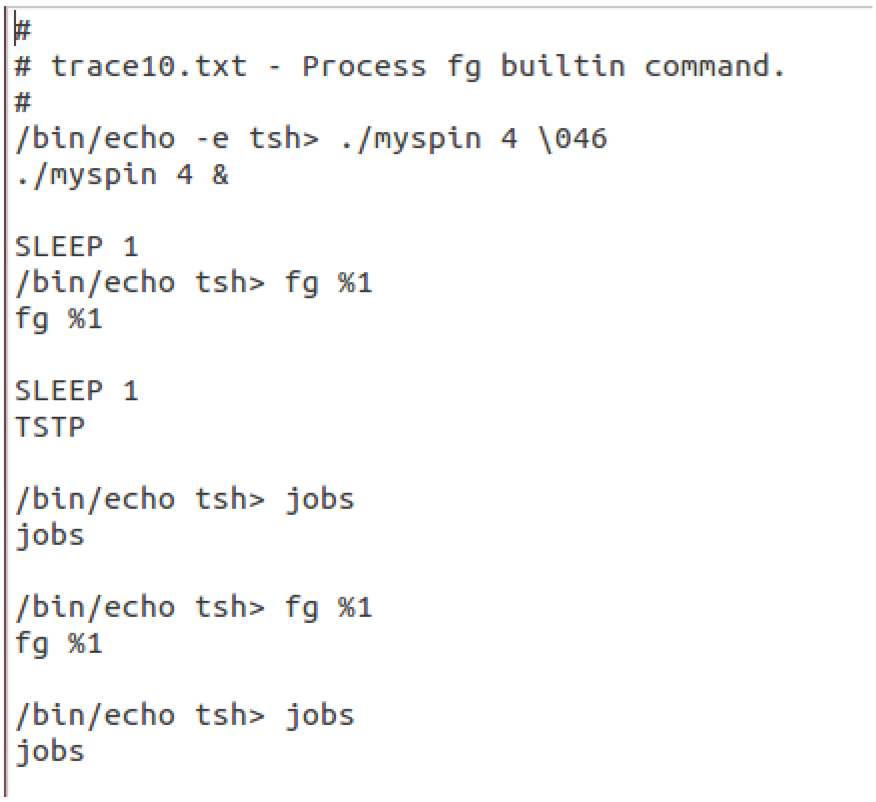
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (3786) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3788) stopped by signal 20
tsh> jobs
[1] (3786) Running ./myspin 4 &
[2] (3788) Stopped ./myspin 5
tsh> bg %2
[2] (3788) ./myspin 5
tsh> jobs
[1] (3786) Running ./myspin 4 &
[2] (3788) Running ./myspin 5
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (3797) ./myspin 4 &
tsh> ./myspin 5
Job [2] (3799) stopped by signal 20
tsh> jobs
[1] (3797) Running ./myspin 4 &
[2] (3799) Stopped ./myspin 5
tsh> bg %2
[2] (3799) ./myspin 5
tsh> jobs
[1] (3797) Running ./myspin 4 &
[2] (3799) Running ./myspin 5
```

9.trace10

进程fg内置命令



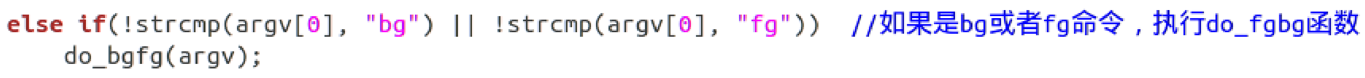
```
#
# trace10.txt - Process fg builtin command.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

SLEEP 1
/bin/echo tsh> fg %1
fg %1

SLEEP 1
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> fg %1
fg %1

/bin/echo tsh> jobs
jobs
```

将后台的进程更改为前台正在运行的程序。测试文中进程1根据&可以知道，进程1是一个后台进程。先使用fg命令将其转化为前台的一个程序，接下来停止进程1，然后打印出进程信息，这时候进程1应该是前台程序同时被挂起了，接下来使用fg命令使其继续运行，使用jobs来打印出进程信息

fg <job>:将一个已停止或正在运行的后台作业更改为正在前台运行的作业。

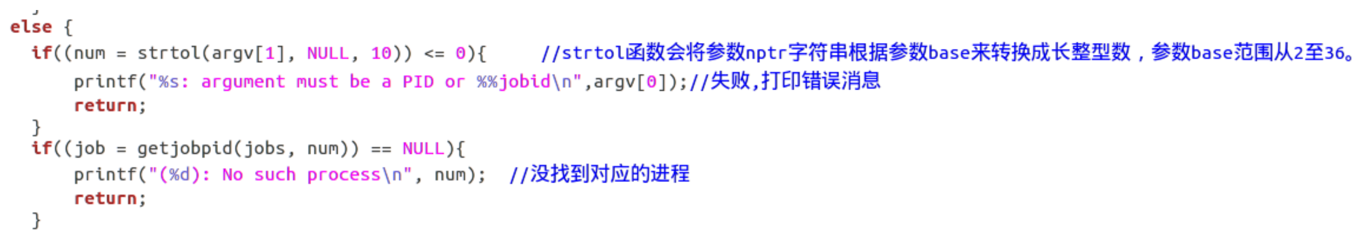
解答：

builtin_cmd()

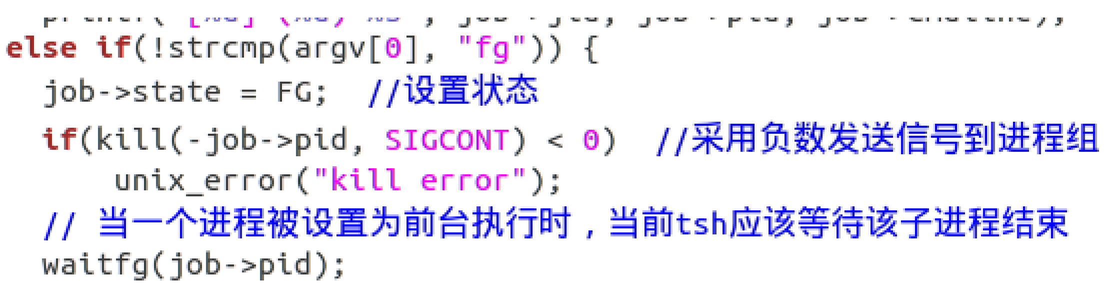


```
else if(!strcmp(argv[0], "bg") || !strcmp(argv[0], "fg"))  //如果是bg或者fg命令，执行do_fgbg函数
    do_bgfg(argv);
```

do_bgfg()函数中加入相关处理

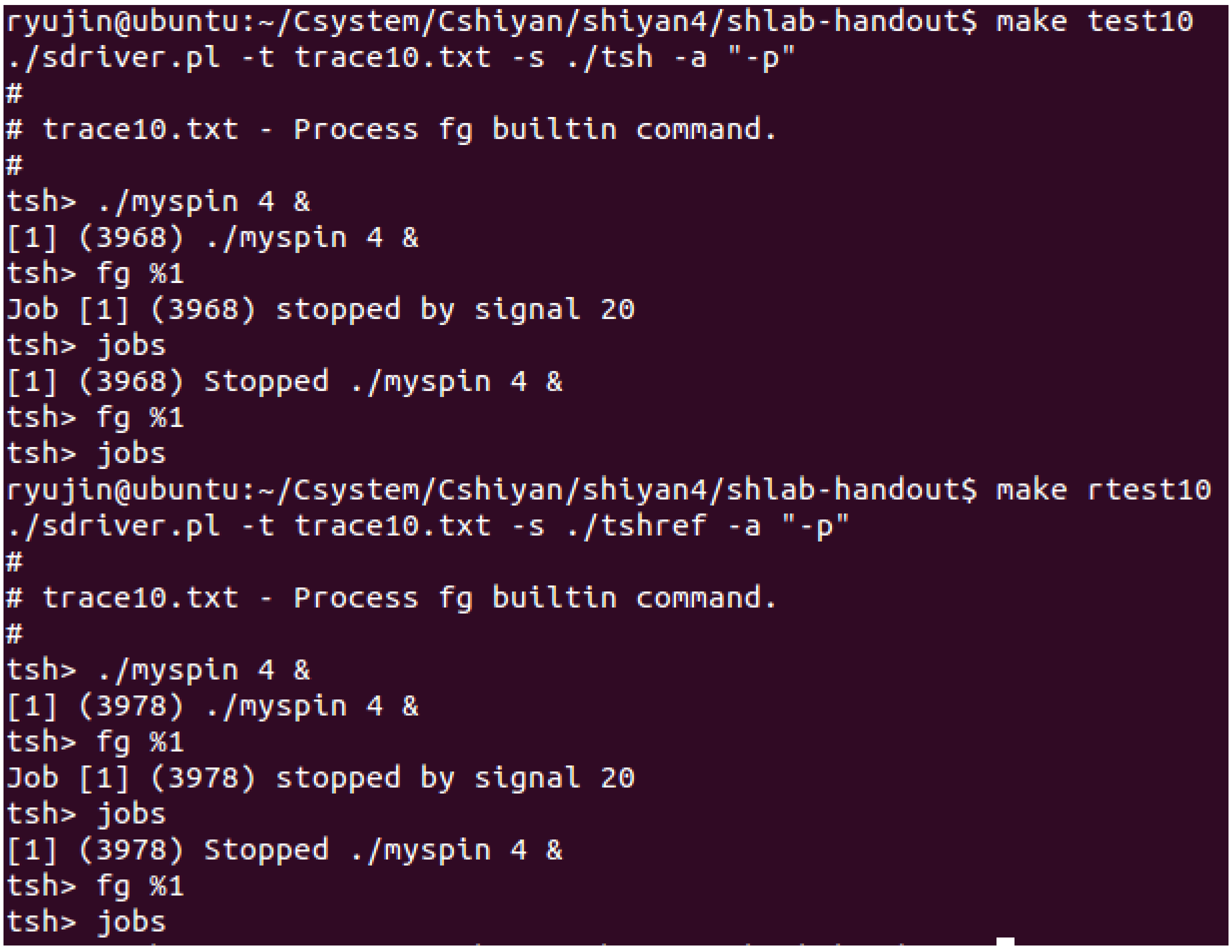


```
else {
  if((num = strtol(argv[1], NULL, 10)) <= 0){     //strtol函数会将参数nptr字符串根据参数base来转换成长整型数，参数base范围从2至36。
      printf("%s: argument must be a PID or %%jobid\n",argv[0]);//失败,打印错误消息
      return;
  }
  if((job = getjobpid(jobs, num)) == NULL){
      printf("(%d): No such process\n", num);  //没找到对应的进程
      return;
  }
```



```
else if(!strcmp(argv[0], "fg")) {
  job->state = FG;  //设置状态
  if(kill(-job->pid, SIGCONT) < 0)  //采用负数发送信号到进程组
      unix_error("kill error");
  // 当一个进程被设置为前台执行时，当前tsh应该等待该子进程结束
  waitfg(job->pid);
```

test

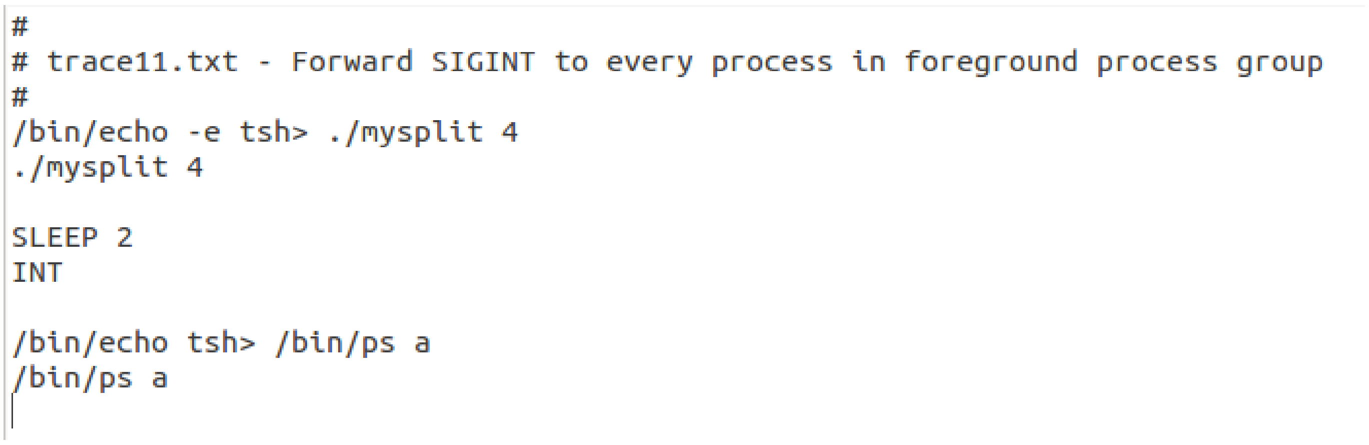


```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (3968) ./myspin 4 &
tsh> fg %1
Job [1] (3968) stopped by signal 20
tsh> jobs
[1] (3968) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (3978) ./myspin 4 &
tsh> fg %1
Job [1] (3978) stopped by signal 20
tsh> jobs
[1] (3978) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

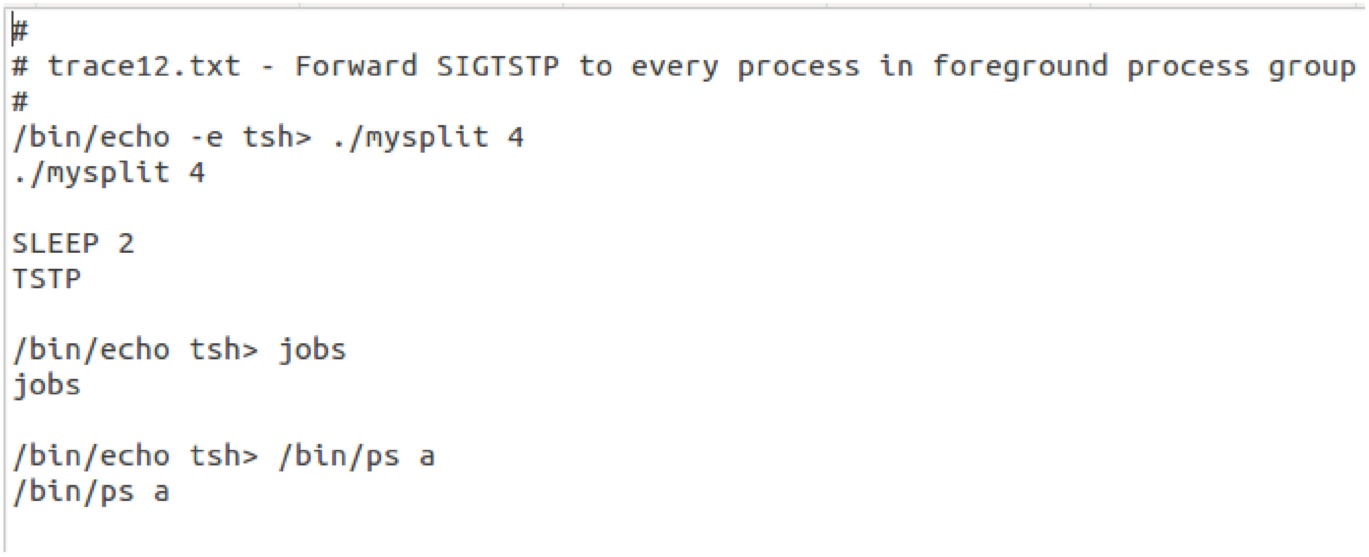
10.trace11 和 trace12

将SIGINT转发给前台进程组中的每个进程

将SIGTSTP转发给前台进程组中的每个进程



```
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
INT

/bin/echo tsh> /bin/ps a
/bin/ps a
```



```
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> /bin/ps a
/bin/ps a
```

将SIGINT/SIGTSTP发给前台进程组中的每个进程。ps –a 显示所有进程，这里是有两个进程的，mysplit创建了一个子进程，接下来发送指令SIGINT/SIGTSTP，所以进程组中的所有进程都应该停止，接下来调用pl来查看该进程组中的每个进程是否都停止了。

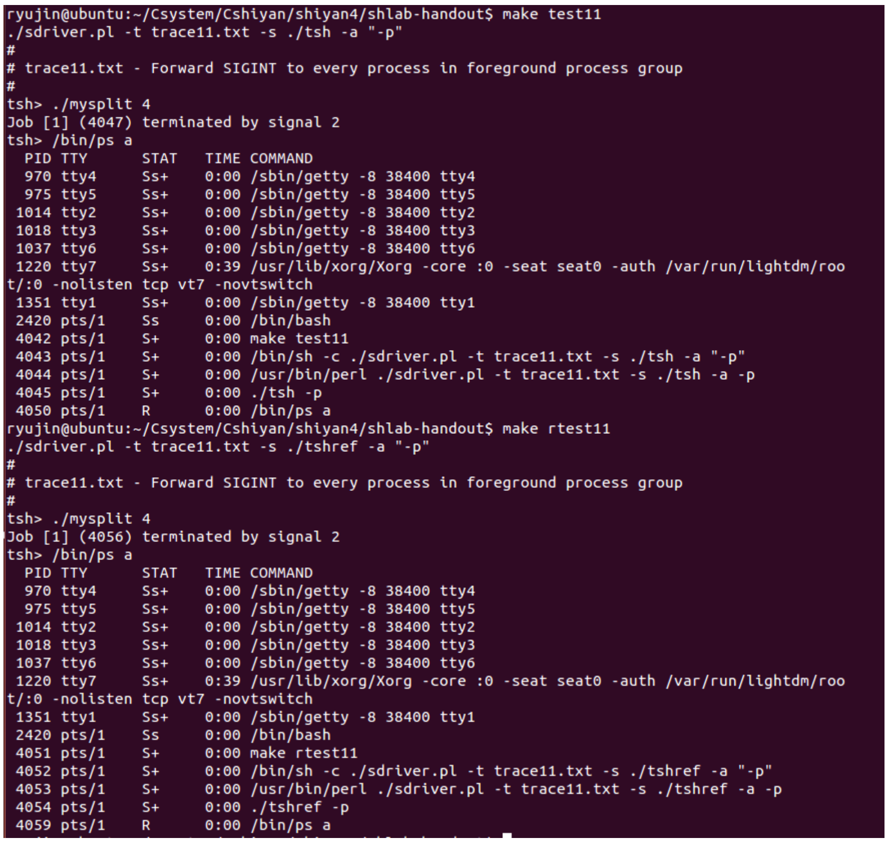
sigint_handler（）函数

```c
void sigint_handler(int sig)//捕获SIGINT信号
{
    if(verbose)
        puts("sigint_handler: entering");
    pid_t pid = fgpid(jobs);

    if(pid){
        // 发送SIGINT给前台进程组里的所有进程
        // 需要注意的是，前台进程组内的进程除了当前前台进程以外，还包括前台进程的子进程。
        // 最多只能存在一个前台进程，但前台进程组内可以存在多个进程
        if(kill(-pid, SIGINT) < 0)
            unix_error("kill (sigint) error");
        if(verbose){
            printf("sigint_handler: Job (%d) killed\n", pid);
        }
    }
    if(verbose)
        puts("sigint_handler: exiting");
    return;
}
```

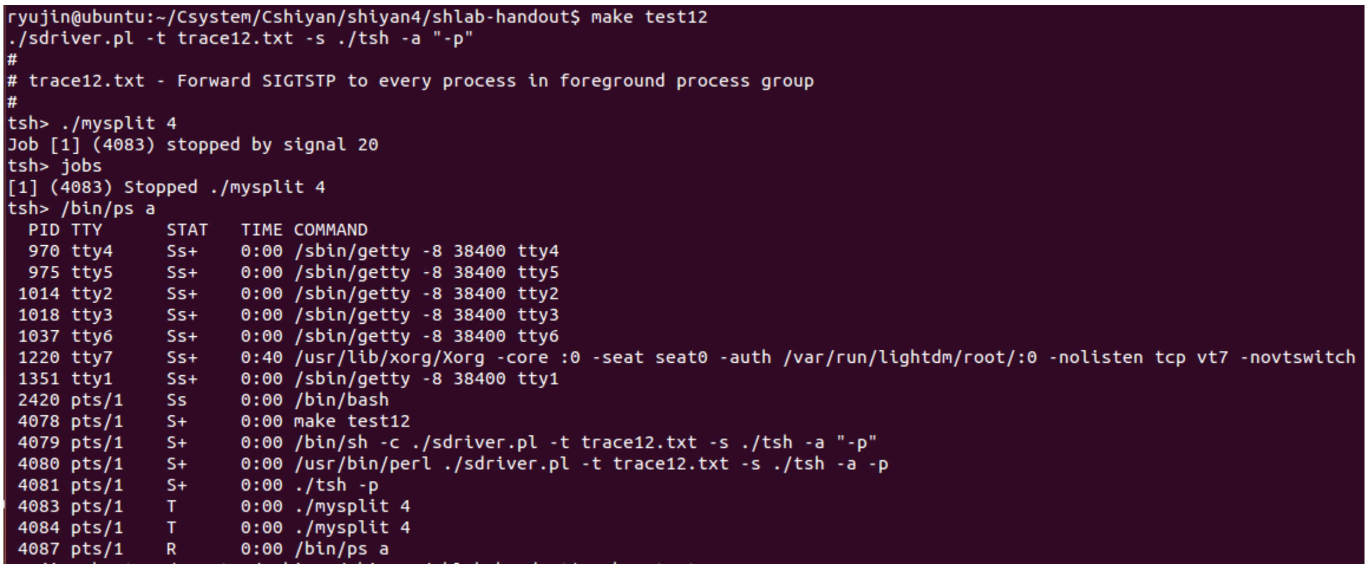
sigtstp_handler（）函数

```c
void sigtstp_handler(int sig)//捕获SIGTSTP信号
{
    if(verbose)
        puts("sigstp_handler: entering");

    pid_t pid = fgpid(jobs);
    struct job_t *job = getjobpid(jobs, pid);//用fgpid(jobs)获取前台进程pid，判断当前是否有前台进程，如果没有直接返回。

    if(pid){
        if(kill(-pid, SIGTSTP) < 0)      //用kill(-pid,sig)函数发送SIGTSPT信号给前台进程组。
            unix_error("kill (tstp) error");
        if(verbose){
            printf("sigstp_handler: Job [%d] (%d) stopped\n", job->jid, pid);
        }
    }
    if(verbose)
        puts("sigstp_handler: exiting");
    return;
}
```

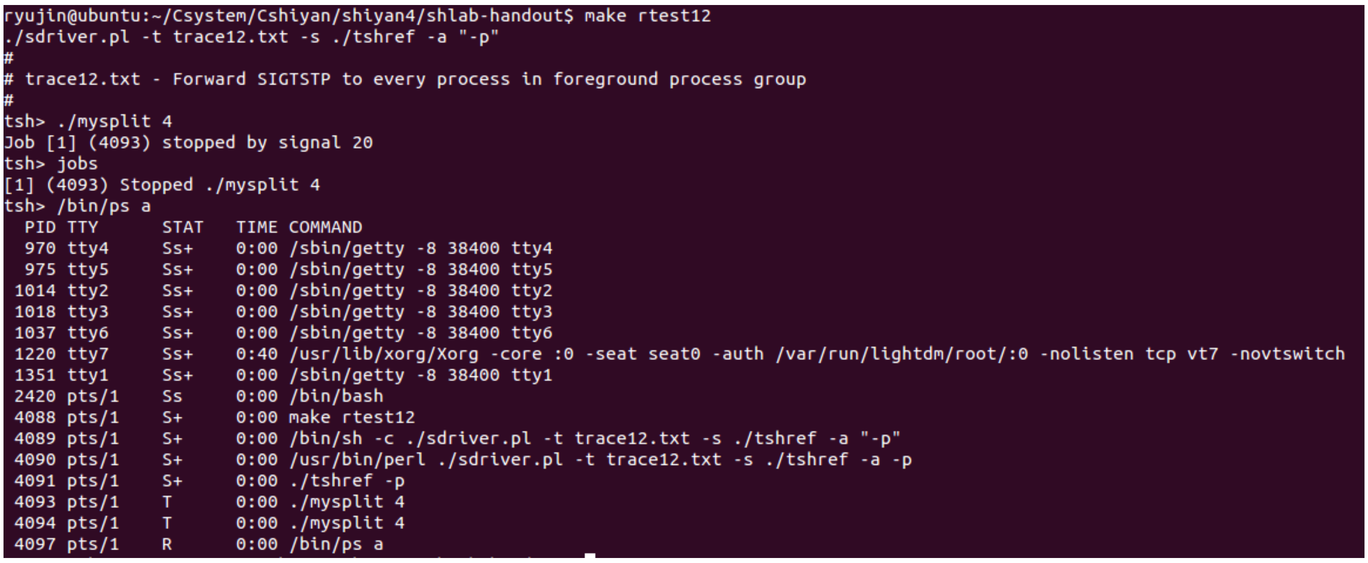
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4047) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:39 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4042 pts/1    S+     0:00 make test11
 4043 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
 4044 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
 4045 pts/1    S+     0:00 ./tsh -p
 4050 pts/1    R      0:00 /bin/ps a
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4056) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:39 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4051 pts/1    S+     0:00 make rtest11
 4052 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
 4053 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tshref -a -p
 4054 pts/1    S+     0:00 ./tshref -p
 4059 pts/1    R      0:00 /bin/ps a
```



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4083) stopped by signal 20
tsh> jobs
[1] (4083) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:40 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4078 pts/1    S+     0:00 make test12
 4079 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
 4080 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
 4081 pts/1    S+     0:00 ./tsh -p
 4083 pts/1    T      0:00 ./mysplit 4
 4084 pts/1    T      0:00 ./mysplit 4
 4087 pts/1    R      0:00 /bin/ps a
```



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4093) stopped by signal 20
tsh> jobs
[1] (4093) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:40 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/root/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4088 pts/1    S+     0:00 make rtest12
 4089 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
 4090 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tshref -a -p
 4091 pts/1    S+     0:00 ./tshref -p
 4093 pts/1    T      0:00 ./mysplit 4
 4094 pts/1    T      0:00 ./mysplit 4
 4097 pts/1    R      0:00 /bin/ps a
```

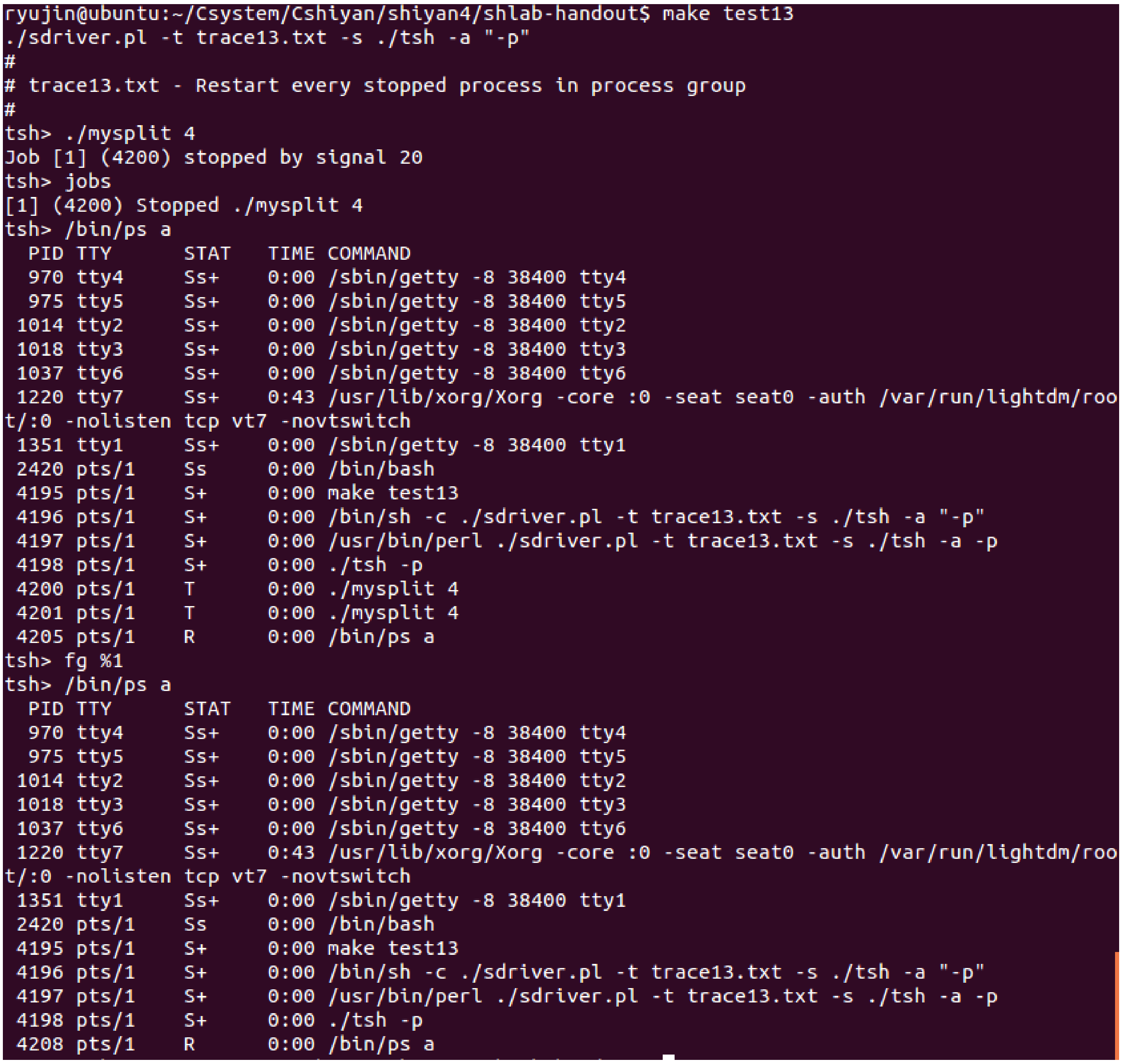
11.trace13

重新启动进程组中的每个已停止的进程

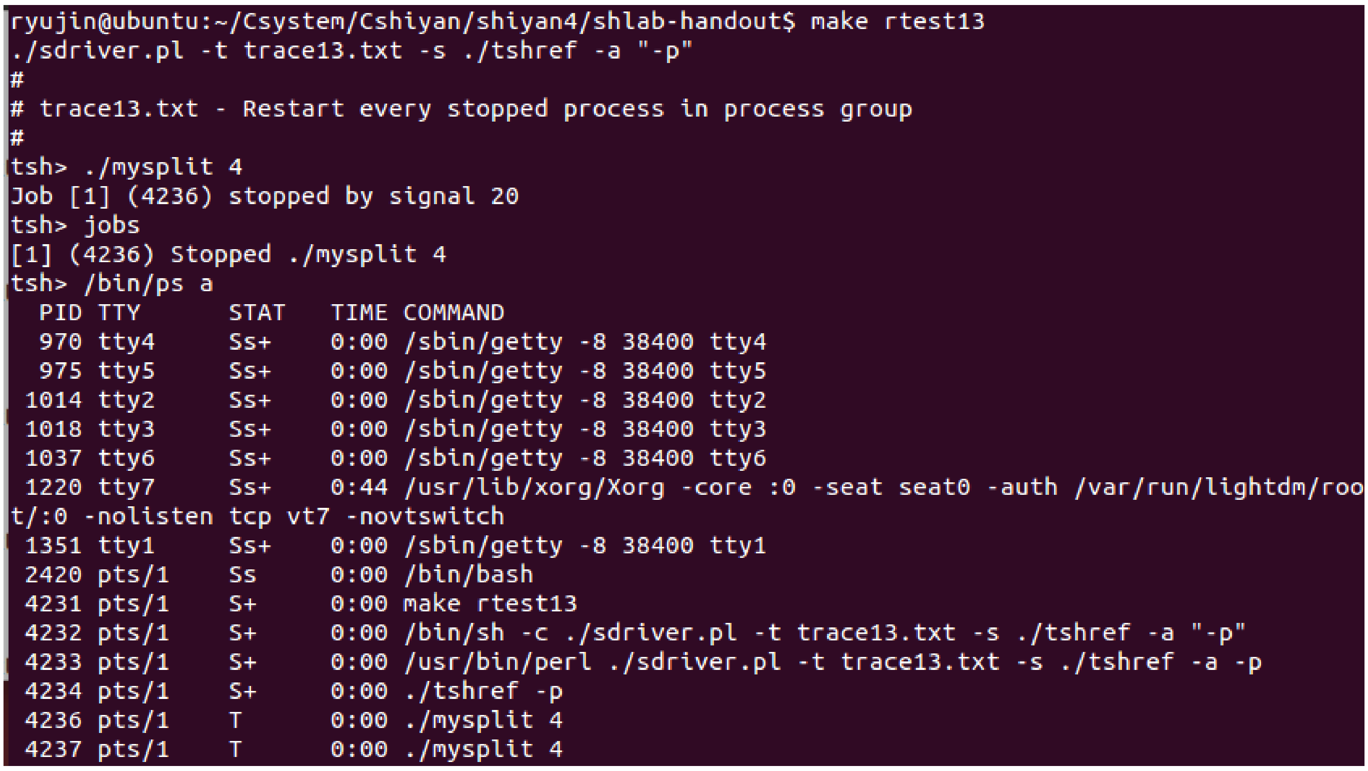
```
#
# trace13.txt - Restart every stopped process in process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> /bin/ps a
/bin/ps a

/bin/echo tsh> fg %1
fg %1

/bin/echo tsh> /bin/ps a
/bin/ps a
```

该程序是为了测试重新启动进程组中的每个停止的进程。这里也就是使用fg来唤醒整个工作，中间使用ps -a来查看停止整个工作和唤醒整个工作的区别。

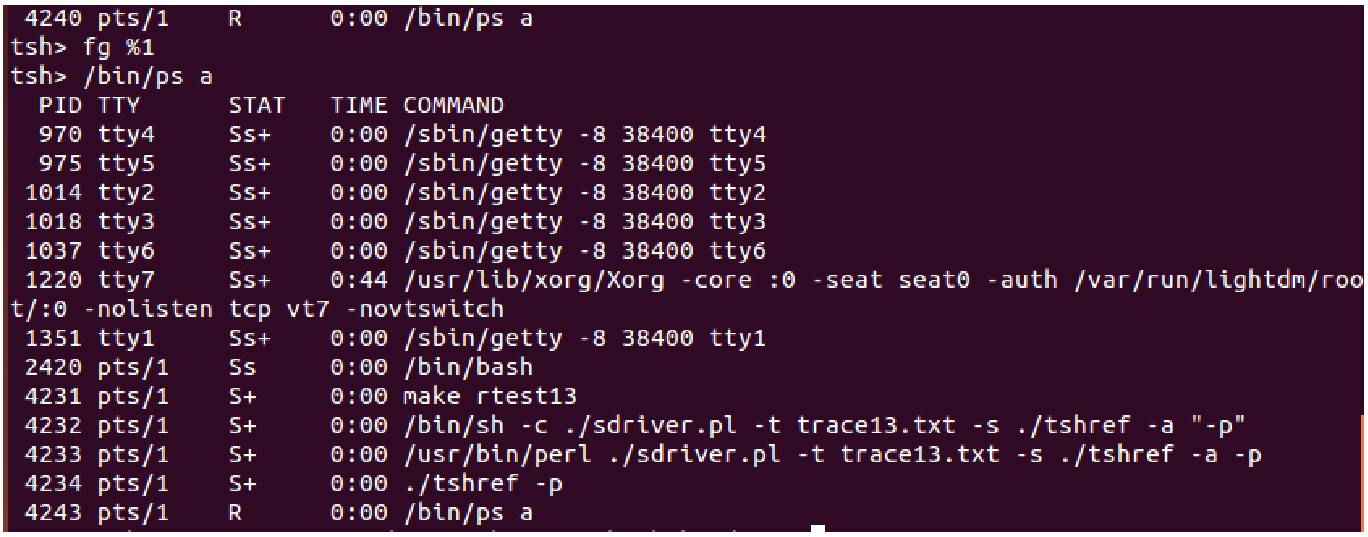
test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4200) stopped by signal 20
tsh> jobs
[1] (4200) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:43 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4195 pts/1    S+     0:00 make test13
 4196 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
 4197 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
 4198 pts/1    S+     0:00 ./tsh -p
 4200 pts/1    T      0:00 ./mysplit 4
 4201 pts/1    T      0:00 ./mysplit 4
 4205 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:43 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4195 pts/1    S+     0:00 make test13
 4196 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
 4197 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
 4198 pts/1    S+     0:00 ./tsh -p
 4208 pts/1    R      0:00 /bin/ps a
```

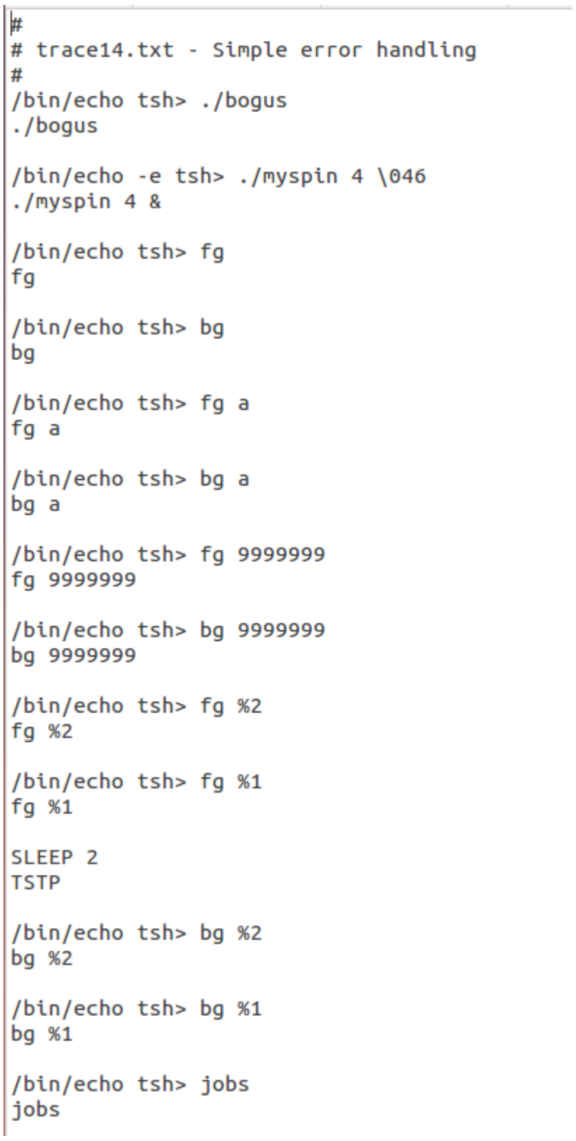


```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (4236) stopped by signal 20
tsh> jobs
[1] (4236) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:44 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4231 pts/1    S+     0:00 make rtest13
 4232 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
 4233 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
 4234 pts/1    S+     0:00 ./tshref -p
 4236 pts/1    T      0:00 ./mysplit 4
 4237 pts/1    T      0:00 ./mysplit 4
```


```
 4240 pts/1    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  970 tty4     Ss+    0:00 /sbin/getty -8 38400 tty4
  975 tty5     Ss+    0:00 /sbin/getty -8 38400 tty5
 1014 tty2     Ss+    0:00 /sbin/getty -8 38400 tty2
 1018 tty3     Ss+    0:00 /sbin/getty -8 38400 tty3
 1037 tty6     Ss+    0:00 /sbin/getty -8 38400 tty6
 1220 tty7     Ss+    0:44 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/lightdm/roo
t/:0 -nolisten tcp vt7 -novtswitch
 1351 tty1     Ss+    0:00 /sbin/getty -8 38400 tty1
 2420 pts/1    Ss     0:00 /bin/bash
 4231 pts/1    S+     0:00 make rtest13
 4232 pts/1    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
 4233 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tshref -a -p
 4234 pts/1    S+     0:00 ./tshref -p
 4243 pts/1    R      0:00 /bin/ps a
```

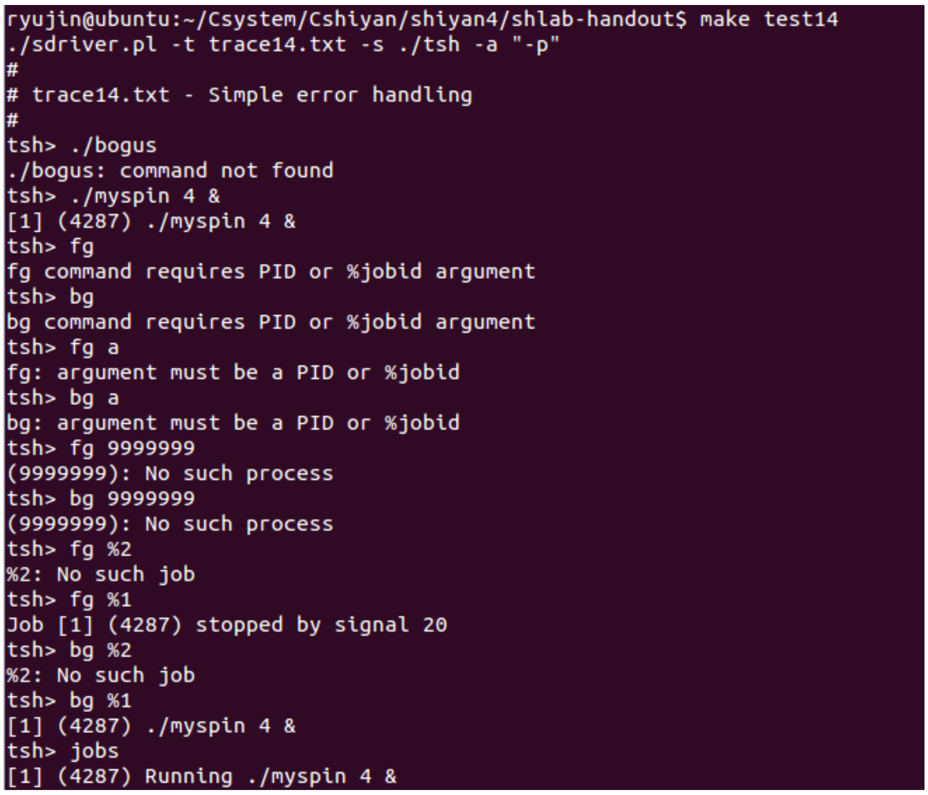
12.trace14

简单的错误处理,处理输入未实现的命令、fg、bg参数不正确等错误情况

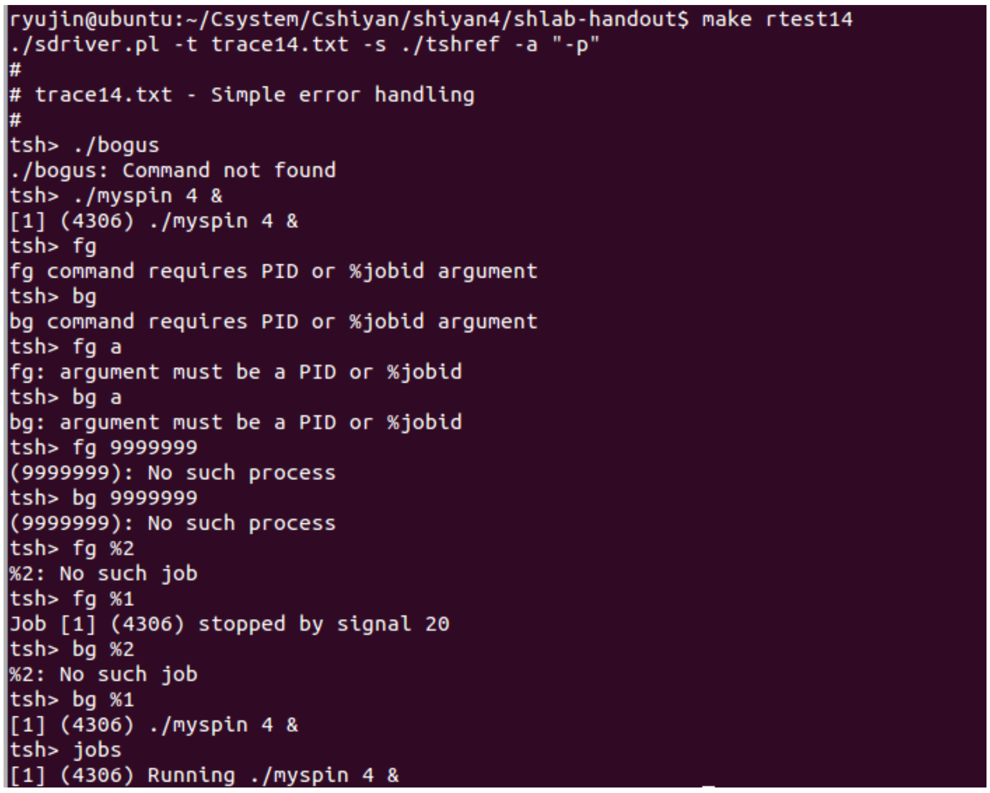

```
#
# trace14.txt - Simple error handling
#
/bin/echo tsh> ./bogus
./bogus

/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo tsh> fg
fg

/bin/echo tsh> bg
bg

/bin/echo tsh> fg a
fg a

/bin/echo tsh> bg a
bg a

/bin/echo tsh> fg 9999999
fg 9999999

/bin/echo tsh> bg 9999999
bg 9999999

/bin/echo tsh> fg %2
fg %2

/bin/echo tsh> fg %1
fg %1

SLEEP 2
TSTP

/bin/echo tsh> bg %2
bg %2

/bin/echo tsh> bg %1
bg %1

/bin/echo tsh> jobs
jobs
```

根据注释可以知道这个文件是为了测试简单的错误处理。这里的测试文件，也就是测试fg和bg后面的参数，我们知道fg和bg后面需要一个JID或者是PID，其中JID是加上%的整型数。其余参数都应该报错，或是没有参数也应该报错。

test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4287) ./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] (4287) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4287) ./myspin 4 &
tsh> jobs
[1] (4287) Running ./myspin 4 &
```

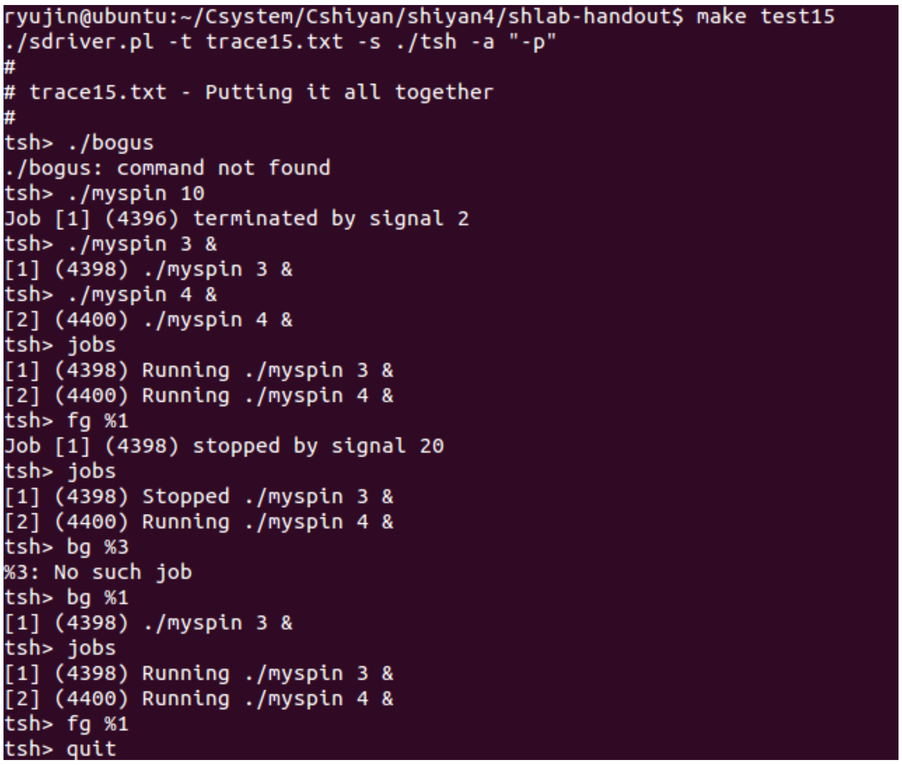


```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest14
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (4306) ./myspin 4 &
tsh> fg
fg command requires PID or %jobid argument
tsh> bg
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] (4306) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (4306) ./myspin 4 &
tsh> jobs
[1] (4306) Running ./myspin 4 &
```

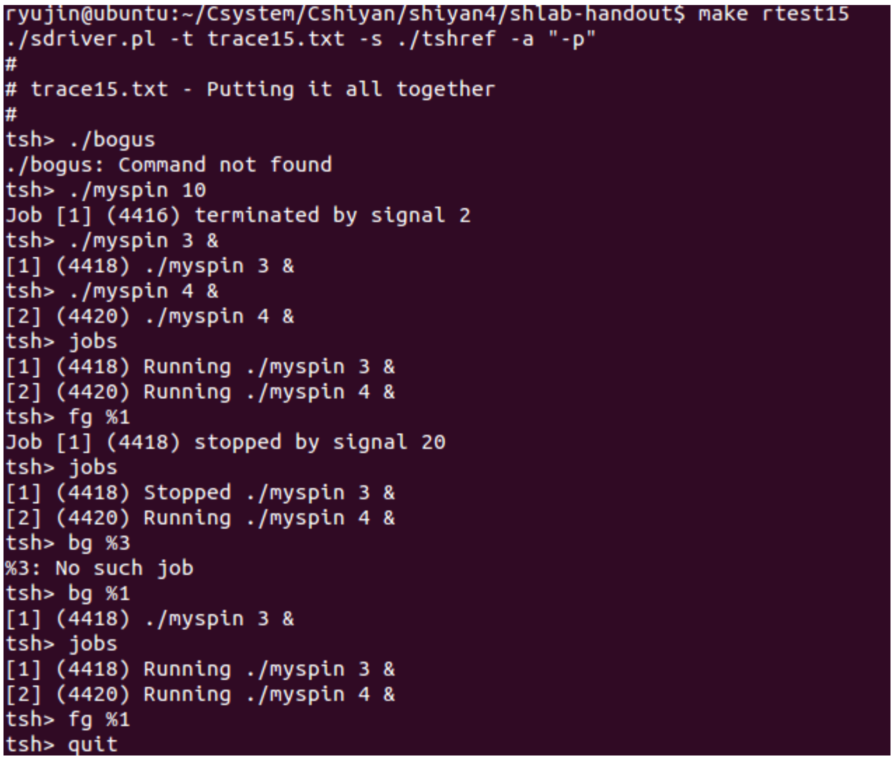
13.trace15

所有命令一起运行

test



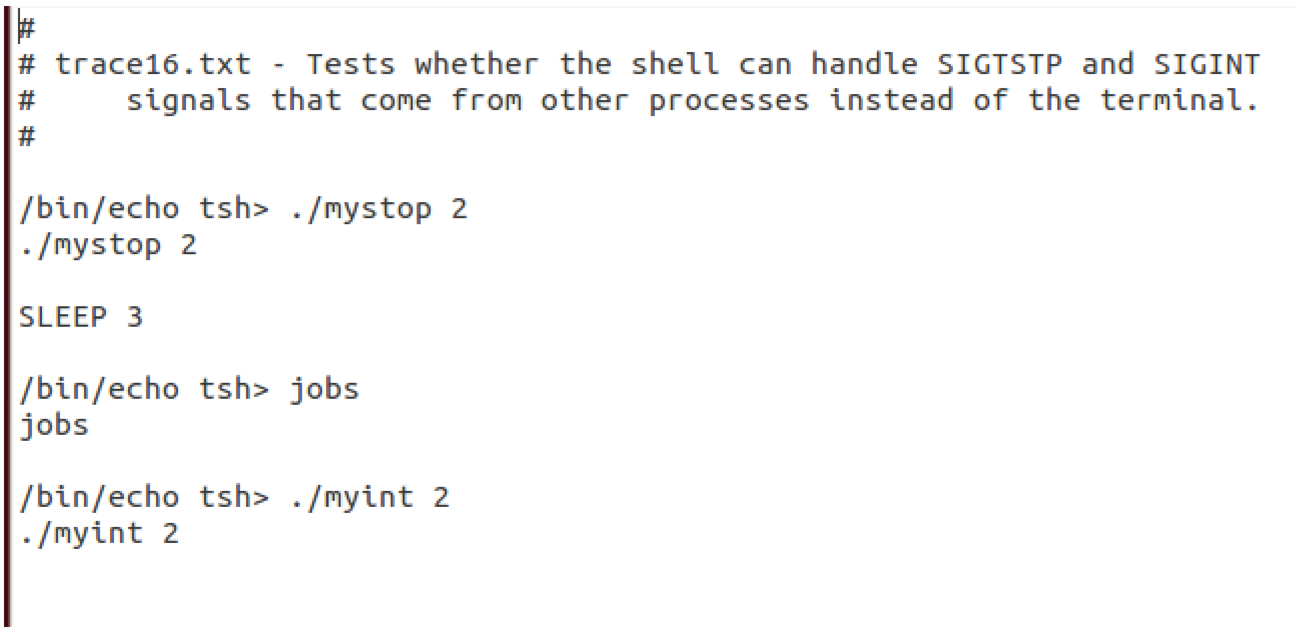
```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: command not found
tsh> ./myspin 10
Job [1] (4396) terminated by signal 2
tsh> ./myspin 3 &
[1] (4398) ./myspin 3 &
tsh> ./myspin 4 &
[2] (4400) ./myspin 4 &
tsh> jobs
[1] (4398) Running ./myspin 3 &
[2] (4400) Running ./myspin 4 &
tsh> fg %1
Job [1] (4398) stopped by signal 20
tsh> jobs
[1] (4398) Stopped ./myspin 3 &
[2] (4400) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (4398) ./myspin 3 &
tsh> jobs
[1] (4398) Running ./myspin 3 &
[2] (4400) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ 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] (4416) terminated by signal 2
tsh> ./myspin 3 &
[1] (4418) ./myspin 3 &
tsh> ./myspin 4 &
[2] (4420) ./myspin 4 &
tsh> jobs
[1] (4418) Running ./myspin 3 &
[2] (4420) Running ./myspin 4 &
tsh> fg %1
Job [1] (4418) stopped by signal 20
tsh> jobs
[1] (4418) Stopped ./myspin 3 &
[2] (4420) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (4418) ./myspin 3 &
tsh> jobs
[1] (4418) Running ./myspin 3 &
[2] (4420) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

14.trace16

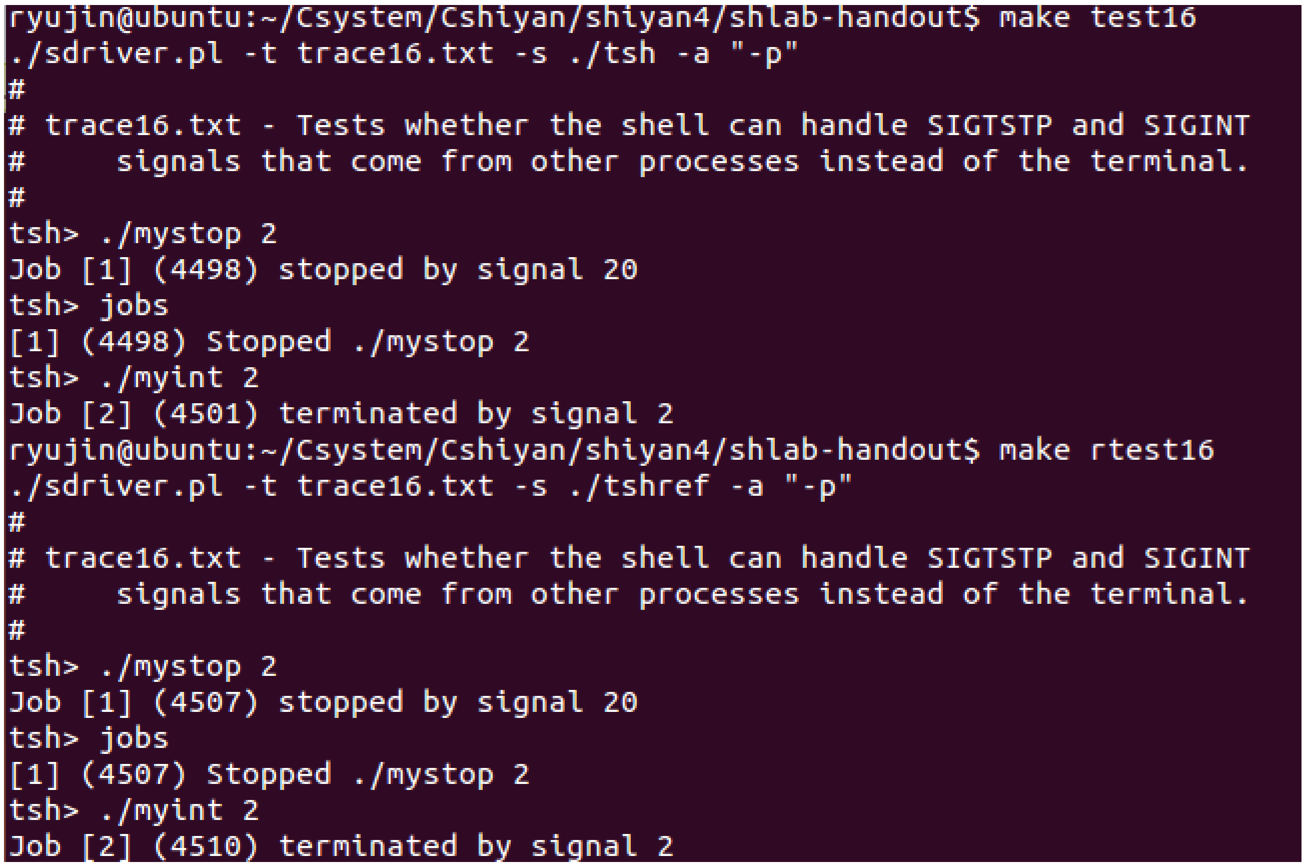
测试shell是否能够处理来自其他进程而不是终端的SIGTSTP和SIGINT信号



```
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#

/bin/echo tsh> ./mystop 2
./mystop 2

SLEEP 3

/bin/echo tsh> jobs
jobs

/bin/echo tsh> ./myint 2
./myint 2
```

用户程序向job 2传送了中止信号，所以最后会输出进程2被中止的信息。同时，mystop需要自己停止才能给别的进程发送信号，所以中间也会出现进程1被中止的信息

test



```
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make test16
./sdriver.pl -t trace16.txt -s ./tsh -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (4498) stopped by signal 20
tsh> jobs
[1] (4498) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (4501) terminated by signal 2
ryujin@ubuntu:~/Csystem/Cshiyan/shiyan4/shlab-handout$ make rtest16
./sdriver.pl -t trace16.txt -s ./tshref -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (4507) stopped by signal 20
tsh> jobs
[1] (4507) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (4510) terminated by signal 2
```

六.总结

了解Shell的任务控制，学习进程控制(创建新进程，回收僵尸进程)和信号处理（阻塞信号，信号处理程序）。了解了ctrl-c和ctrl-z按键、fg、bg和jobs命令，对Unix的进程控制、信号和信号处理有大致的了解。