ShellLab

The Function to be completed

- *eval*: Main routine that parses and interprets the command line. [70 lines]
- *builtin cmd* : Recognizes and interprets the built-in commands: quit, fg, bg, and jobs. [25 lines]
- *do bgfg* : Implements the bg and fg built-in commands. [50 lines]
- *waitfg* : Waits for a foreground job to complete. [20 lines]
- *sigchld handler* : Catches SIGCHILD signals. 80 lines]
- *sigint handler* : Catches SIGINT (ctrl-c) signals. [15 lines]
- *sigtstp handler* : Catches SIGTSTP (ctrl-z) signals. [15 lines]

eval

这个函数功能是解析命令行后判断为内置命令还是程序路径，分别执行。如果是前台作业，则要等待其完成，如果是后台作业，则要输出其相应信息

```
/*
 * 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) {
        char *argv[MAXARGS];        //存放解析的参数
        char buf[MAXLINE];          //解析cmdline
        int bg;                     //判断程序是前台还是后台执行
        int state;                  //指示前台还是后台运行状态
        pid_t pid;                  //执行程序的子进程的pid

        strcpy(buf, cmdline);
        bg = parseline(buf, argv);  //解析参数
        state = bg ? BG : FG;
        if (argv[0] == NULL)        //空行，直接返回
                return;
        sigset_t mask_all, mask_one, prev_one;
        sigfillset(&mask_all);
        sigemptyset(&mask_one);
        sigaddset(&mask_one, SIGCHLD);
        if (!builtin_cmd(argv)) {                          //判断是否为内置命令
                sigprocmask(SIG_BLOCK, &mask_one, &prev_one);       //fork前阻塞
SIGCHLD信号
                if ((pid = Fork()) == 0) {                          //创建子进程
                        sigprocmask(SIG_SETMASK, &prev_one, NULL);      //解除子进程
的阻塞
                        Setpgid(0, 0);                                  //创建新进程
组，ID设置为进程PID
                        execve(argv[0], argv, environ);                 //执行
                        exit(0);                                        //子线程执行
完毕后一定要退出
                }
                if (state == FG) {
                        sigprocmask(SIG_BLOCK, &mask_all, NULL);            //添加工
作前阻塞所有信号
                        addjob(jobs, pid, state, cmdline);                  //添加至
作业列表
                        sigprocmask(SIG_SETMASK, &mask_one, NULL);
                        waitfg(pid);                                        //等待前
台进程执行完毕
                } else {
                        sigprocmask(SIG_BLOCK, &mask_all, NULL);            //添加工
作前阻塞所有信号
                        addjob(jobs, pid, state, cmdline);                  //添加至
作业列表
                        sigprocmask(SIG_SETMASK, &mask_one, NULL);
                        printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline); //打印后
台进程信息
                }
                sigprocmask(SIG_SETMASK, &prev_one, NULL);          //解除阻塞
        }
        return;
}
```

1. 注意保存和恢复 errno。很多函数会在出错返回式设置 errno，在处理程序中调用这样的函数可能会干扰主程序中其他依赖于 errno 的部分，解决办法是在进入处理函数时用局部变量保存它，运行完成后再将其恢复
2. 访问全局数据时，阻塞所有信号。`sigprocmask`
   ❗ **为什么要在执行前创建新线程组?**

> 因为这里主要是为了将子进程组与 tsh 进程组分开，防止发信号终止子进程组时也将 tsh 进程组终止了
> pdf中的hint也有提及：After the fork, but before the execve, the child process should call setpgid(0, 0), which puts the child in a new process group whose group ID is identical to the child’s PID. This ensures that there will be only one process, your shell, in the foreground process group.

builtin_cmd

```
int builtin_cmd(char **argv) {
        if (!strcmp(argv[0], "quit")) //如果命令是quit，退出
                exit(0);
        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);
        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);
        else
                return 0;     /* not a builtin command */
        return 1;
}
```

do_bgfg

```
//do_bgfg - Execute the builtin bg and fg commands
void do_bgfg(char **argv) {
        struct job_t *job = NULL;        //要处理的job
        int state;                      //输入的命令
        int id;                         //存储jid或pid
        if (!strcmp(argv[0], "bg"))
                state = BG;
        else
                state = FG;
        if (argv[1] == NULL) {           //没带参数
                printf("%s command requires PID or %%jobid argument\n", argv[0]);
                return;
        }
        if (argv[1][0] == '%') {         //说明是jid
                if (sscanf(&argv[1][1], "%d", &id) > 0) {
                        job = getjobjid(jobs, id);  //获得job
                        if (job == NULL) {
                                printf("%%%d: No such job\n", id);
                                return;
                        }
                }
        } else if (!isdigit(argv[1][0])) { //其它符号，非法输入
                printf("%s: argument must be a PID or %%jobid\n", argv[0]);
                return;
        } else {                    //pid
                id = atoi(argv[1]);
                job = getjobpid(jobs, id);
                if (job == NULL) {
                        printf("(%d): No such process\n", id);
                        return;
                }

        }
        kill(-(job->pid), SIGCONT);       //重启进程, 这里发送到进程组
        job->state = state;
        if (state == BG)
                printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);
        else
                waitfg(job->pid);
        return;
}
```

waitfg

```
/*
 * waitfg - Block until process pid is no longer the foreground process
 */
void waitfg(pid_t pid) {
        sigset_t mask;
        sigemptyset(&mask);
        while (fgpid(jobs) != 0) { //前台无任务return 0
                sigsuspend(&mask);      //暂停时取消阻塞,见sigsuspend用法
        }
        return;
}
```

sigchld_handler

```
/*
 * sigchld_handler - The kernel sends a SIGCHLD to the shell whenever
 *     a child job terminates (becomes a zombie), or stops because it
 *     received a SIGSTOP or SIGTSTP signal. The handler reaps all
 *     available zombie children, but doesn't wait for any other
 *     currently running children to terminate.
 */
void sigchld_handler(int sig) {
        int olderrno = errno;   //由于errno是全局变量,注意保存和恢复errno
        int status;
        pid_t pid;
        struct job_t *job;
        sigset_t mask, prev;
        sigfillset(&mask);
        while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) {     //立即返
回该子进程的pid
                sigprocmask(SIG_BLOCK, &mask, &prev);   //阻塞所有信号
                if (WIFEXITED(status)) {                //正常终止
                        deletejob(jobs, pid);
                } else if (WIFSIGNALED(status)) {       //因为信号而终止, 打印
                        printf ("Job [%d] (%d) terminated by signal %d\n",
pid2jid(pid), pid, WTERMSIG(status));
                        deletejob(jobs, pid);
                } else if (WIFSTOPPED(status)) {        //因为信号而停止, 打印
                        printf ("Job [%d] (%d) stoped by signal %d\n",
pid2jid(pid), pid, WSTOPSIG(status));
                        job = getjobpid(jobs, pid);
                        job->state = ST;
                }
                sigprocmask(SIG_SETMASK, &prev, NULL);
        }
        errno = olderrno;
        return;
}
```

在这个函数中，`waitpid`起了很大的作用
这个函数用来挂起调用进程的执行，直到`pid`对应的等待集合的一个子进程的改变才返回，包括三种状态的改变：

- 子进程终止
- 子进程收到信号停止
- 子进程收到信号重新执行

如果一个子进程在调用之前就已经终止了，那么函数就会立即返回，否则，就会阻塞，直到一个子进程改变状态。

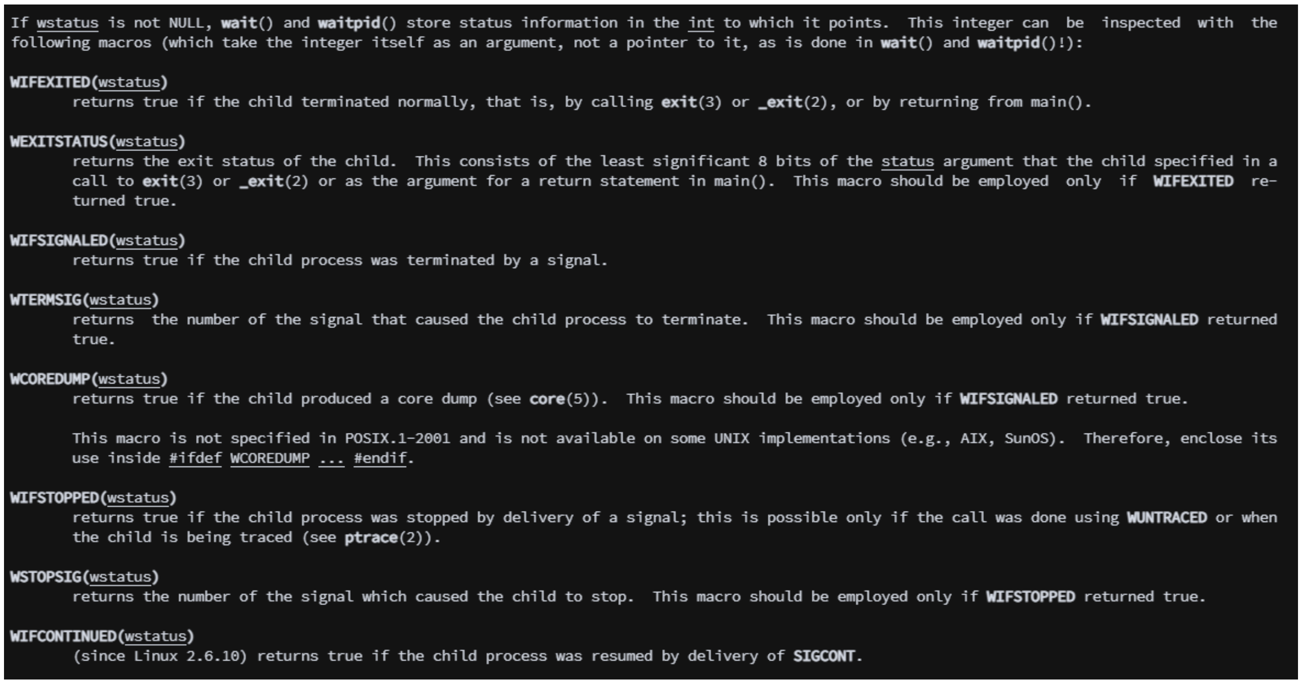
等待集合以及监测那些状态都是用函数的参数确定的，函数定义如下：

```
pid_t waitpid(pid_t pid, int *wstatus, int options);
```

各参数含义及使用

- **pid：判定等待集合成员**
- pid > 0 : 等待集合为 pid 对应的单独子进程
- pid = -1: 等待集合为所有的子进程
- pid < -1: 等待集合为一个进程组，ID 为 pid 的绝对值
- pid = 0 : 等待集合为一个进程组，ID 为调用进程的 pid
- **options：修改默认行为**
- WNOHANG：集合中任何子进程都未终止，立即返回 0
- WUNTRACED：阻塞，直到一个进程终止或停止，返回 PID
- WCONTINUED：阻塞，直到一个停止的进程收到 SIGCONT 信号重新开始执行
- 也可以用或运算把 options 的选项组合起来。例如 WNOHANG | WUNTRACED 表示：立即返回，如果等待集合中的子进程都没有被停职或终止，则返回值为 0；如果有一个停止或终止，则返回值为该子进程的 PID

- **statusp：检查已回收子进程的退出状态**



```
If wstatus is not NULL, wait() and waitpid() store status information in the int to which it points.  This integer can  be  inspected  with  the
following macros (which take the integer itself as an argument, not a pointer to it, as is done in wait() and waitpid()!):

WIFEXITED(wstatus)
       returns true if the child terminated normally, that is, by calling exit(3) or _exit(2), or by returning from main().

WEXITSTATUS(wstatus)
       returns the exit status of the child.  This consists of the least significant 8 bits of the status argument that the child specified in a
       call to exit(3) or _exit(2) or as the argument for a return statement in main().  This macro should be employed  only  if  WIFEXITED  re-
       turned true.

WIFSIGNALED(wstatus)
       returns true if the child process was terminated by a signal.

WTERMSIG(wstatus)
       returns  the number of the signal that caused the child process to terminate.  This macro should be employed only if WIFSIGNALED returned
       true.

WCOREDUMP(wstatus)
       returns true if the child produced a core dump (see core(5)).  This macro should be employed only if WIFSIGNALED returned true.

       This macro is not specified in POSIX.1-2001 and is not available on some UNIX implementations (e.g., AIX, SunOS).  Therefore, enclose its
       use inside #ifdef WCOREDUMP ... #endif.

WIFSTOPPED(wstatus)
       returns true if the child process was stopped by delivery of a signal; this is possible only if the call was done using WUNTRACED or when
       the child is being traced (see ptrace(2)).

WSTOPSIG(wstatus)
       returns the number of the signal which caused the child to stop.  This macro should be employed only if WIFSTOPPED returned true.

WIFCONTINUED(wstatus)
       (since Linux 2.6.10) returns true if the child process was resumed by delivery of SIGCONT.
```

- waitpid 会在 status 中放上关于导致返回的子进程的状态信息

sigint_handler

实现一个 SIGINT 信号处理函数，将信号传送给前台进程

```c
/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *    user types ctrl-c at the keyboard.  Catch it and send it along
 *    to the foreground job.
 */
void sigint_handler(int sig) {
        int olderrno = errno;
        int pid;
        sigset_t mask_all, prev;
        sigfillset(&mask_all);
        sigprocmask(SIG_BLOCK, &mask_all, &prev);   //jobs为全局变量
        if ((pid = fgpid(jobs)) != 0) {
                sigprocmask(SIG_SETMASK, &prev, NULL);
                kill(-pid, SIGINT);
        }
        errno = olderrno;
        return;
}
```

sigtstp_handler

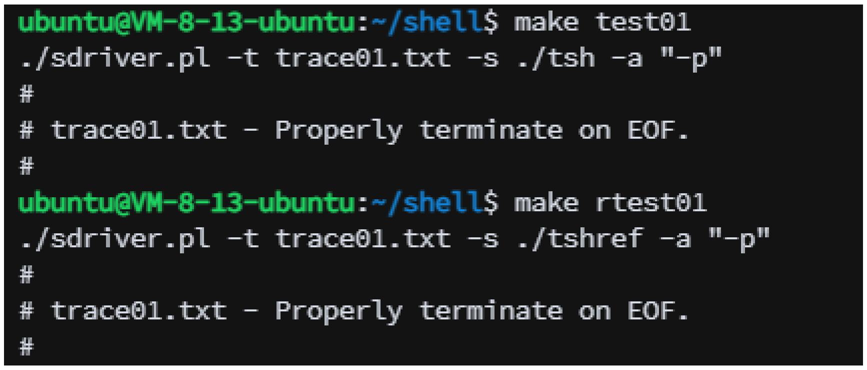
```
/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *     the user types ctrl-z at the keyboard. Catch it and suspend the
 *     foreground job by sending it a SIGTSTP.
 */
void sigtstp_handler(int sig) {
        int olderrno = errno;
        int pid;
        sigset_t mask_all, prev;
        sigfillset(&mask_all);
        sigprocmask(SIG_BLOCK, &mask_all, &prev);
        if ((pid = fgpid(jobs)) > 0) { //如果找不到job会返回0
                sigprocmask(SIG_SETMASK, &prev, NULL);
                kill(-pid, SIGSTOP);
        }
        errno = olderrno;
        return;

}
```

结果检查

trace01

txt内容

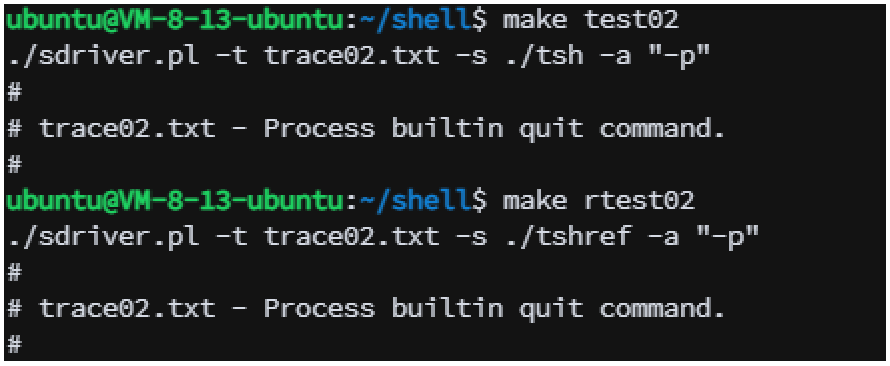
```
#
# trace01.txt - Properly terminate on EOF.
#
CLOSE
WAIT
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

trace02 quit command

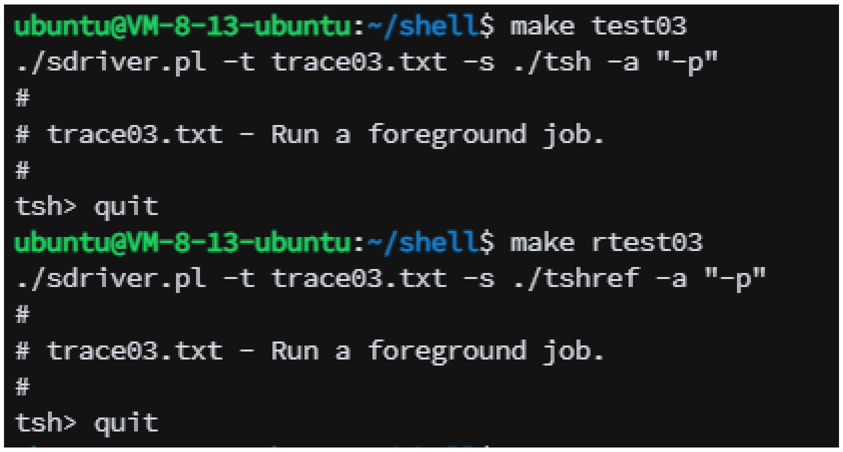
```
#
# trace02.txt - Process builtin quit command.
#
quit
WAIT
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
```

trace03 Run a foreground job

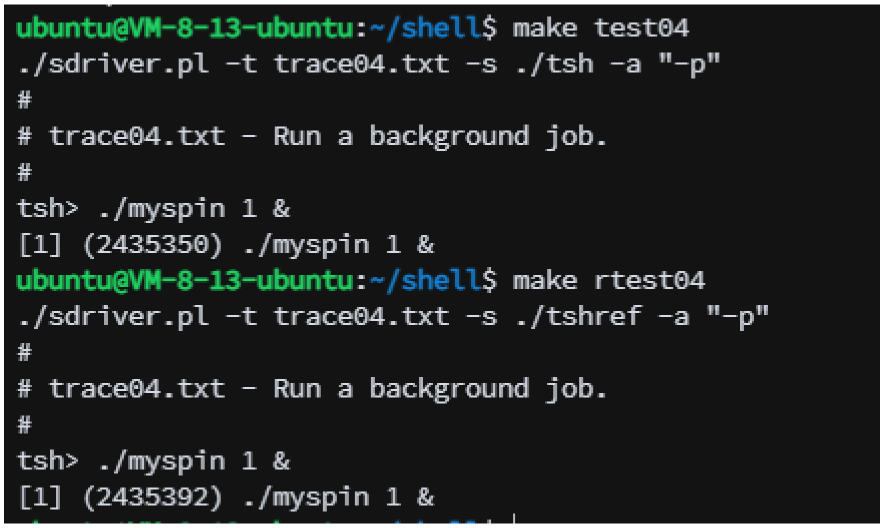
```
#

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


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

trace04 Run a background job.

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

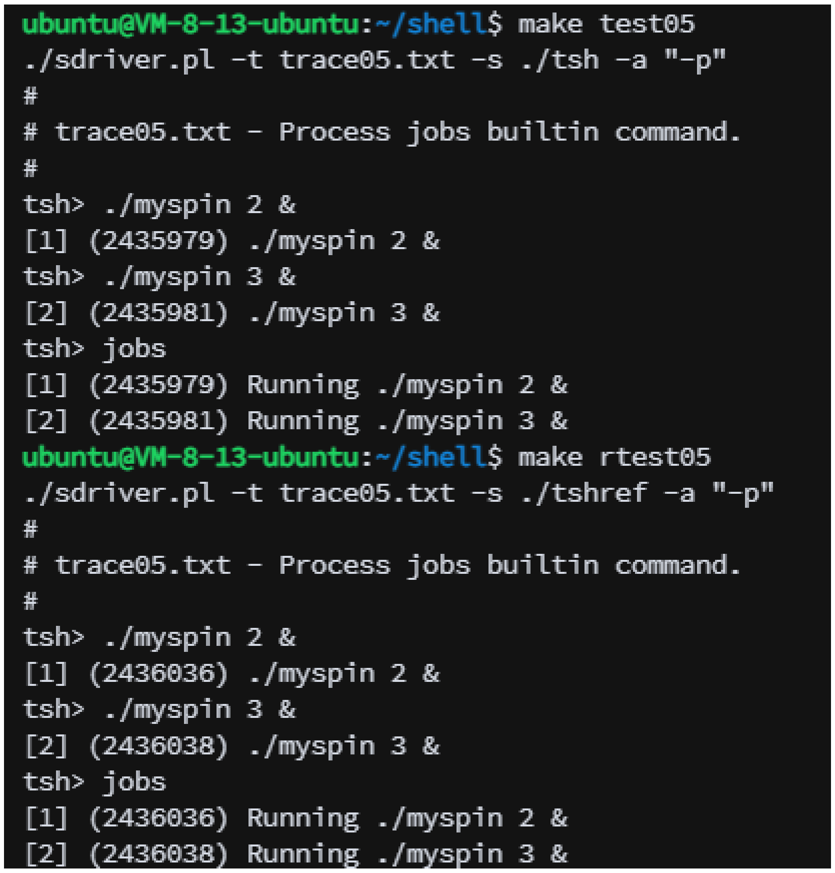
先在前台执行echo命令，等待程序执行完毕回收子进程。&代表是一个后台程序，myspin睡眠1秒，然后停止。因为在后台，所以显示下面一句，如果在前台则无


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (2435350) ./myspin 1 &
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (2435392) ./myspin 1 &
```

trace05 Process jobs builtin command

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

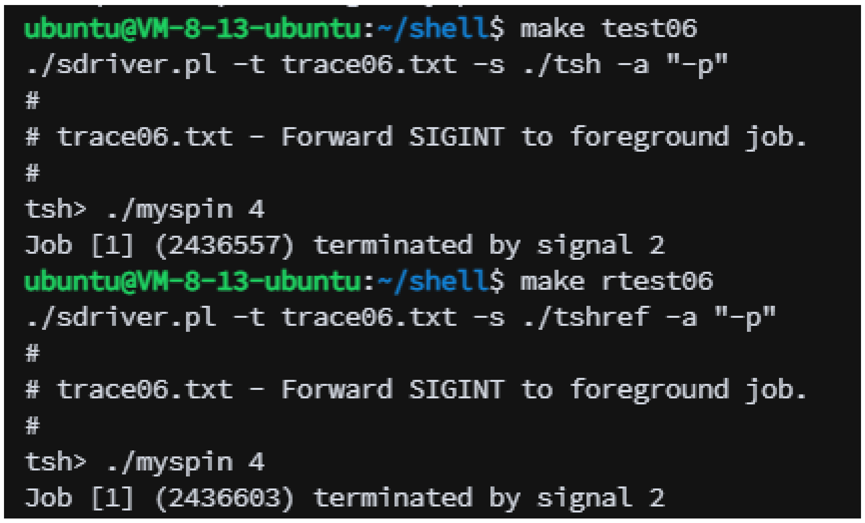
```
/bin/echo -e tsh> ./myspin 3 \046
./myspin 3 &

/bin/echo tsh> jobs
jobs
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (2435979) ./myspin 2 &
tsh> ./myspin 3 &
[2] (2435981) ./myspin 3 &
tsh> jobs
[1] (2435979) Running ./myspin 2 &
[2] (2435981) Running ./myspin 3 &
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (2436036) ./myspin 2 &
tsh> ./myspin 3 &
[2] (2436038) ./myspin 3 &
tsh> jobs
[1] (2436036) Running ./myspin 2 &
[2] (2436038) Running ./myspin 3 &
```

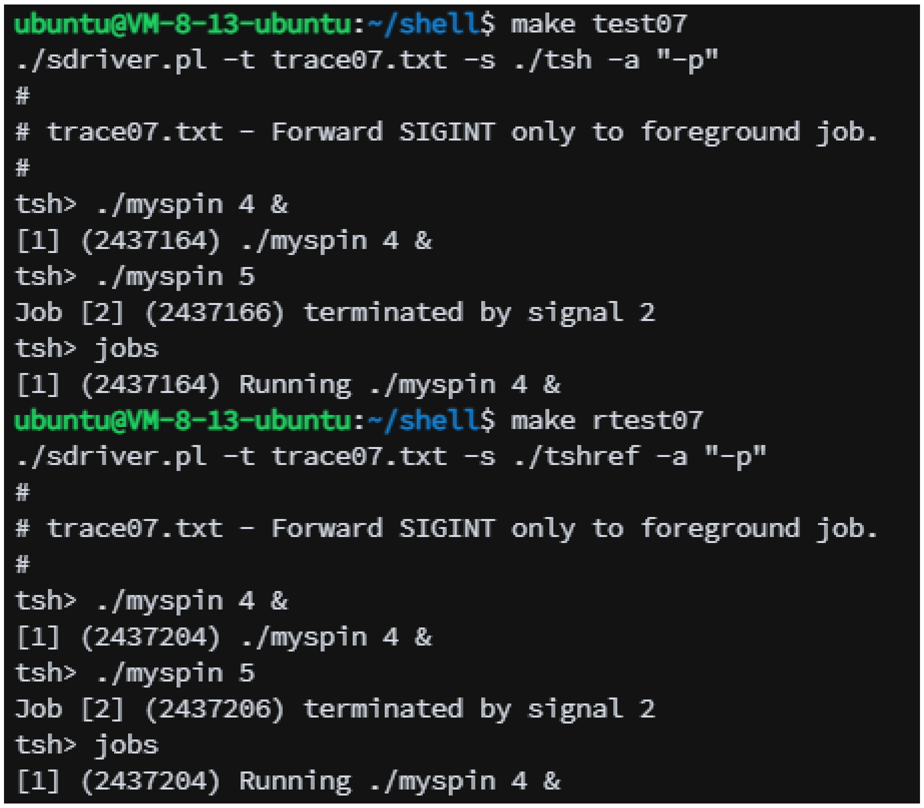
trace06 Forward SIGINT to foreground job

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

SLEEP 2
INT
```


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (2436557) terminated by signal 2
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (2436603) terminated by signal 2
```

trace07 Forward SIGINT only to foreground job

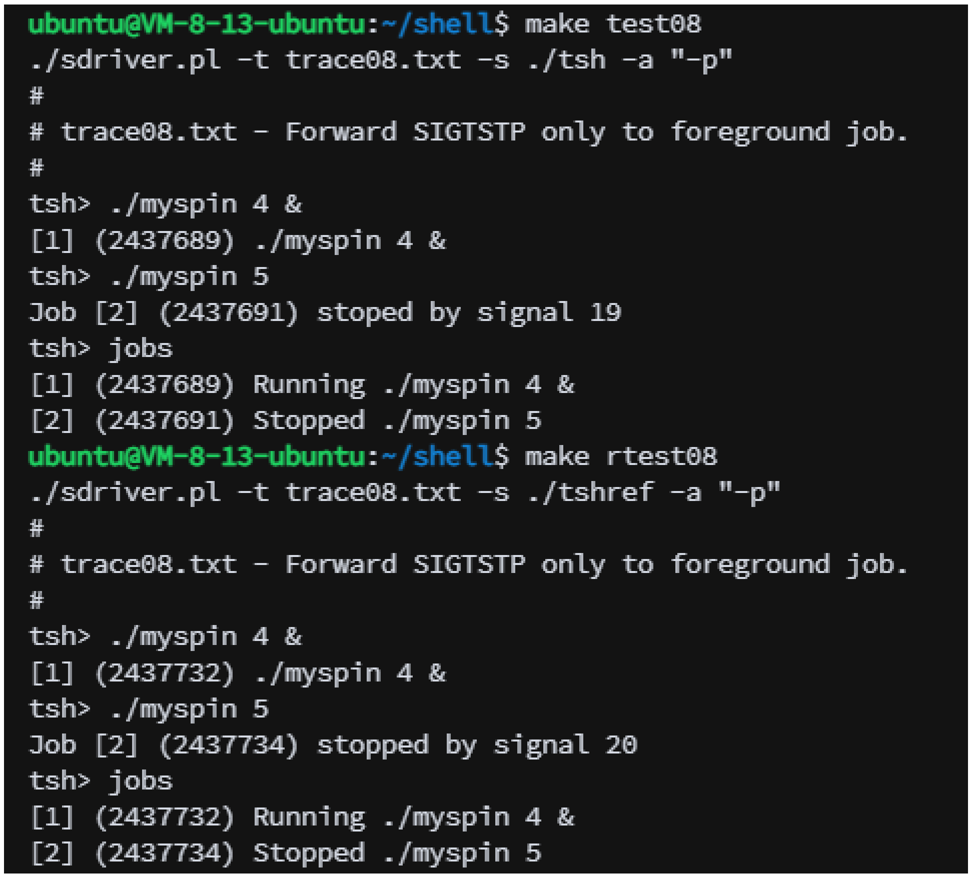
```
#
# trace07.txt - Forward SIGINT only to foreground job.
#
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

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

SLEEP 2
INT

/bin/echo tsh> jobs
jobs
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (2437164) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2437166) terminated by signal 2
tsh> jobs
[1] (2437164) Running ./myspin 4 &
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (2437204) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2437206) terminated by signal 2
tsh> jobs
[1] (2437204) Running ./myspin 4 &
```

trace08 Forward SIGTSTP only to foreground job

```
#
# 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
```

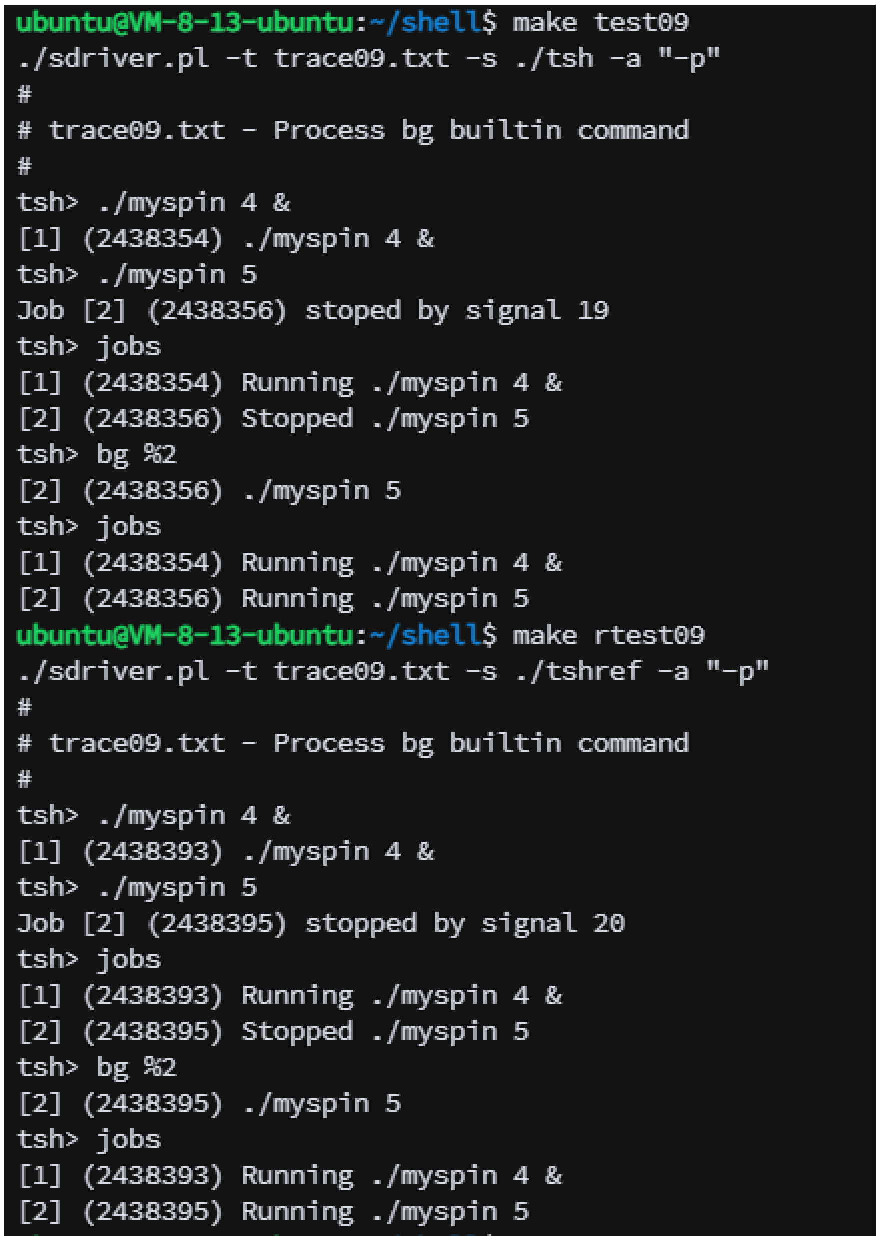


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (2437689) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2437691) stoped by signal 19
tsh> jobs
[1] (2437689) Running ./myspin 4 &
[2] (2437691) Stopped ./myspin 5
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (2437732) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2437734) stopped by signal 20
tsh> jobs
[1] (2437732) Running ./myspin 4 &
[2] (2437734) Stopped ./myspin 5
```

trace09 Process bg builtin command

```
#
# 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
```

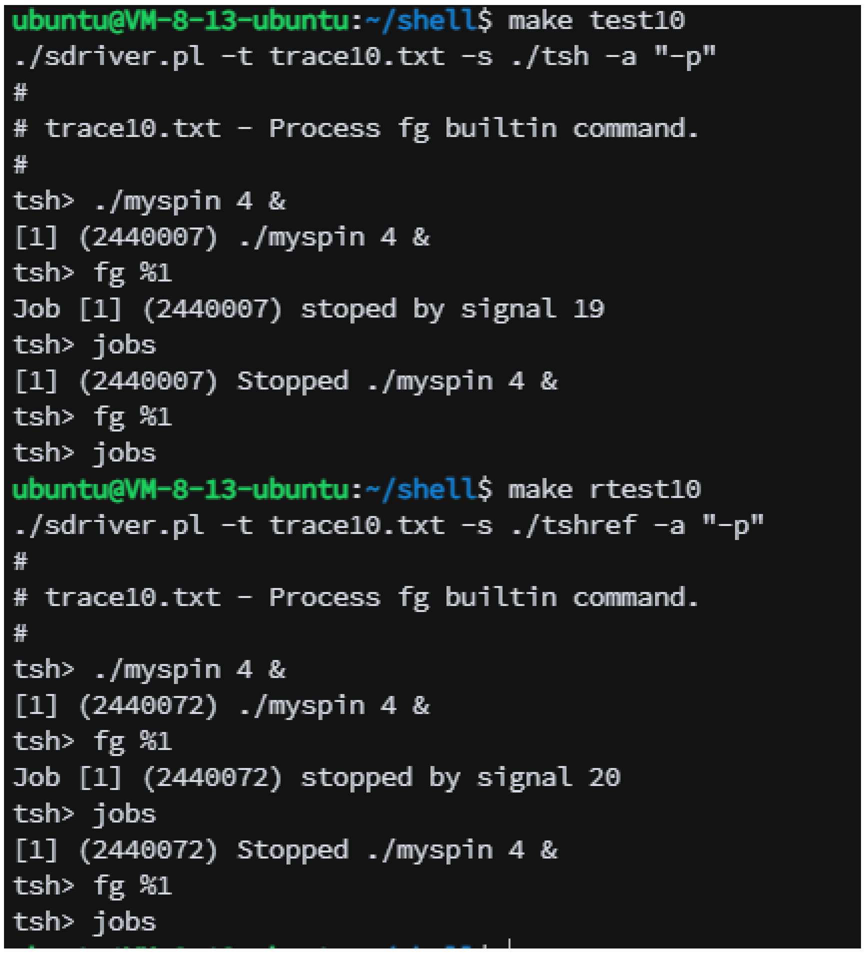


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (2438354) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2438356) stoped by signal 19
tsh> jobs
[1] (2438354) Running ./myspin 4 &
[2] (2438356) Stopped ./myspin 5
tsh> bg %2
[2] (2438356) ./myspin 5
tsh> jobs
[1] (2438354) Running ./myspin 4 &
[2] (2438356) Running ./myspin 5
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (2438393) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2438395) stopped by signal 20
tsh> jobs
[1] (2438393) Running ./myspin 4 &
[2] (2438395) Stopped ./myspin 5
tsh> bg %2
[2] (2438395) ./myspin 5
tsh> jobs
[1] (2438393) Running ./myspin 4 &
[2] (2438395) Running ./myspin 5
```

trace10 Process fg builtin command

```
#
# 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
```

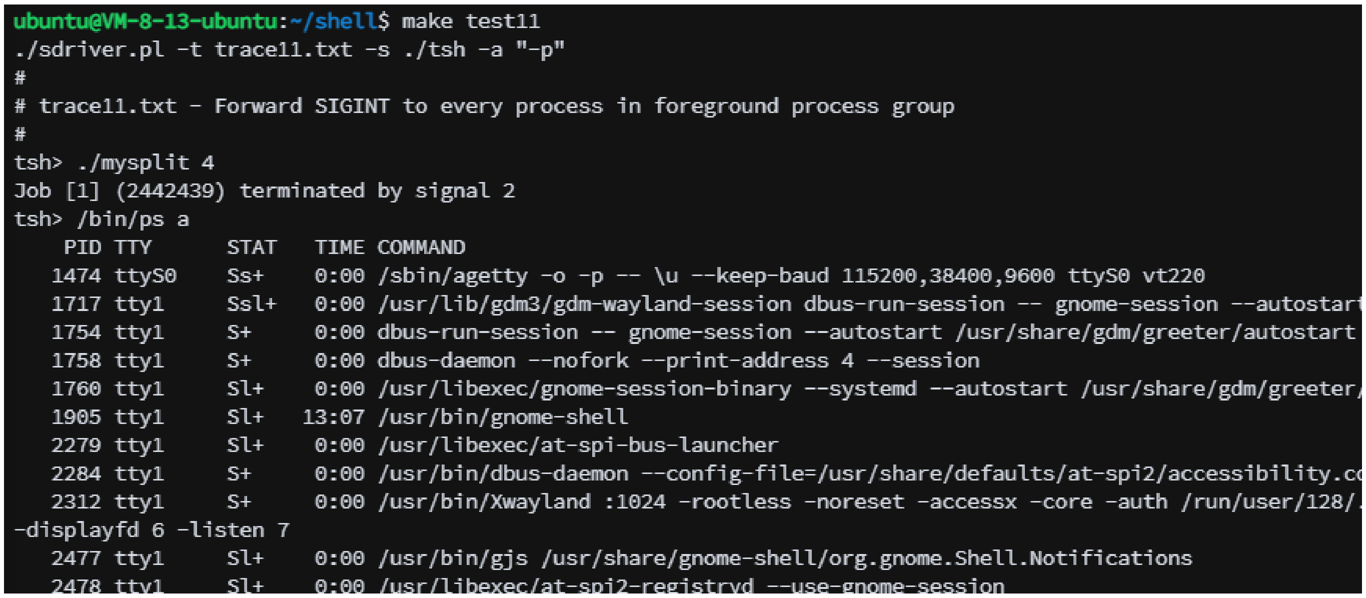


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (2440007) ./myspin 4 &
tsh> fg %1
Job [1] (2440007) stoped by signal 19
tsh> jobs
[1] (2440007) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (2440072) ./myspin 4 &
tsh> fg %1
Job [1] (2440072) stopped by signal 20
tsh> jobs
[1] (2440072) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

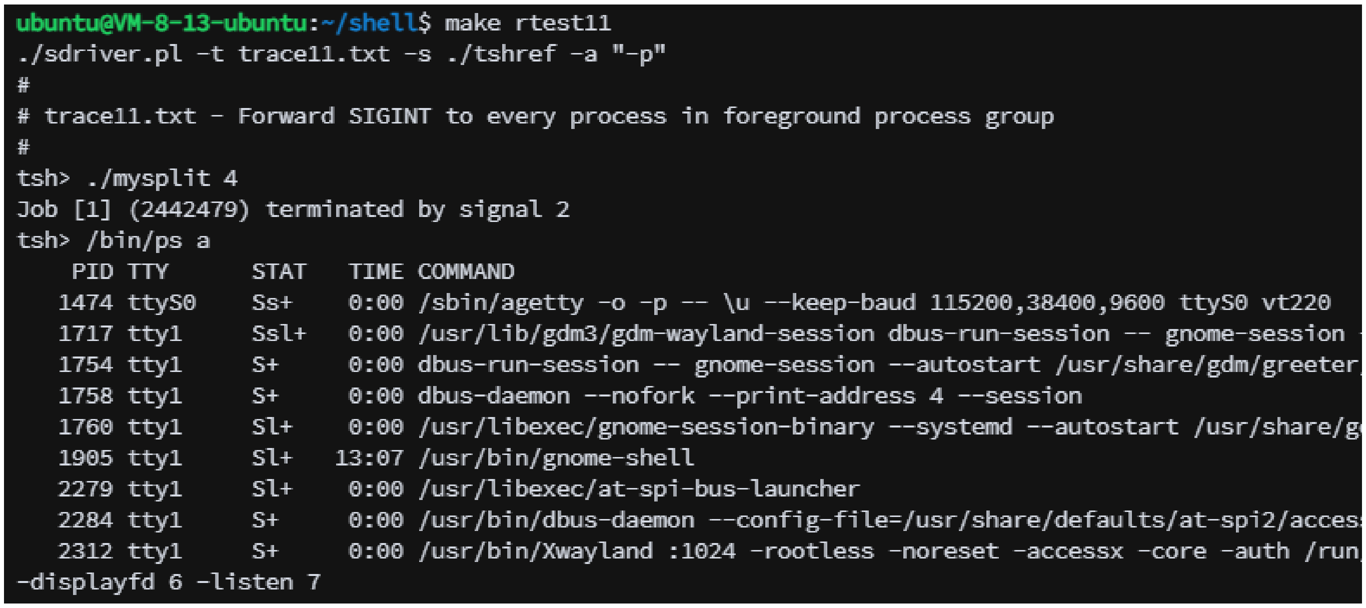
trace11

```
#
# 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
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ 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] (2442439) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1474 ttyS0    Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0 vt220
   1717 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-wayland-session dbus-run-session -- gnome-session --autostart
   1754 tty1     S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter/autostart
   1758 tty1     S+     0:00 dbus-daemon --nofork --print-address 4 --session
   1760 tty1     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/gdm/greeter/
   1905 tty1     Sl+   13:07 /usr/bin/gnome-shell
   2279 tty1     Sl+    0:00 /usr/libexec/at-spi-bus-launcher
   2284 tty1     S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.co
   2312 tty1     S+     0:00 /usr/bin/Xwayland :1024 -rootless -noreset -accessx -core -auth /run/user/128/.
-displayfd 6 -listen 7
   2477 tty1     Sl+    0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications
   2478 tty1     Sl+    0:00 /usr/libexec/at-spi2-registryd --use-gnome-session
```

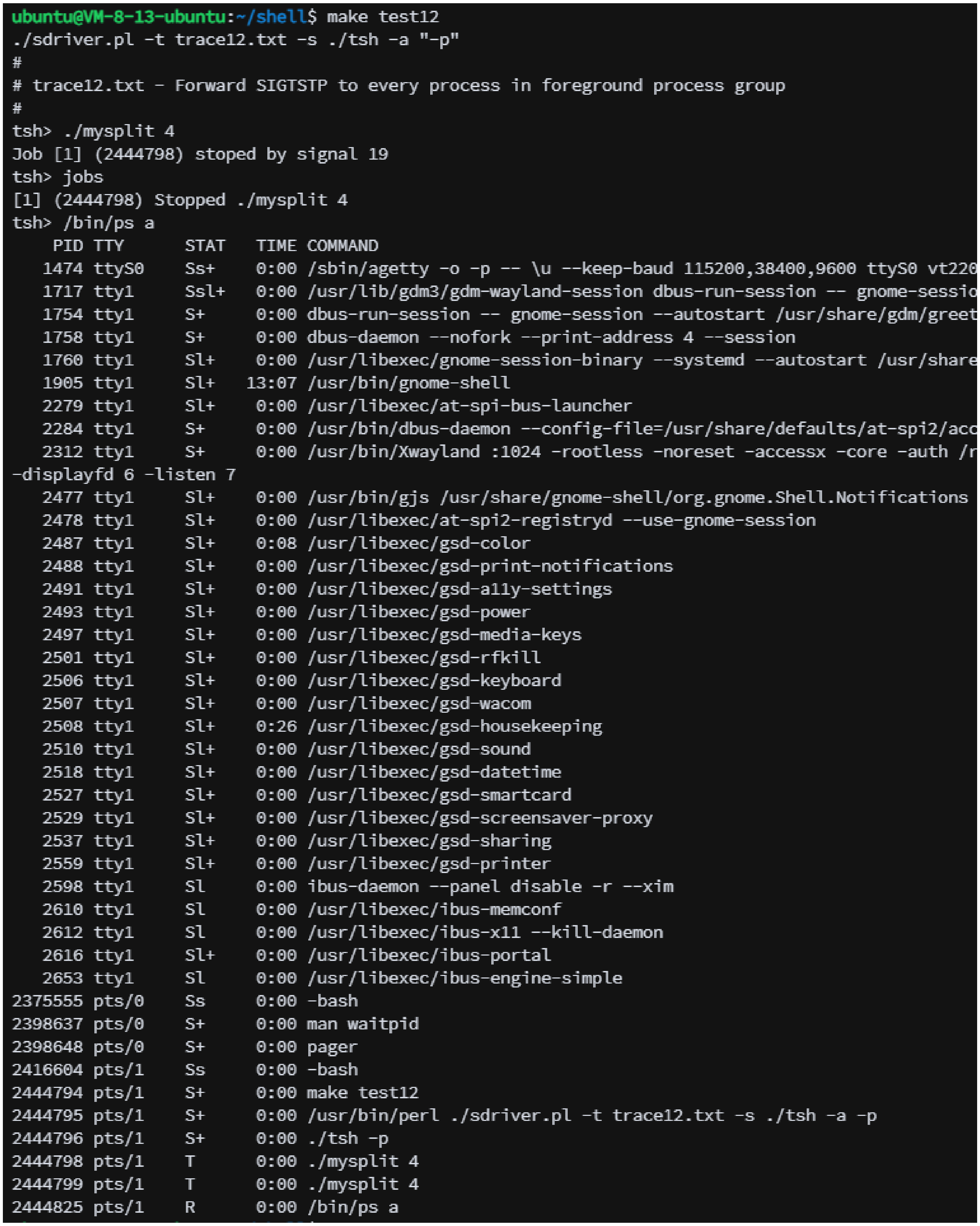


```
ubuntu@VM-8-13-ubuntu:~/shell$ make rtest11
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (2442479) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1474 ttyS0    Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0 vt220
   1717 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-wayland-session dbus-run-session -- gnome-session
   1754 tty1     S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greeter
   1758 tty1     S+     0:00 dbus-daemon --nofork --print-address 4 --session
   1760 tty1     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share/g
   1905 tty1     Sl+   13:07 /usr/bin/gnome-shell
   2279 tty1     Sl+    0:00 /usr/libexec/at-spi-bus-launcher
   2284 tty1     S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/acces
   2312 tty1     S+     0:00 /usr/bin/Xwayland :1024 -rootless -noreset -accessx -core -auth /run
-displayfd 6 -listen 7
```

trace12

```
#
# 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
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ 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] (2444798) stoped by signal 19
tsh> jobs
[1] (2444798) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
   1474 ttyS0    Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 ttyS0 vt220
   1717 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-wayland-session dbus-run-session -- gnome-sessio
   1754 tty1     S+     0:00 dbus-run-session -- gnome-session --autostart /usr/share/gdm/greet
   1758 tty1     S+     0:00 dbus-daemon --nofork --print-address 4 --session
   1760 tty1     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --autostart /usr/share
   1905 tty1     Sl+   13:07 /usr/bin/gnome-shell
   2279 tty1     Sl+    0:00 /usr/libexec/at-spi-bus-launcher
   2284 tty1     S+     0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/acc
   2312 tty1     S+     0:00 /usr/bin/Xwayland :1024 -rootless -noreset -accessx -core -auth /r
-displayfd 6 -listen 7
   2477 tty1     Sl+    0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications
   2478 tty1     Sl+    0:00 /usr/libexec/at-spi2-registryd --use-gnome-session
   2487 tty1     Sl+    0:08 /usr/libexec/gsd-color
   2488 tty1     Sl+    0:00 /usr/libexec/gsd-print-notifications
   2491 tty1     Sl+    0:00 /usr/libexec/gsd-a11y-settings
   2493 tty1     Sl+    0:00 /usr/libexec/gsd-power
   2497 tty1     Sl+    0:00 /usr/libexec/gsd-media-keys
   2501 tty1     Sl+    0:00 /usr/libexec/gsd-rfkill
   2506 tty1     Sl+    0:00 /usr/libexec/gsd-keyboard
   2507 tty1     Sl+    0:00 /usr/libexec/gsd-wacom
   2508 tty1     Sl+    0:26 /usr/libexec/gsd-housekeeping
   2510 tty1     Sl+    0:00 /usr/libexec/gsd-sound
   2518 tty1     Sl+    0:00 /usr/libexec/gsd-datetime
   2527 tty1     Sl+    0:00 /usr/libexec/gsd-smartcard
   2529 tty1     Sl+    0:00 /usr/libexec/gsd-screensaver-proxy
   2537 tty1     Sl+    0:00 /usr/libexec/gsd-sharing
   2559 tty1     Sl+    0:00 /usr/libexec/gsd-printer
   2598 tty1     Sl     0:00 ibus-daemon --panel disable -r --xim
   2610 tty1     Sl     0:00 /usr/libexec/ibus-memconf
   2612 tty1     Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
   2616 tty1     Sl+    0:00 /usr/libexec/ibus-portal
   2653 tty1     Sl     0:00 /usr/libexec/ibus-engine-simple
2375555 pts/0    Ss     0:00 -bash
2398637 pts/0    S+     0:00 man waitpid
2398648 pts/0    S+     0:00 pager
2416604 pts/1    Ss     0:00 -bash
2444794 pts/1    S+     0:00 make test12
2444795 pts/1    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
2444796 pts/1    S+     0:00 ./tsh -p
2444798 pts/1    T      0:00 ./mysplit 4
2444799 pts/1    T      0:00 ./mysplit 4
2444825 pts/1    R      0:00 /bin/ps a
```

.mysplit4的父子进程都被终端暂停（TSTP）

trace13

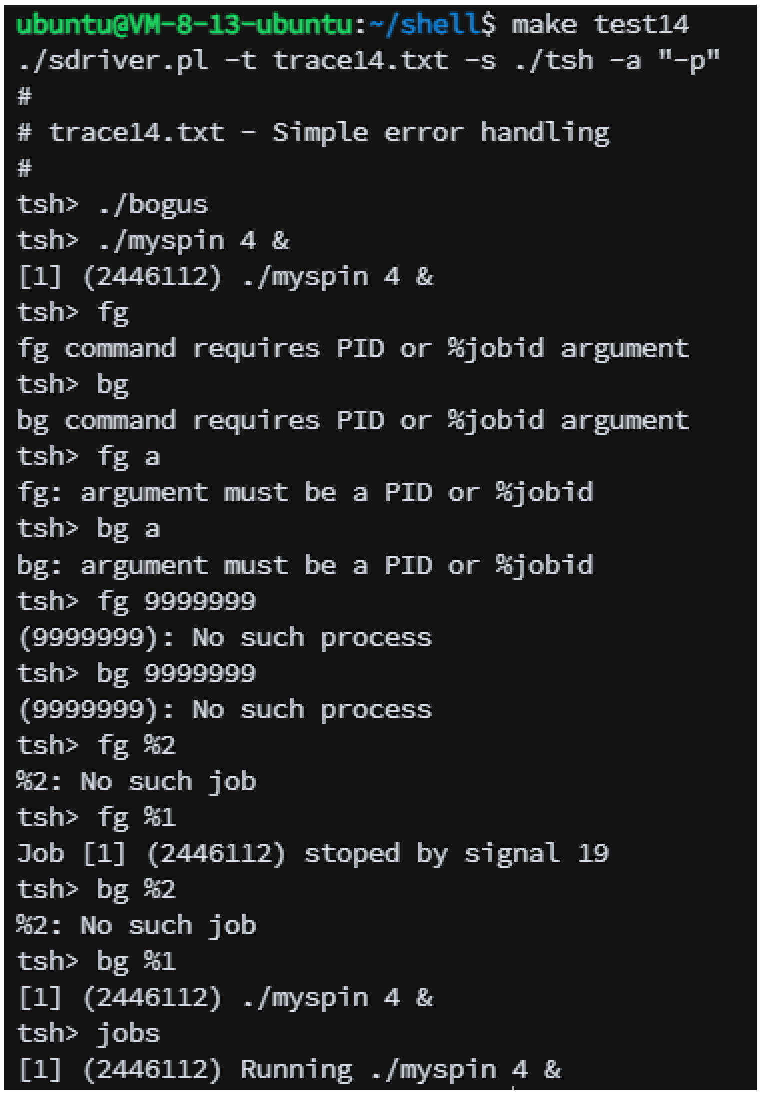
```
#
# trace13.txt - Restart every stopped process in process group
#
/bin/echo -e tsh> ./mysplit 4
./mysplit 4

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
TLNS

trace14 Simple error handling

```
#
# 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
```

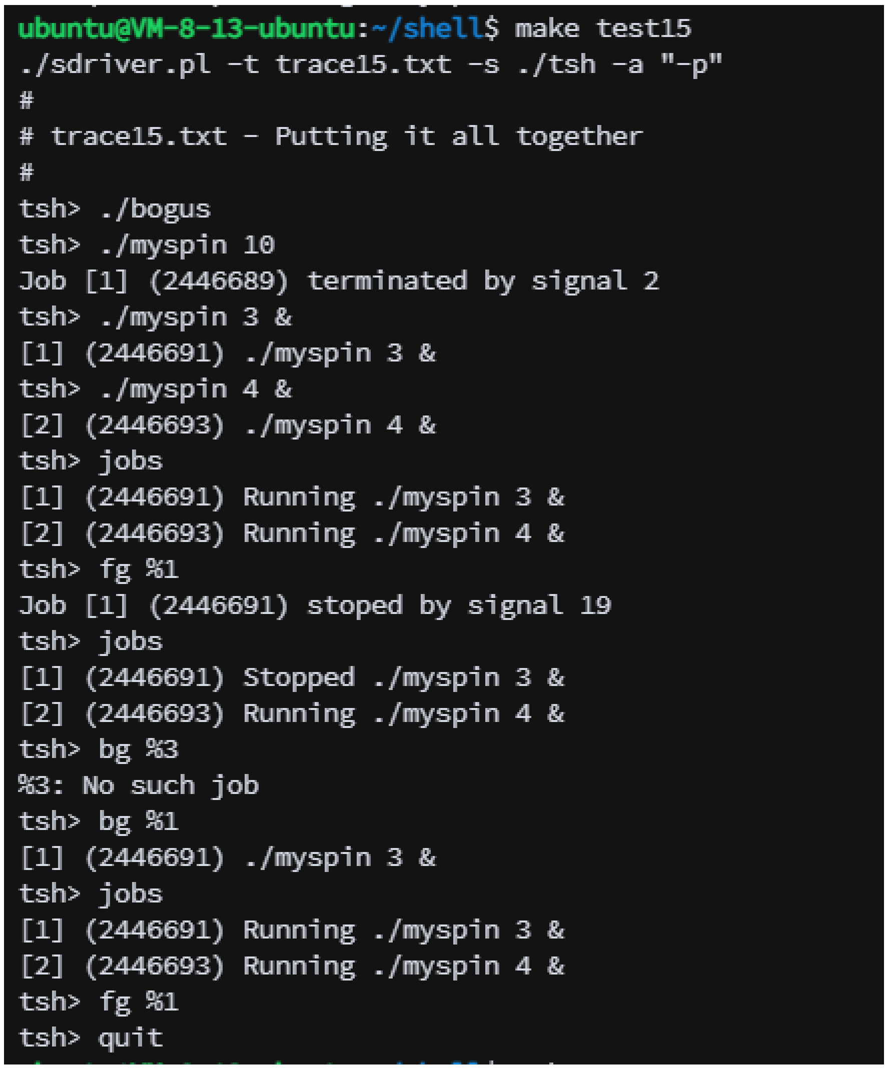
```
bg

/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
```


```
ubuntu@VM-8-13-ubuntu:~/shell$ make test14
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
tsh> ./myspin 4 &
[1] (2446112) ./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] (2446112) stoped by signal 19
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (2446112) ./myspin 4 &
tsh> jobs
[1] (2446112) Running ./myspin 4 &
```

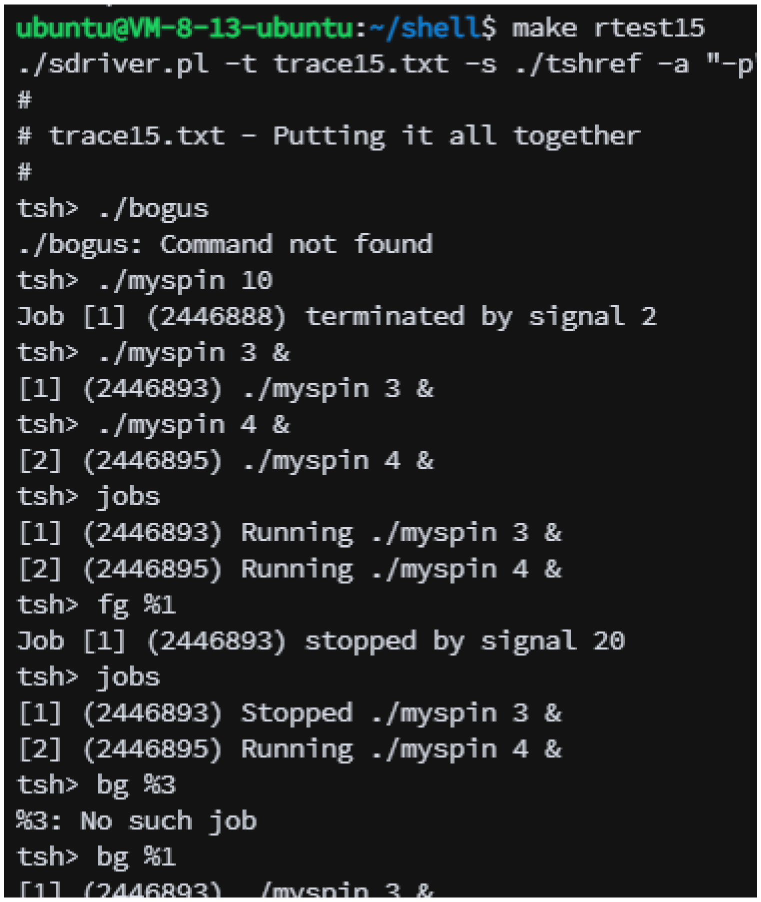
trace15 Putting it all together

```
#
# trace15.txt - Putting it all together
#

/bin/echo tsh> ./bogus
./bogus

/bin/echo tsh> ./myspin 10
./myspin 10

SLEEP 2
INT

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

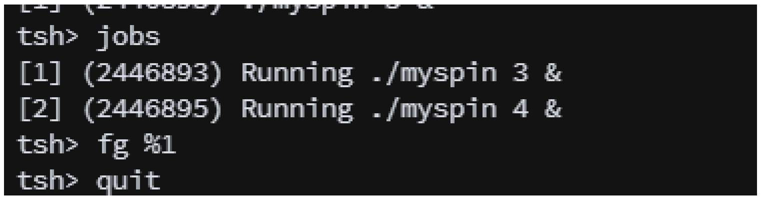
```
/bin/echo -e tsh> ./myspin 4 \046
./myspin 4 &

/bin/echo tsh> jobs
jobs

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

SLEEP 2
TSTP

/bin/echo tsh> jobs
jobs

/bin/echo tsh> bg %3
bg %3

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

/bin/echo tsh> jobs
jobs

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

/bin/echo tsh> quit
quit
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
tsh> ./myspin 10
Job [1] (2446689) terminated by signal 2
tsh> ./myspin 3 &
[1] (2446691) ./myspin 3 &
tsh> ./myspin 4 &
[2] (2446693) ./myspin 4 &
tsh> jobs
[1] (2446691) Running ./myspin 3 &
[2] (2446693) Running ./myspin 4 &
tsh> fg %1
Job [1] (2446691) stoped by signal 19
tsh> jobs
[1] (2446691) Stopped ./myspin 3 &
[2] (2446693) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (2446691) ./myspin 3 &
tsh> jobs
[1] (2446691) Running ./myspin 3 &
[2] (2446693) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```



```
ubuntu@VM-8-13-ubuntu:~/shell$ 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] (2446888) terminated by signal 2
tsh> ./myspin 3 &
[1] (2446893) ./myspin 3 &
tsh> ./myspin 4 &
[2] (2446895) ./myspin 4 &
tsh> jobs
[1] (2446893) Running ./myspin 3 &
[2] (2446895) Running ./myspin 4 &
tsh> fg %1
Job [1] (2446893) stopped by signal 20
tsh> jobs
[1] (2446893) Stopped ./myspin 3 &
[2] (2446895) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (2446893) ./myspin 3 &
```

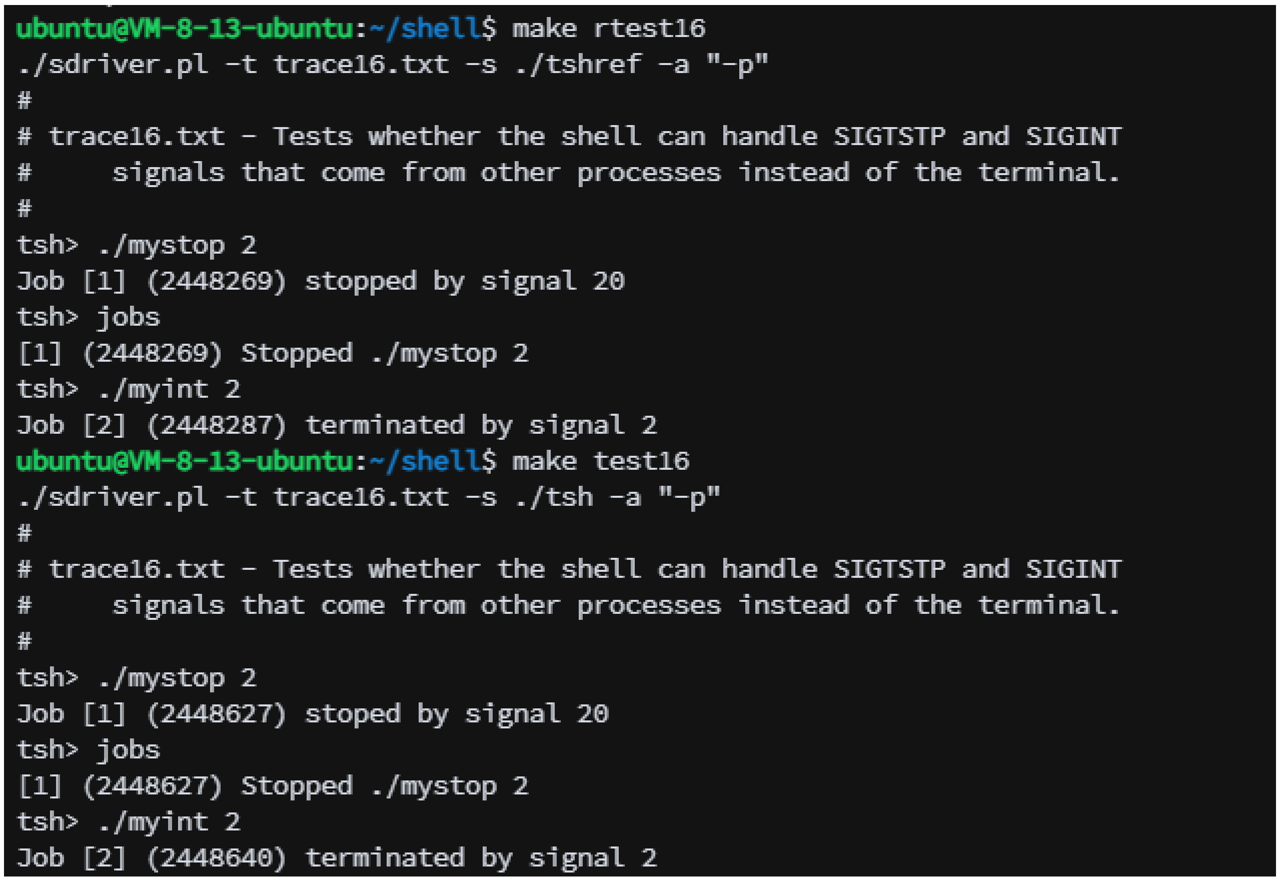


```
tsh> jobs
[1] (2446893) Running ./myspin 3 &
[2] (2446895) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

trace16

```
# 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
```

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



```
ubuntu@VM-8-13-ubuntu:~/shell$ 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] (2448269) stopped by signal 20
tsh> jobs
[1] (2448269) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (2448287) terminated by signal 2
ubuntu@VM-8-13-ubuntu:~/shell$ 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] (2448627) stoped by signal 20
tsh> jobs
[1] (2448627) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (2448640) terminated by signal 2
```

参考