**Shell Lab**

16307130194 陈中钰

16级 计算机科学技术学院

Contents

1. Preparation 2
2. trace01 2
3. trace02 2
4. trace03 3
5. trace04 3
6. trace05 5
7. trace06 5
8. trace07 6
9. trace08 6
10. trace09 & trace10 7
11. trace11 & trace12 & trace13 8
12. trace14 10
13. trace15 11
14. trace16 11
15. Thoughts 12
16. **Preparation**
    1. 阅读材料

《深入理解计算机系统》第8章 异常控制流，shlab.pdf

* 1. 实验内容：完成一个tiny shell

利用已学知识、已给出的函数和库函数，填充tsh.c中7个函数，make编译，make test运行对应的trace文件，并与make rtest结果进行对比，结果除了pid之外必须一模一样。

* 1. Shell的架构

Shell调用main()函数，循环往复地接收指令cmdline，并调用eval(cmdline)对命令进行解释，与另外6个函数一同来实现cmdline，除此以外还包括一些报错的机制。

* 1. 实验环境

VMware Workstation 11.0.0；ubuntu 14.04

* 1. 注意事项
     1. 在signal handler中用kill函数发送信号时，用-pid代替pid
     2. 在waitfg中使用sigsuspend，而在sigchld\_handler中仅使用一次waitpid
     3. 在fork之后、execve之前，调用setpgid(0, 0)分离进程
  2. 实验过程

阅读trace内容，make rtest查看结果

填充相关函数，

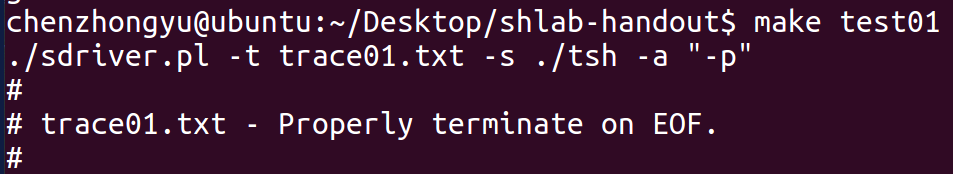
make编译

make test验证，

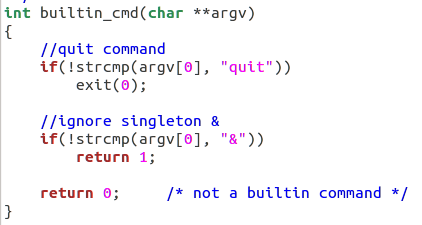
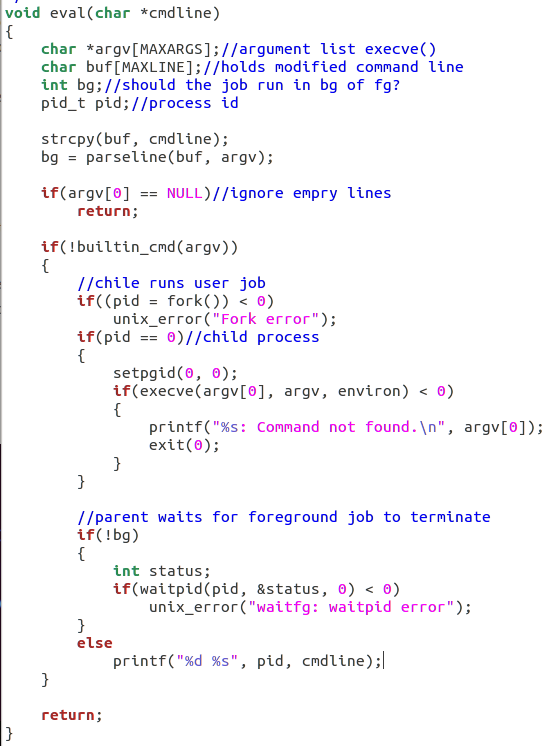
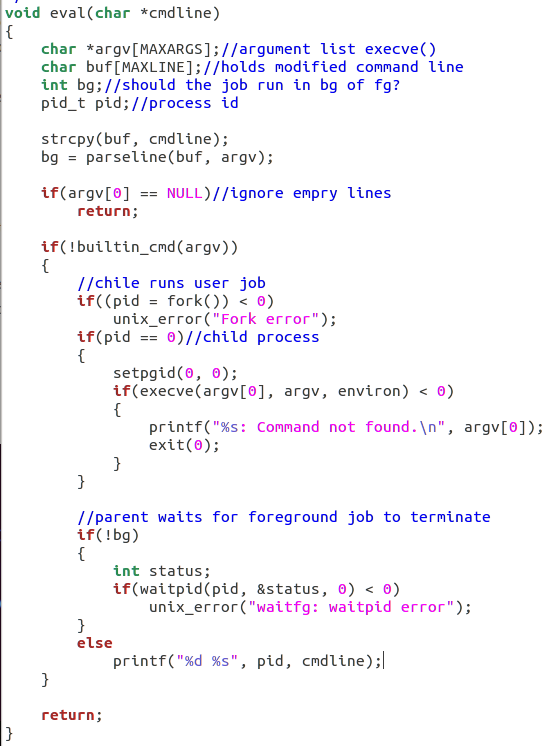
修正相关函数

1. **trace01**

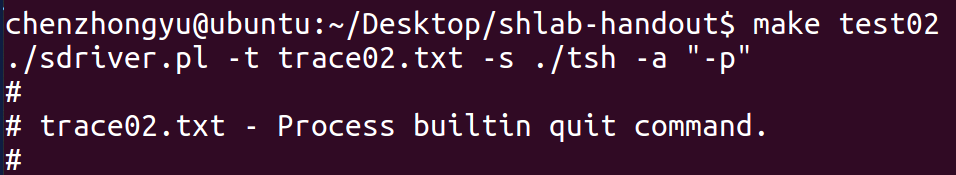
EOF功能本身就实现好了，直接运行就成功了。



1. **trace02**

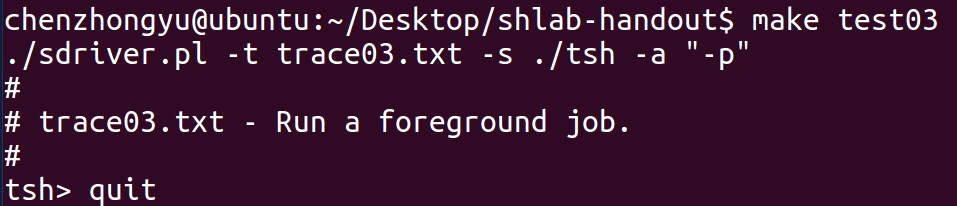
处理builtin quit command，首先要填充处理任何命令都会用的eval()函数，并在调用到的builtin\_cmd()函数中加入对quit命令的处理。而eval()和builtin\_cmd()函数都是仿照书中代码写的。此外，按照pdf要求，还需要在fork()和execve()之间使用setpgid(0, 0)来分离进程。

make test运行，成功。



1. **trace03**

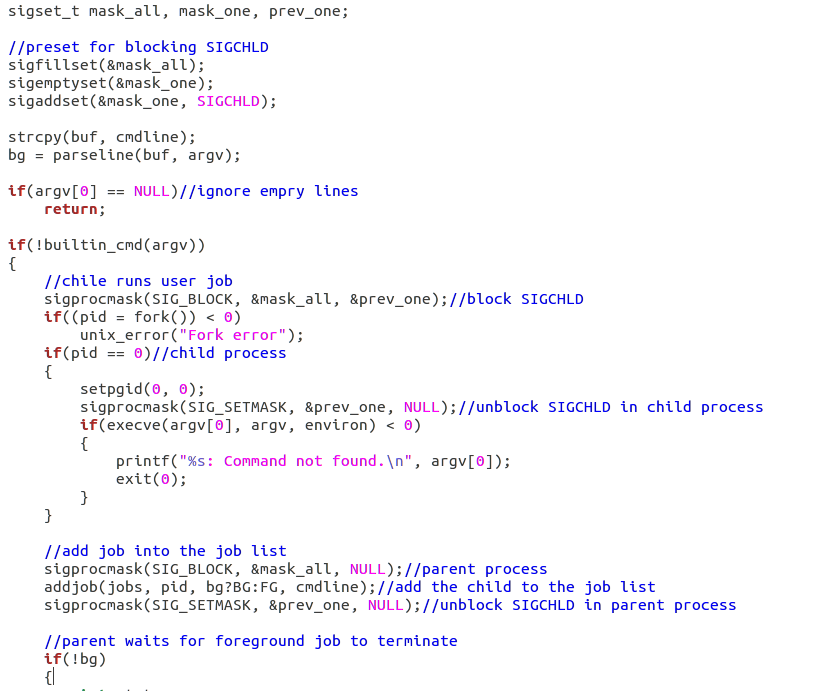
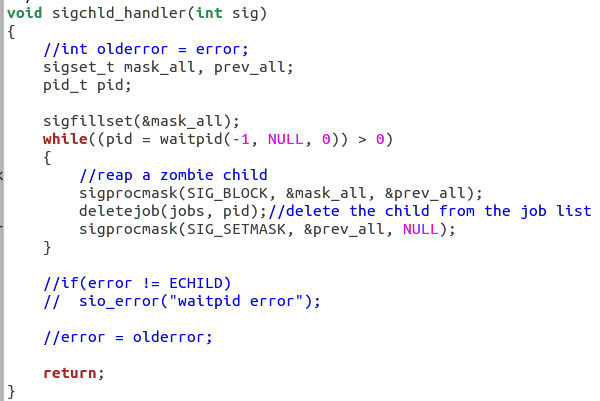
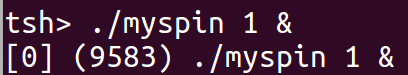
调用echo函数输出内容，并输入quit命令退出shell。而处理非builtin command的机制已经做好了，quit命令也已经做好了。直接make test运行，成功。

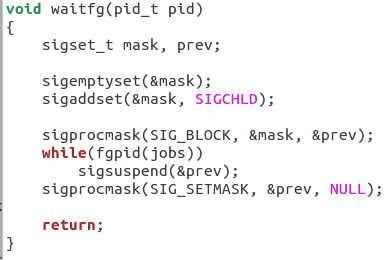
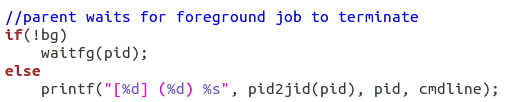


1. **trace04**

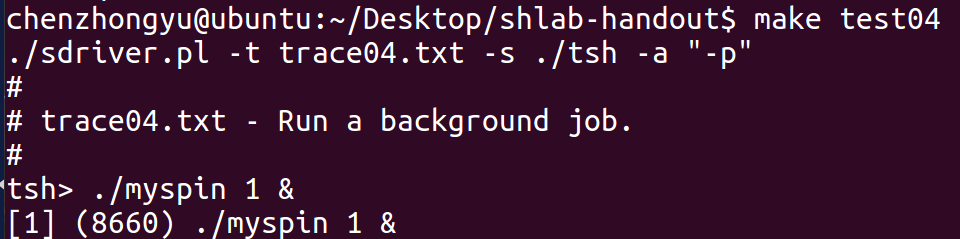
运行一个后台程序。先直接运行，发现后台程序job描述的输出格式不一致，故修改输出格式，

C:\Users\ECHOES\AppData\Local\Temp\1523070020(1).png

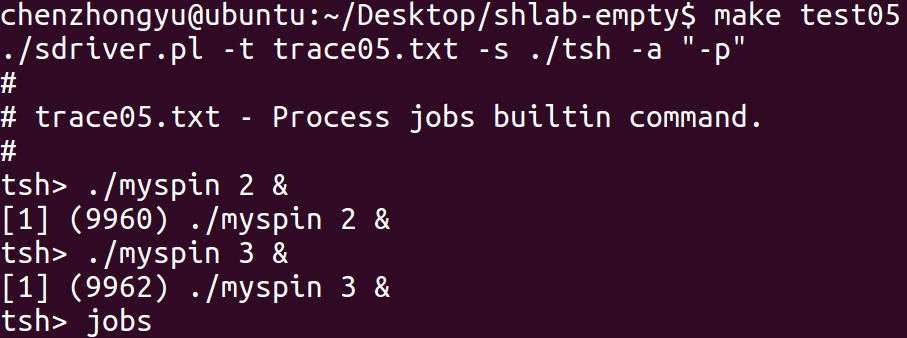
再次运行，对比发现不同的地方只有[]中的0应该为1，分析可见，eval()函数中，父进程需要在运行后台程序后addjob，当程序收到SIGCHLD信号后，还需要进行deletejob。在eval()中添加addjob操作，在sigchld\_handler()中添加deletejob操作，并注意恰当地阻塞SIGCHLD信号，避免addjob和deletejob之间的race问题，以及父进程和子进程都要分别解除阻塞信号。

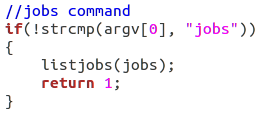
再次make test验证，发现[]中本应为1却变成2，分析可得未等待echo前台程序结束，就运行了后台程序，所以jid为2。那么还需要添加waitfg()函数，等待前台函数结束才能输入命令行，进而运行后台程序。

再次make test验证，成功。



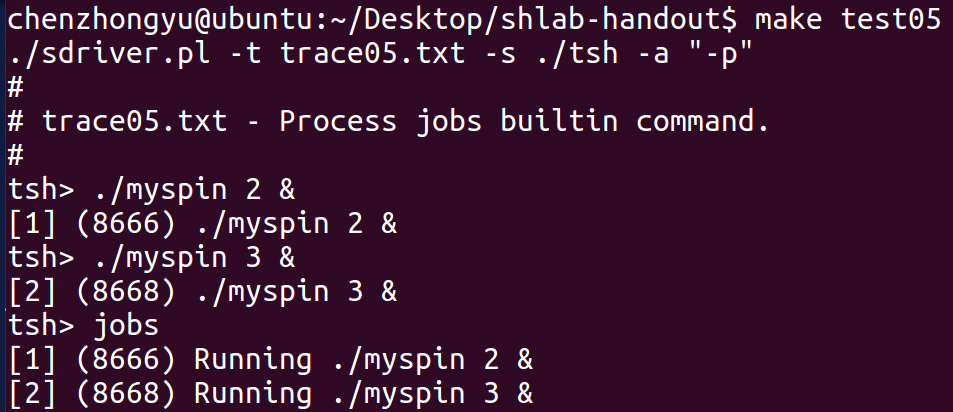
1. **trace05**

make rtest查看，需要实现jobs的内置命令处理。在builtin\_cmd()函数中添加jobs命令，利用listjobs()函数打印jobs列表。运行并查看结果。

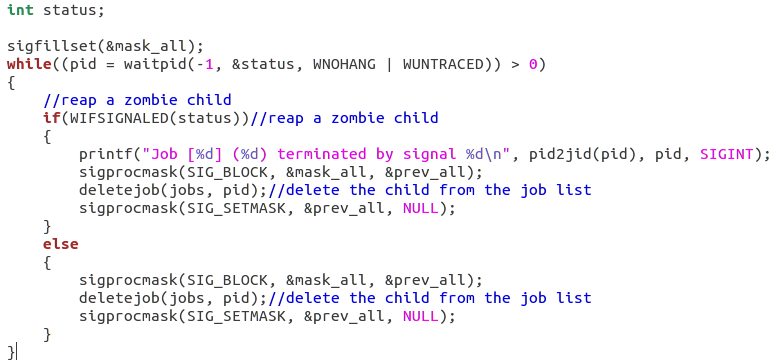
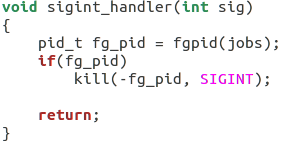
发现所有的jobs都结束了。说明while(waitpid)会等待所有进程都结束才会结束，而实际上只需要所有zombie进程都结束了就行了，故需修改options默认行为。

C:\Users\ECHOES\AppData\Local\Temp\1523071200(1).png

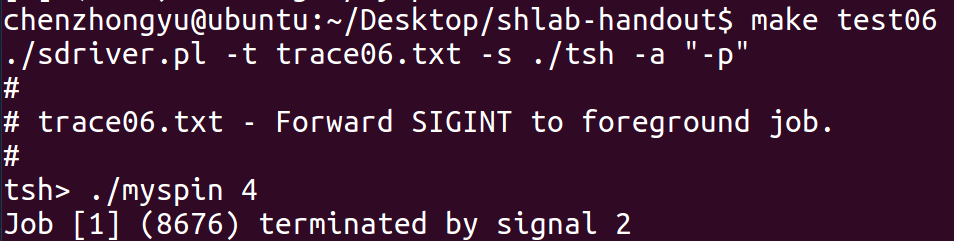
运行，成功。



1. **trace06**

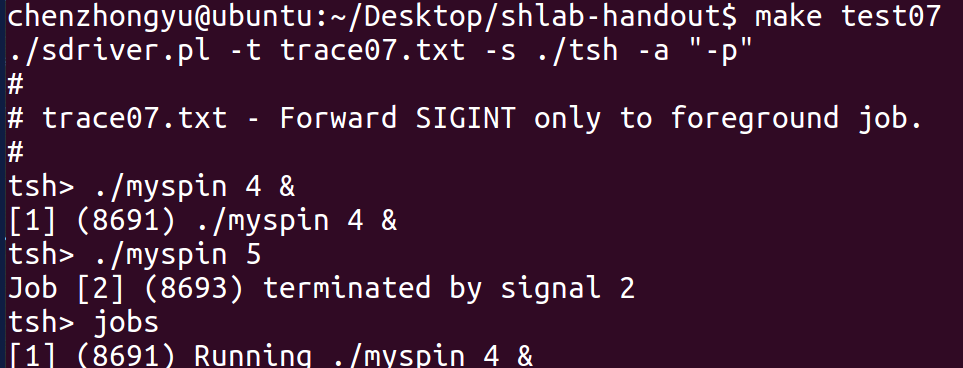
向前台程序发送SIGINT信号，并终止前台程序，还需要输出job收到SIGINT后终止的描述。那么需要填充sigint\_handler()函数来发送SIGINT，并在sigchld\_handler()函数deletejob的时候，用status以及相关的宏来判断退出状态，对收到SIGINT信号而终止的情况输出终止描述，而正常终止的情况不输出。

make test验证，成功。

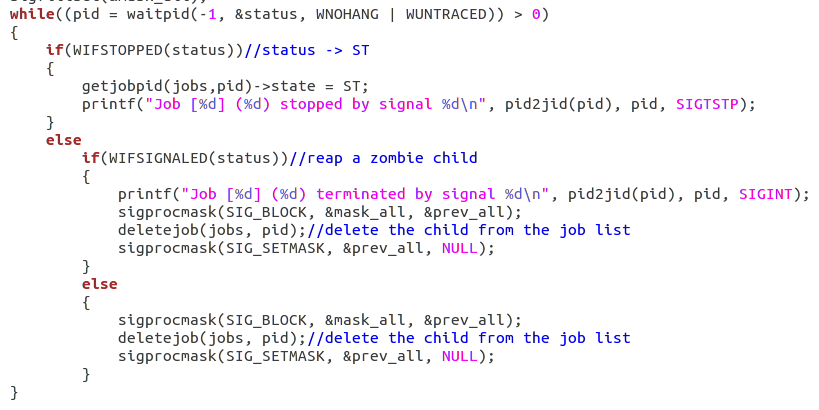
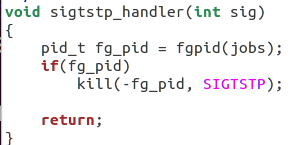


1. **trace07**

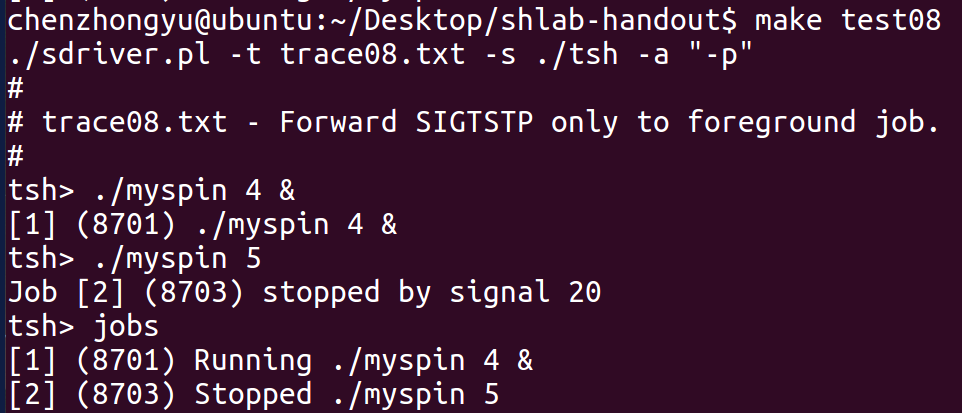
只对前台进程发送SIGINT。在sigint\_handler()中已经保证了。直接运行，成功。



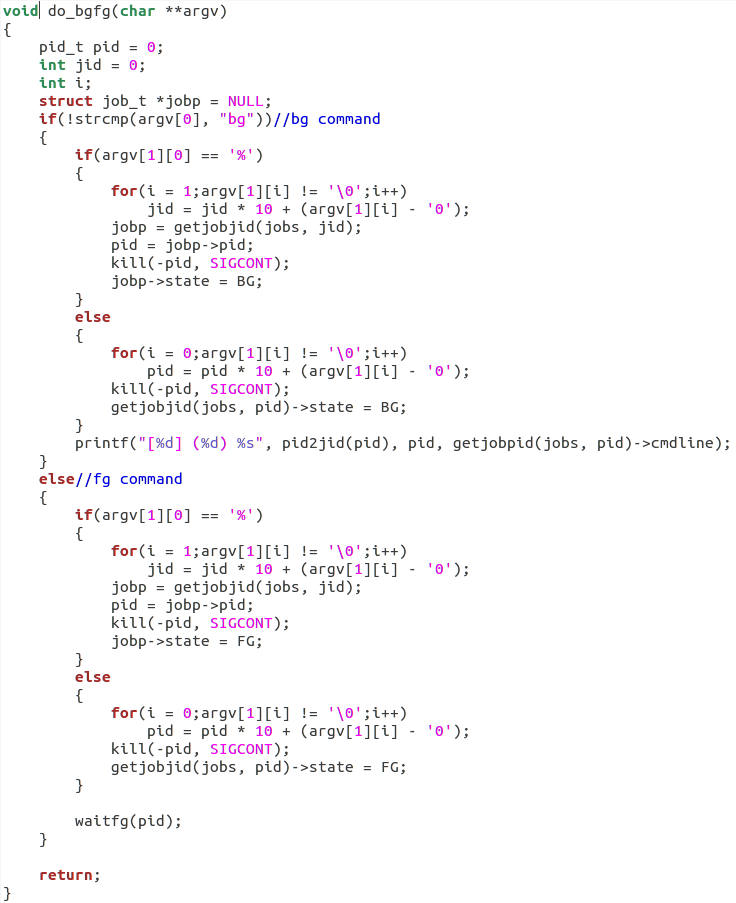
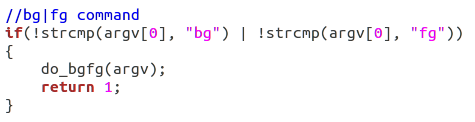
1. **trace08**

只对前台程序发送SIGTSTP。需要填充sigtstp\_handler()函数，只对前台程序发送SIGTSTP，并在sigchld\_handler()中通过status和对应的宏来判断由于收到SIGTSTP而停止的情况，输出stopped描述，并更改job的state。

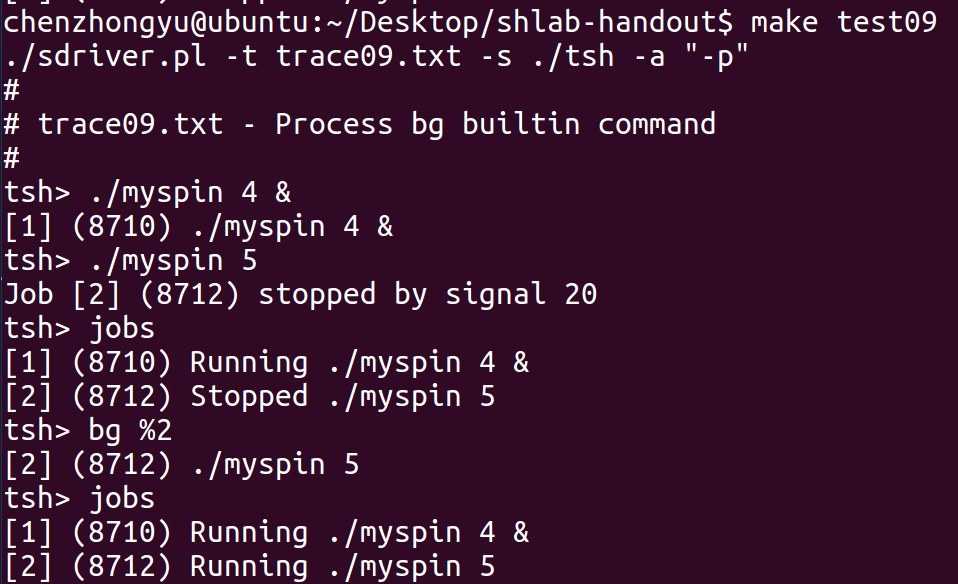
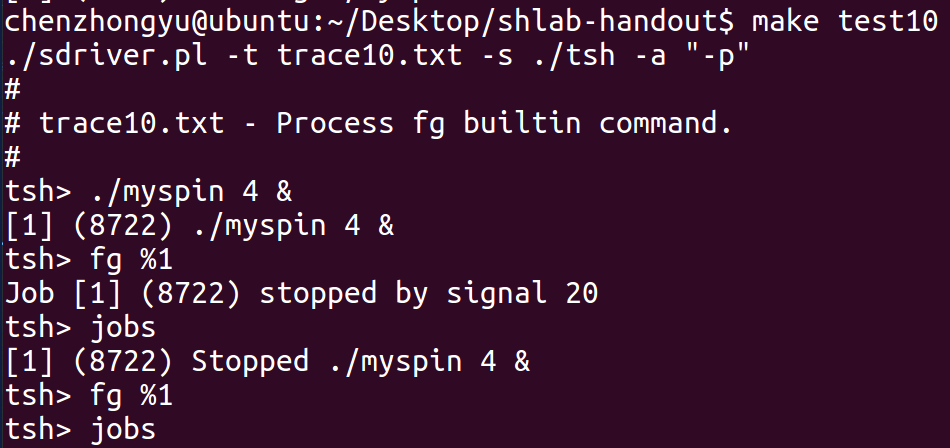
make test验证，成功。



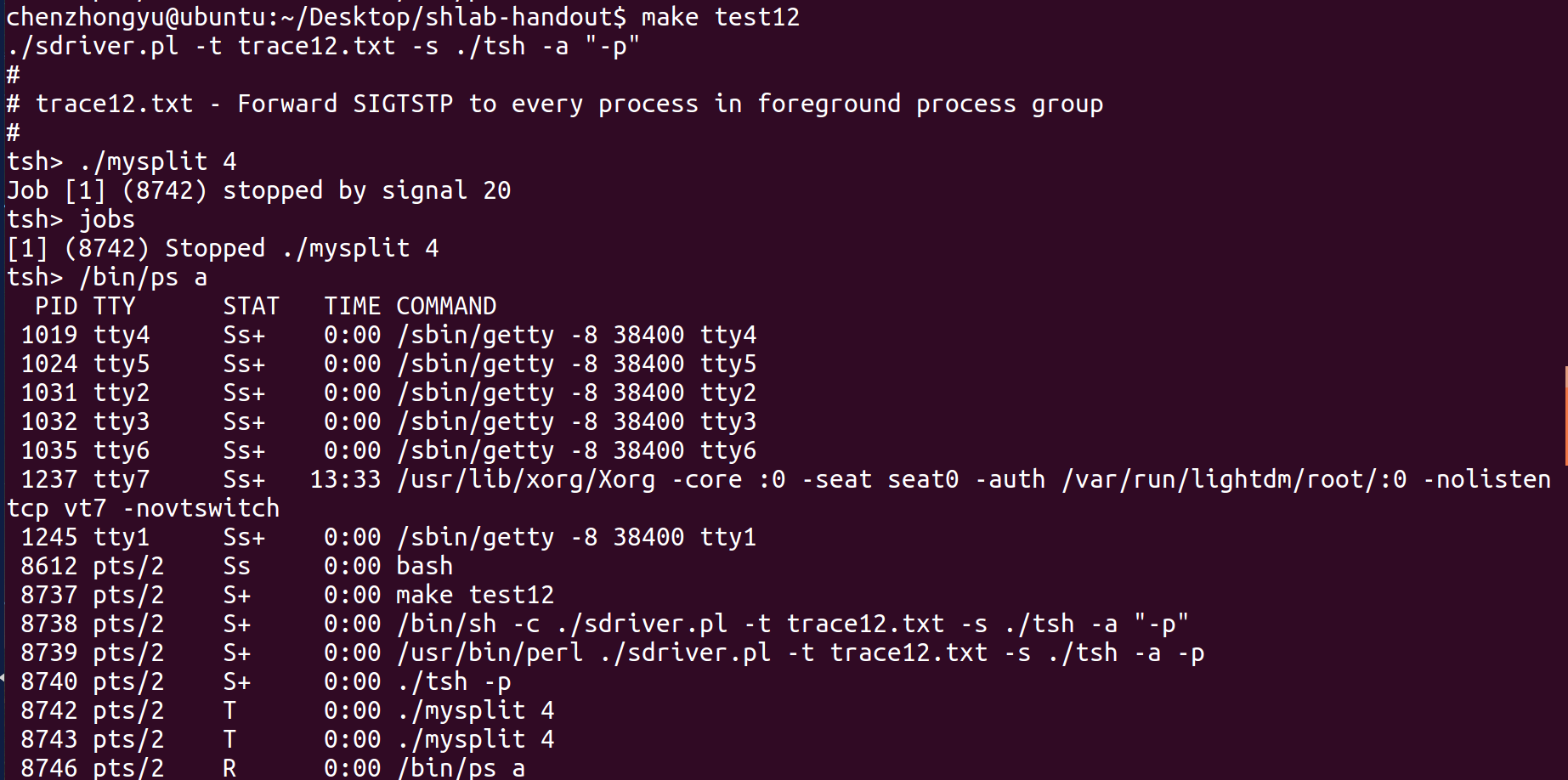
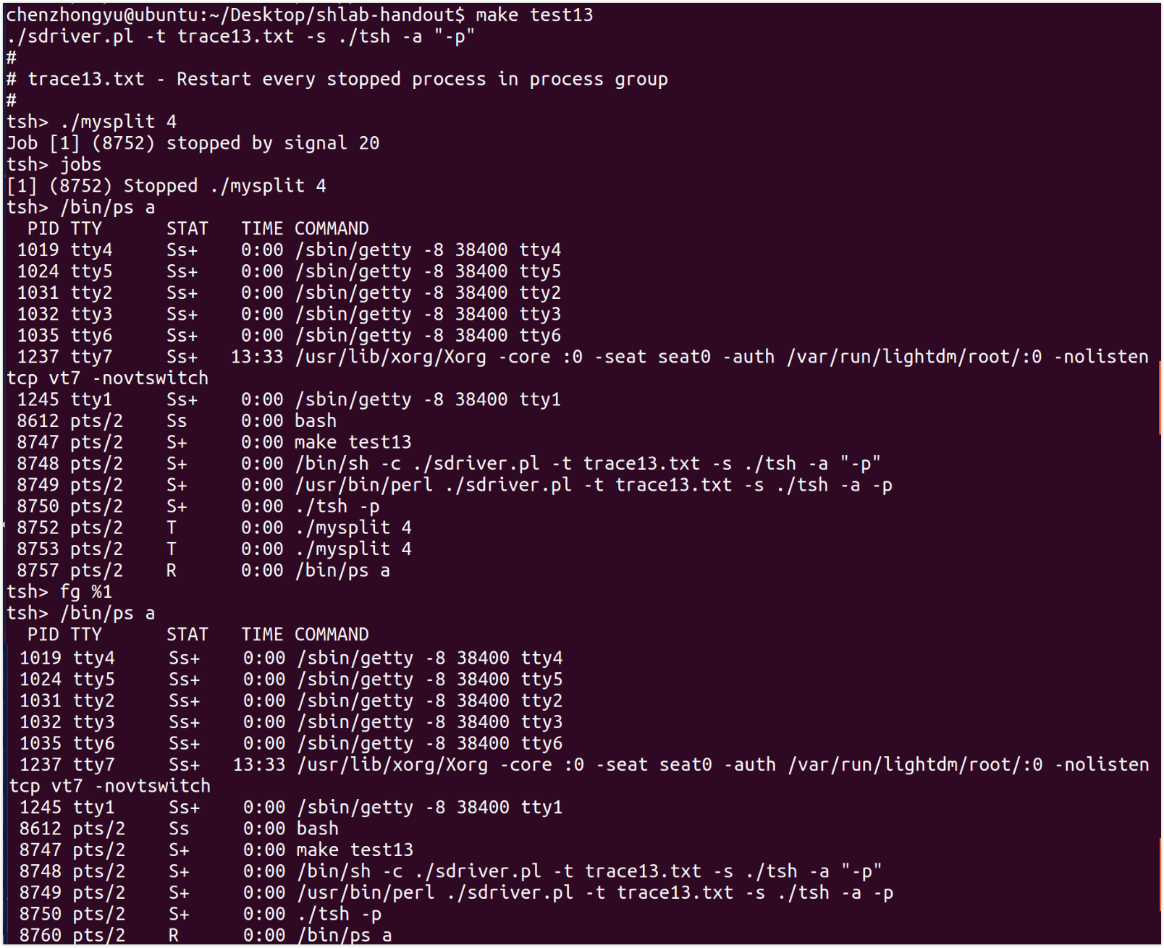
1. **trace09 & trace10**

处理bg与fg的builtin command。首先要在builtin\_cmd()中添加对bg和fg命令的识别，并调用do\_bgfg()函数来运行bg或fg命令，接着要填充do\_bgfg()函数。在do\_bgfg()函数中，首先要区分bg和fg命令，接着通过’%’区分输入的是pid还是jid，并从argv[1]中提取出pid或jid，如果是jid则还需要转换为pid。然后通过kill()发送SIGCONT给pid进程（注意：pdf中指示用-pid代替pid），并把state更改为对应的BG或FG。如果是bg命令，最后还需要按照格式输出job的描述。

make test验证，成功。



1. **trace11 & trace12 & trace13**
   1. trace11、12分别向前台进程组中的所有进程发送SIGINT、SIGTSTP。在之前的设计中已经完成了。直接make test验证，成功。
   2. trace13通过fg命令重启进程组中所有停止的进程。在之前的设计中已经完成了。直接make test验证，成功。



1. **trace14**

这是一组报错机制的实现。分为以下类型：

* 1. 错误指令

在eval()函数中的execve()，如果返回值<0，报错并exit(0)。

* 1. fg/bg + NULL

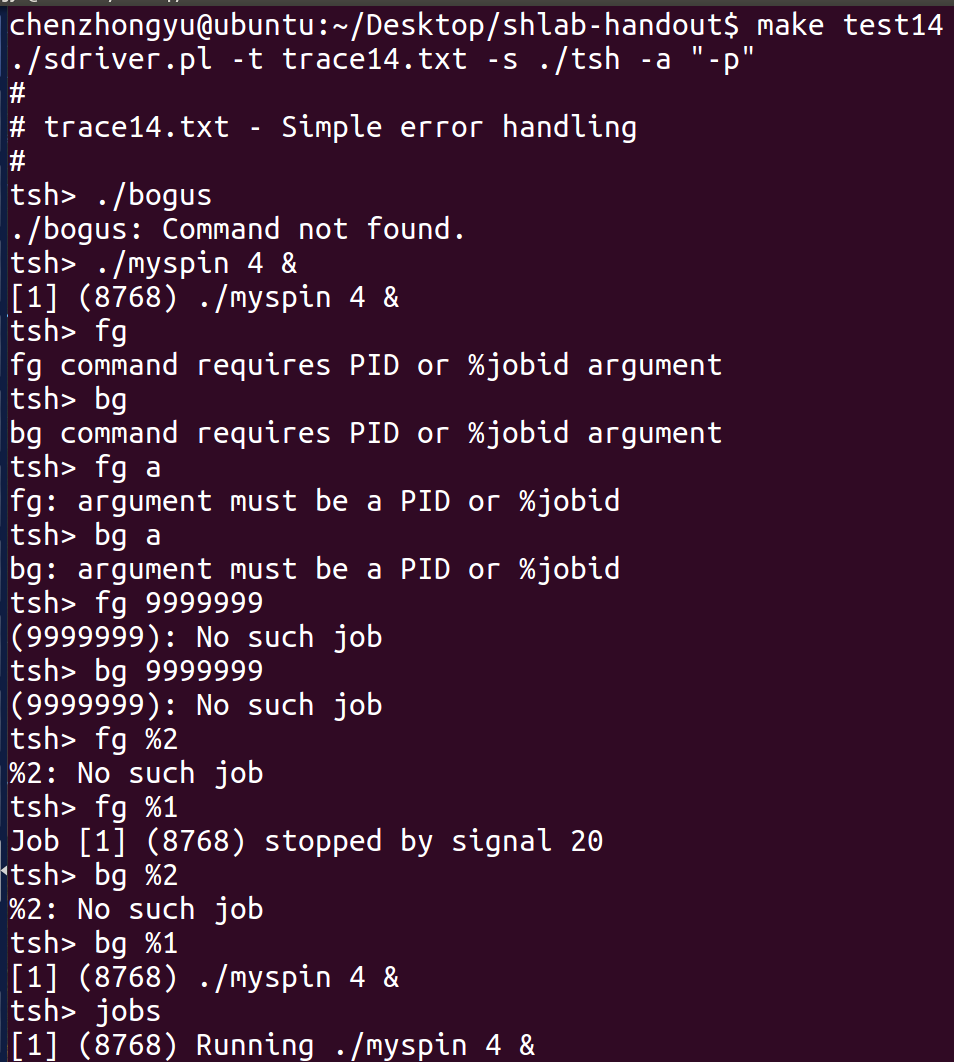
在do\_bgfg()中，如果argv[1]为NULL，报错并return。

* 1. fg/bg + 非数字

判断argv[1]是否是一个数字，如果其中任意一个字符不是0~9，报错并return。

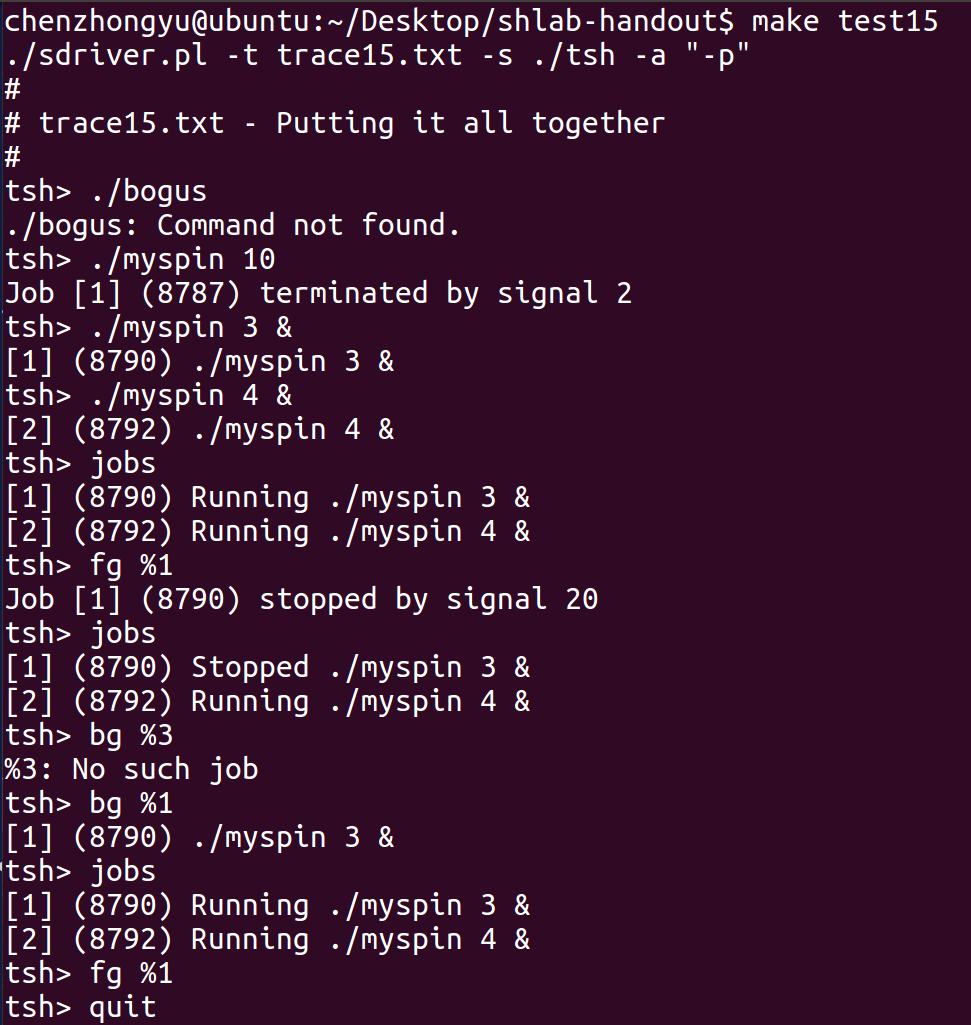
* 1. 进程不存在

如果getjobpid或getjobjid返回NULL，说明进程不存在，报错并return。

最后还需要注意报错格式对应make rtest14中的格式。以上报错机制只需对do\_bgfg()函数进行修改即可，而函数代码在tsh.h中已体现，此处不再贴出。最后make test验证，成功。

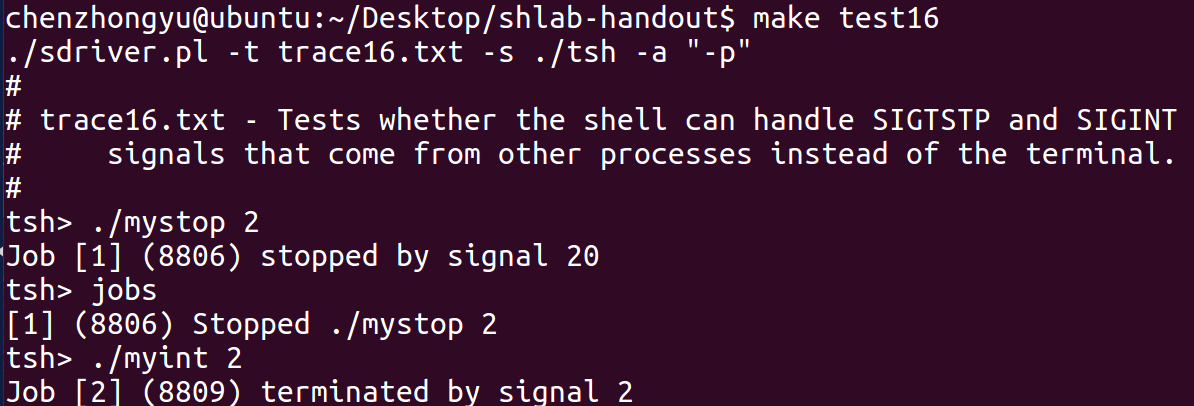
1. **trace15**

综合测试，直接make test15验证，成功。



1. **trace16**

测试shell是否能处理是其他进程而不是terminal发出的SIGTSTP和SIGINT信号。直接make test16验证，成功。



1. **Thoughts**
   1. 对进程控制、任务控制、信号机制等与异常控制流的相关的知识，有了更深的理解；
   2. 对Shell的运行机制更加明晰；
   3. 学习到书中没有涉及到的前后台程序的运行及操作机制；
   4. 加深对races的理解，并学会使用sigprocmask、sigsuspend等函数来避免races；
   5. 加深对waitpid函数参数的使用的理解；
   6. 为学习与并发编程相关的知识打下基础，加深对并发的理解。