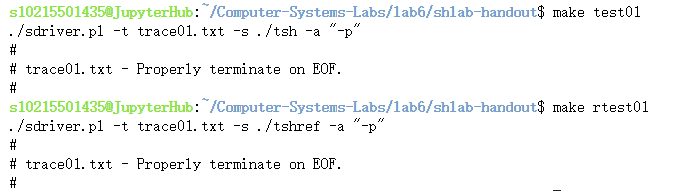
**Shell Lab 实验报告**

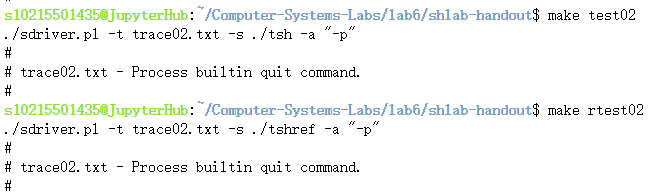
**——10215501435 杨茜雅**

检查16个trace文件：

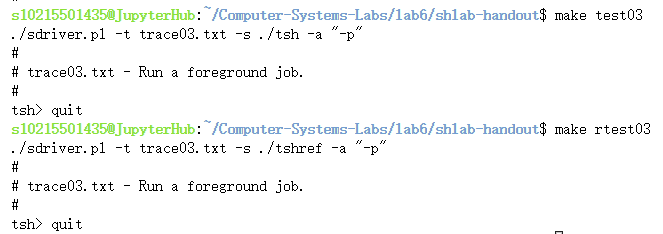
**Test01:**



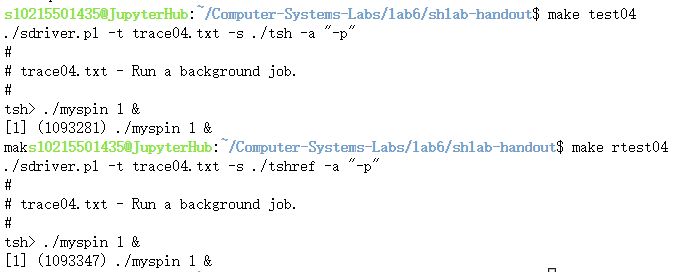
**Test02:**



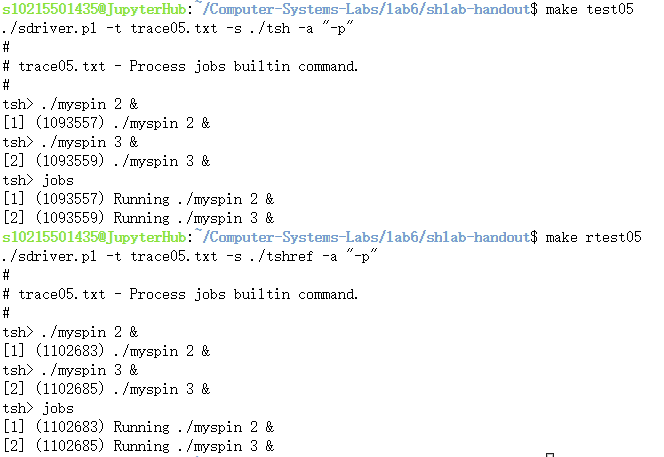
**Test03:**



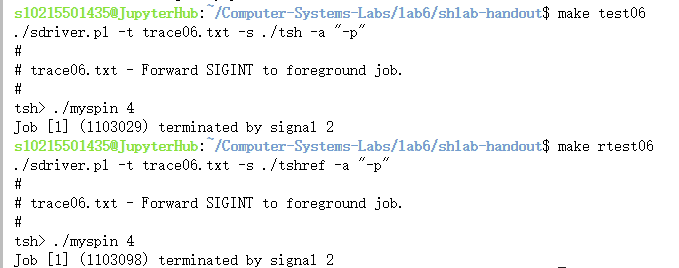
**Test04:**



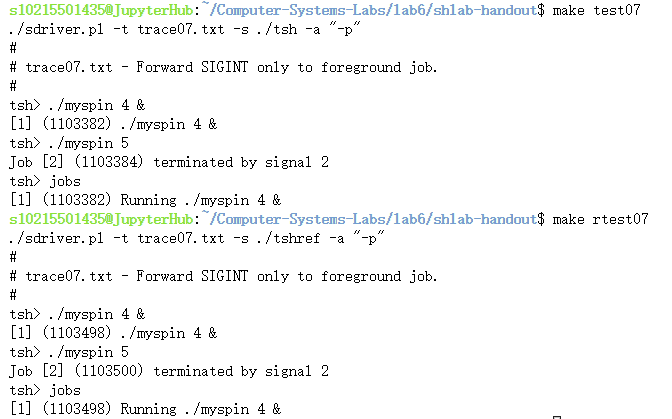
**Test05:**



**Test06:**



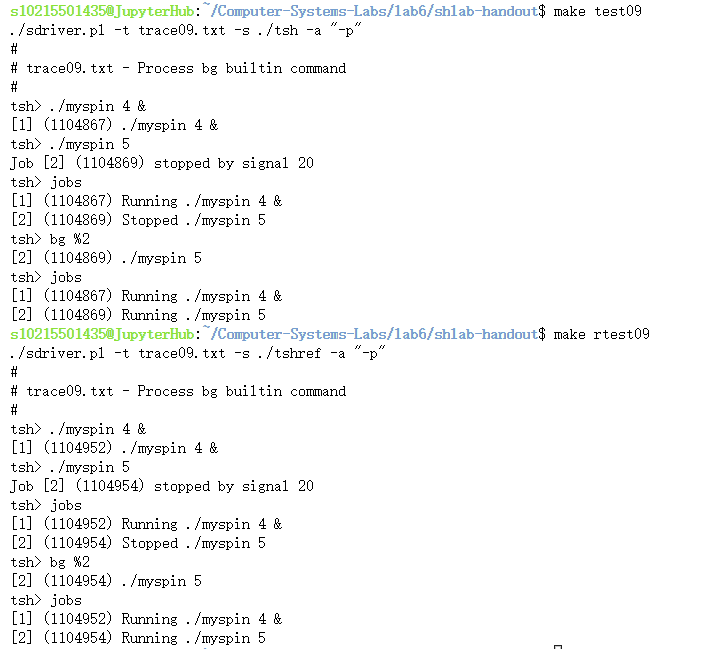
**Test07:**



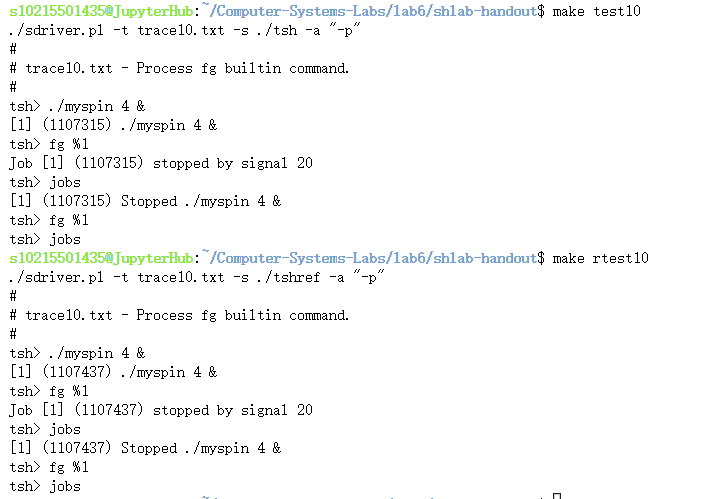
**Test08:**



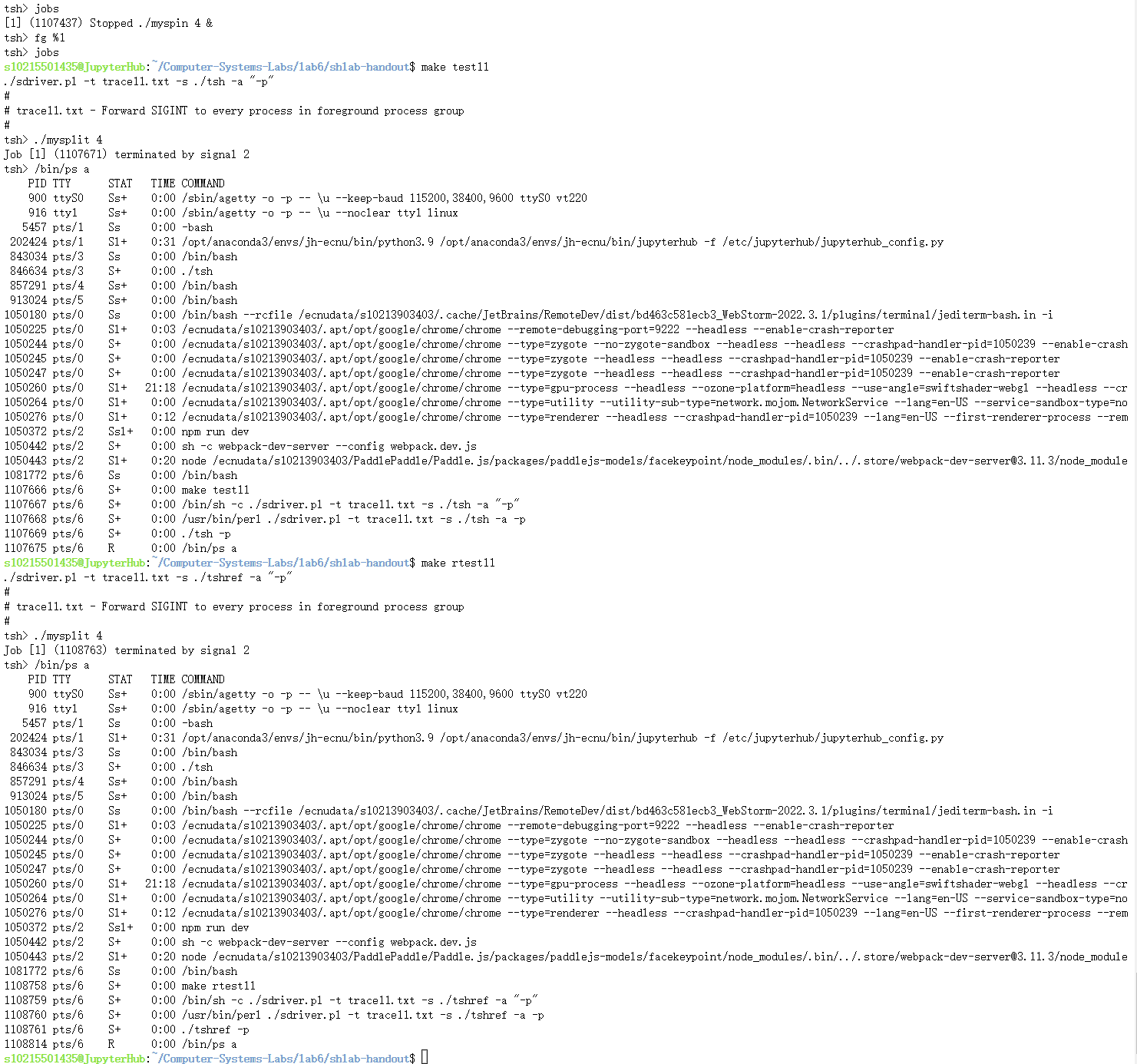
**Test09:**



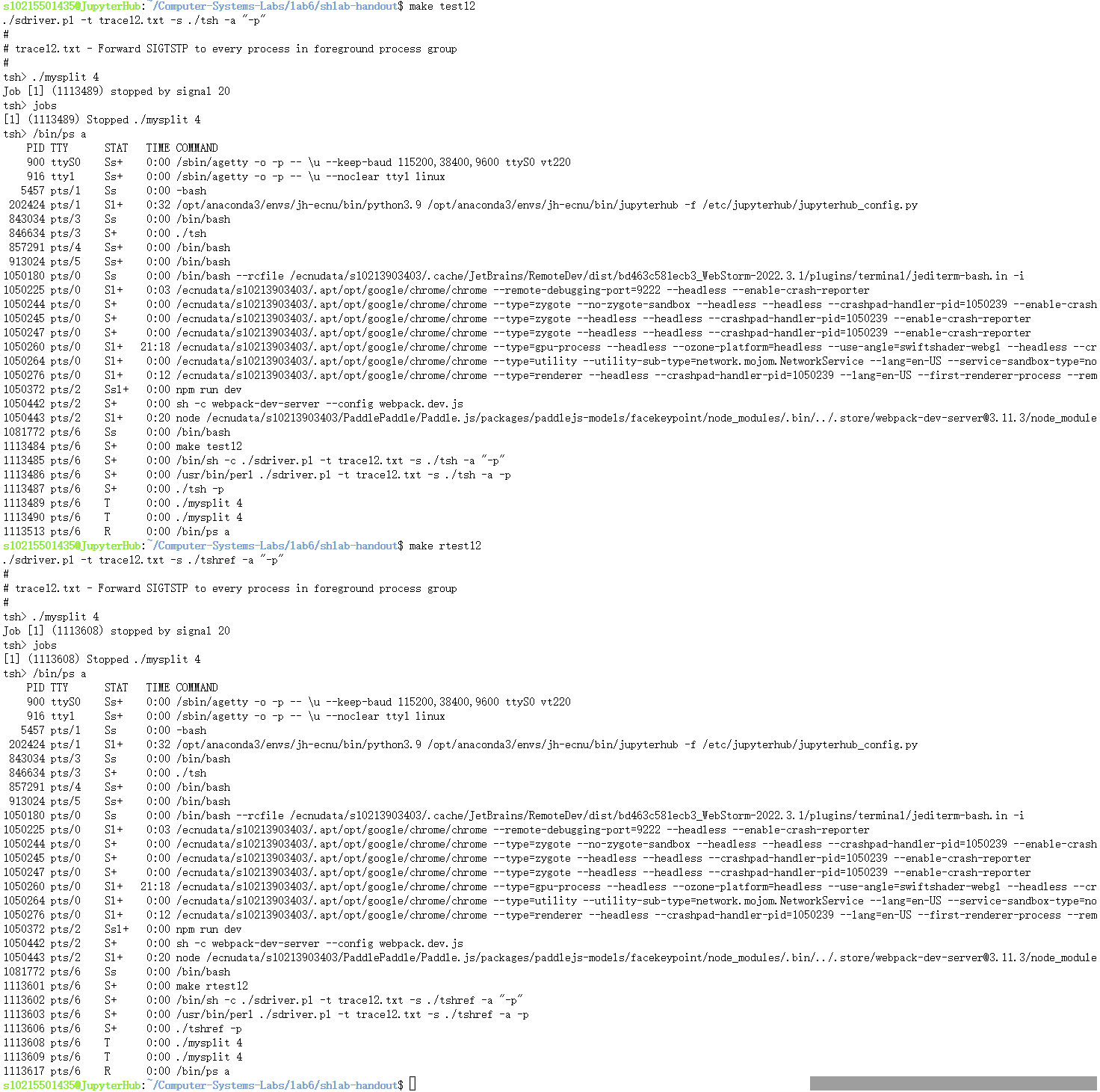
**Test10:**



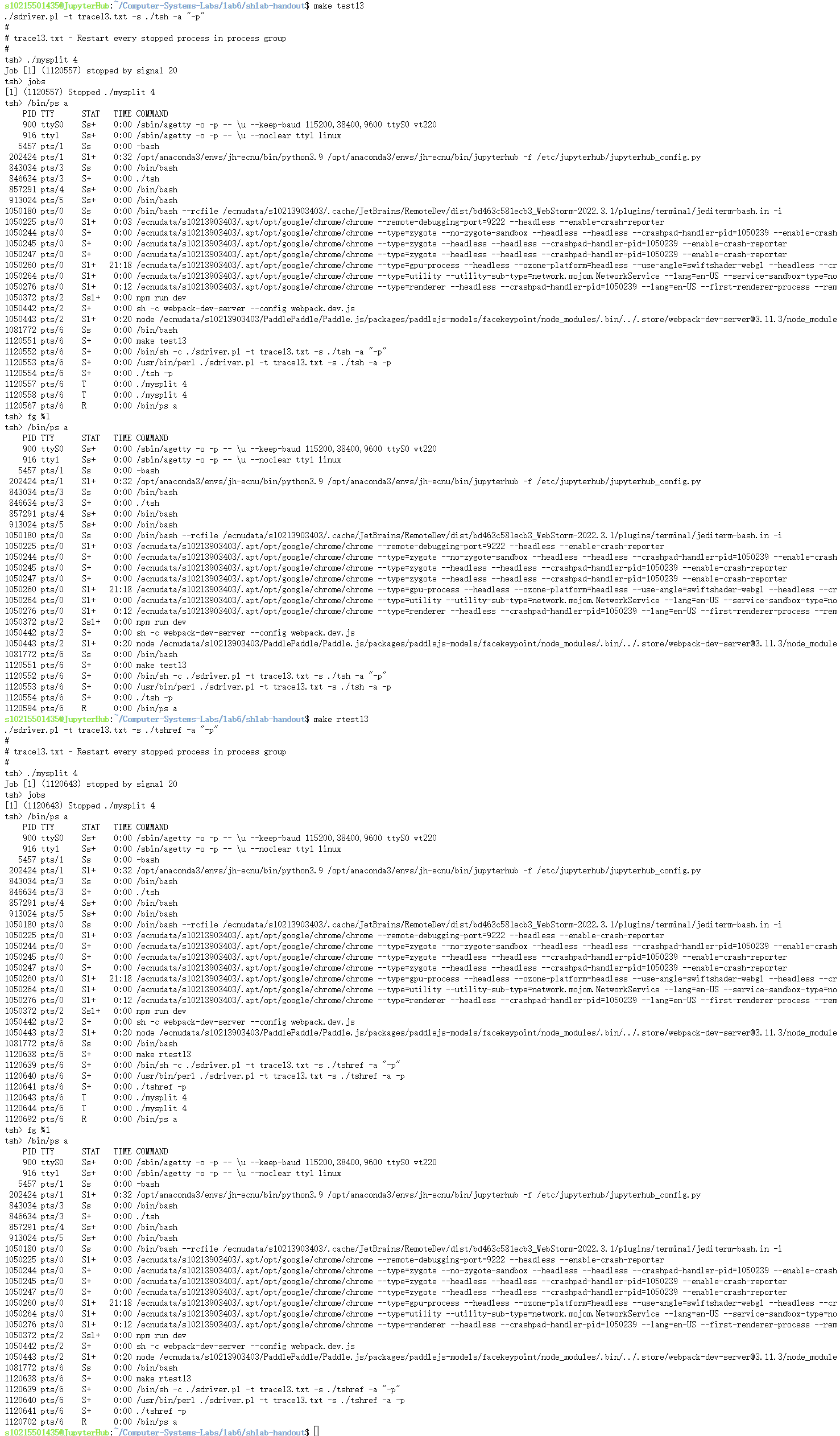
**Test11:**



**Test12:**



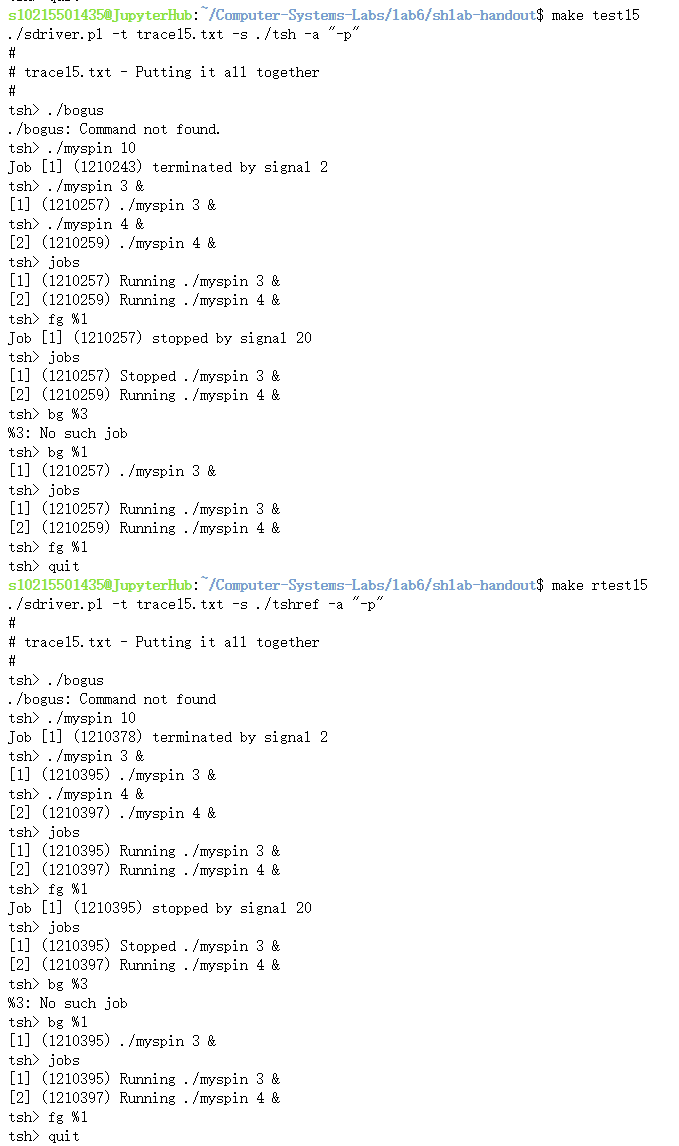
**Test13:**



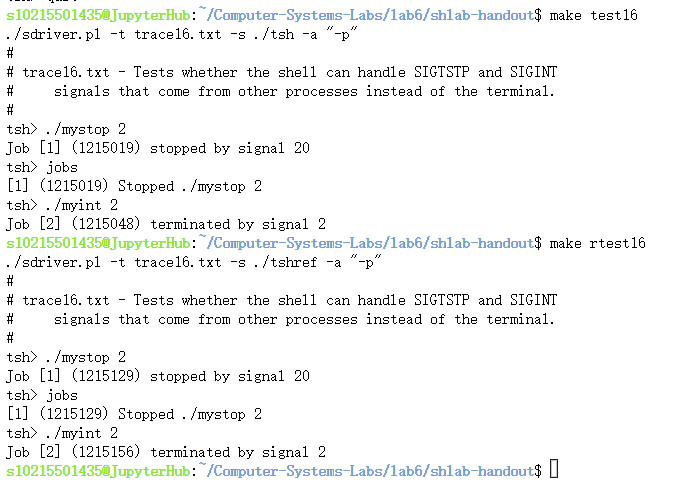
**Test14:**



**Test15:**



**Test16:**



检查完16个trace文件，make testxx和make rtestxx的结果都一样，自测完成。

七个函数：

1. **Eval函数**

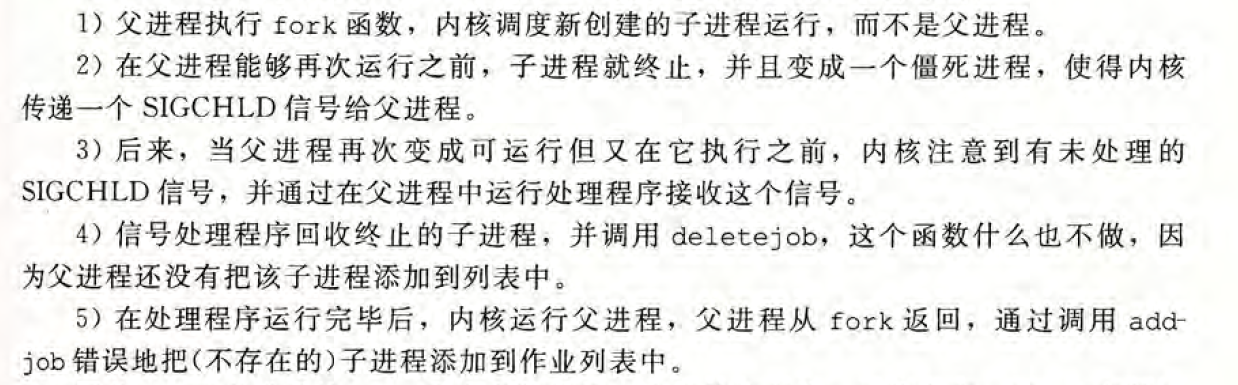
**作用：**解析命令行后判断为内置命令还是程序路径，分别执行。如果是前台作业，则要等待其完成，如果是后台作业，则要输出其相关信息。

**收获到的知识点:**竞争&消除竞争

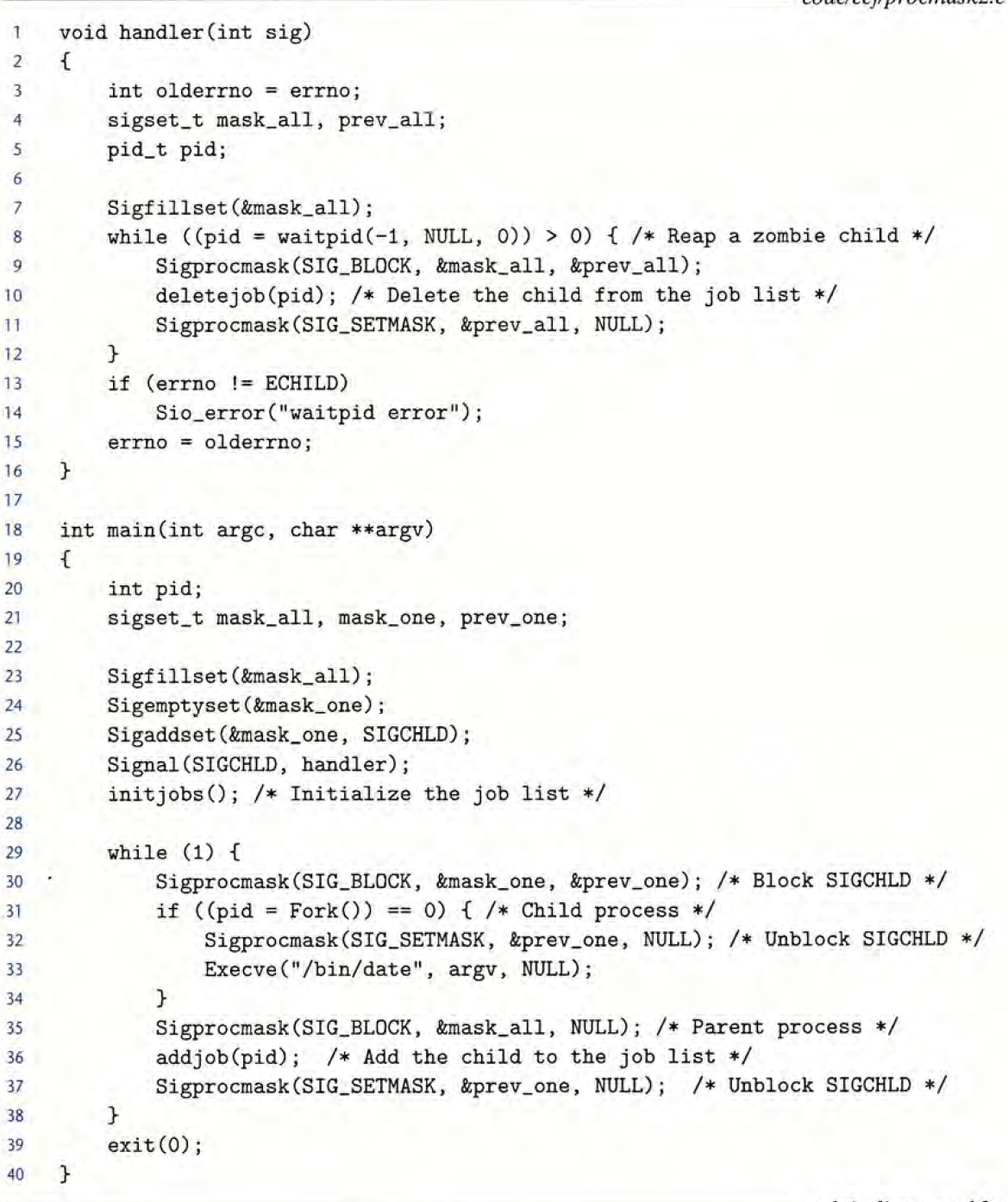


(书p542)

这是一个有细微错误的shell程序，创建子进程以后在运行到addjob之前就结束了并发送SIGCHLD信号来回收子进程，父进程收到信号，调用deletejob函数，但是job不在列表里，父进程就去调用addjob函数，而这个子进程一句在父进程开始运行前就结束了。也就是说调用deletejob的行为发生在addjob之前，两者存在竞争。



消除竞争的方法（书p543）



简单来说，就是在调用fork之前阻塞SIGCHLD信号，然后在调用addjob之后取消阻塞信号，保证addjob一定在deletejob之前。

**遇到的bug:**

1. 为什么调用printf时也阻塞信号？

因为打印了全局变量，而printf函数是线程不安全的，比如可能会出现读内存的时候另一个线程在同步修改这个内存的情况。

1. 为什么要在执行前创造新线程组？

为了将子进程与tsh进程组分开，以免信号终止子进程组时也将tsh进程组终止了。

**设计过程：**

需要了解eval函数和execve执行流程和fork多进程运行方式，程序首先执行eval()，在eval函数中判断是不是内置命令，如果不是就会调用fork创建子进程并利用Execve函数通过argv寻找路径并在子进程中运行可执行文件，若未找到可执行文件则终止子进程。Point:在fork之后，execve之前，子进程应该调用setpgid（0，0），这将把子进程放到一个新的进程组中，其ID与子进程PID相同，这可以确保前台进程组中只有一个进程。

代码和注释如下：

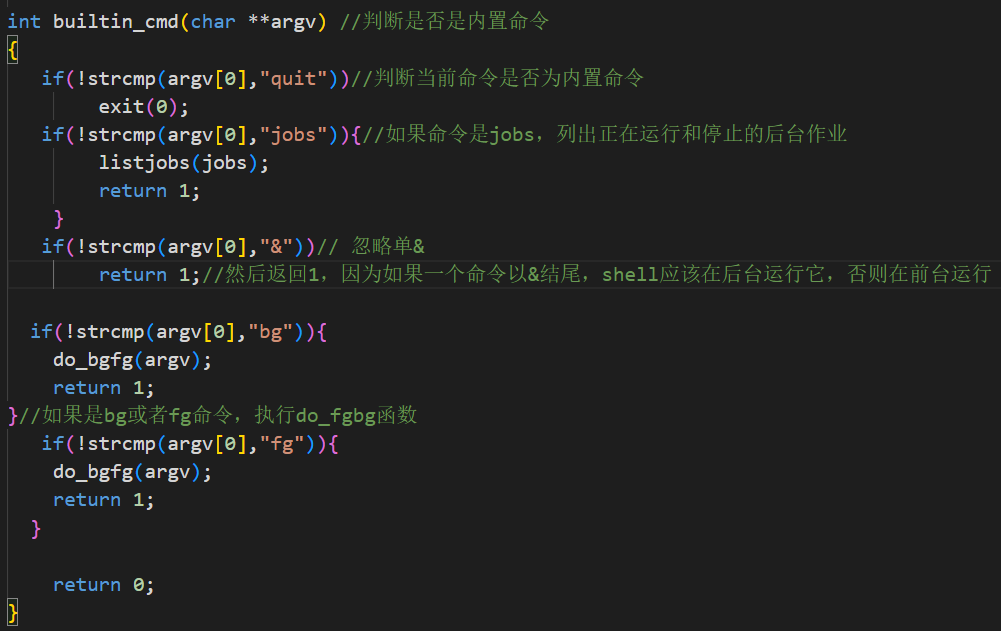


1. **builtin\_cmd函数**

**作用：**识别并执行内置命令：quit,fg,bg和jobs

**设计过程：**当命令行参数为quit时，终止shell；当命令行参数为jobs时，调用listjobs函数，显示job列表；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；判断不是内置命名时exit（0）。

代码和注释如下：



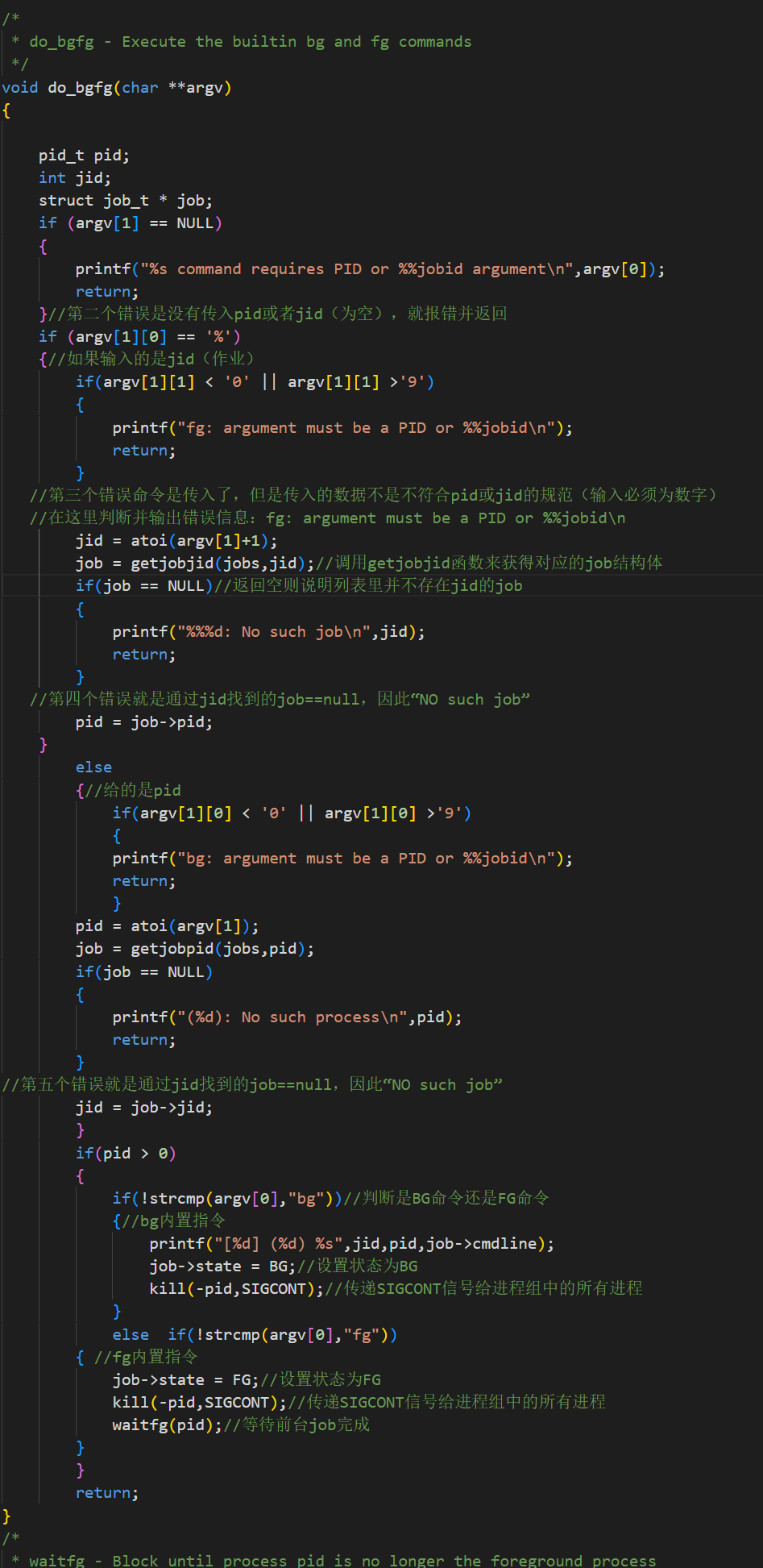
1. **do\_bgfg函数**

**作用：**实现内置命令bg和fg，bg的功能是通过向job对应的作业发送SIGCONT信号来使它重启并放在后台运行；fg的功能是通过向job对应的作业发送SIGCONT信号来使它重启并放在前台运行。

**设计过程：**判断argv()是否带%，若为整数则传入pid，若带%则传入jid。调用getjobjid函数获得对应的job结构体，如果返回为空说明列表中不存在jid的job。

Strcmp函数判断是BG命令or FG命令，if是BG命令，使目标进程重新开始工作，设置状态为BG，打印进程信息；if命令使FG，使目标进程重新开始工作，设置状态为FG，等待进程结束。

代码和注释如下：

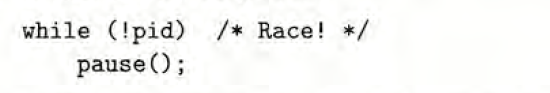


**遇到的bug:**该函数难点在于判断命令行参数，要考虑健壮性。原来的后台进程可能变为前台进程所以要修改jov结构体的相关信息。

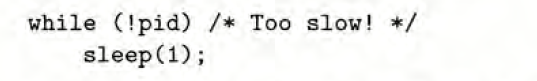
1. **waitfg函数**

**作用：**阻塞前台进程直到其不再是前台进程or等待一个前台作业结束

收获的知识点：显示地等待信号（书p543）

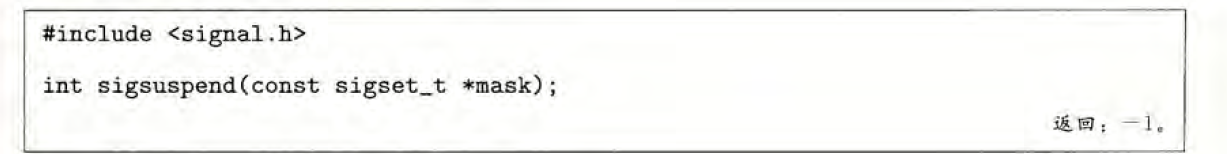
书p544页写了一种用循环等待信号的代码，代码正确但循环在浪费处理器资源。

在循环体内插入pause，但这段代码会引起很严重的竞争条件：如果在while测试后和pause之前收到SIGCHLD信号，pause会一直休眠。

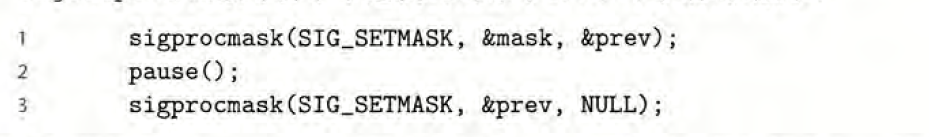


一个解决方法是用sleep函数替换pause，但是间隔太小，循环会很浪费；间隔太大，程序又会很慢。

另一个方法是用sigsuspend函数:



用mask替换当前的阻塞集合，挂起该进程，直到收到信号。如果信号行为是终止该进程，则不从sigsuspend返回就直接终止；如果信号行为是运行一个处理程序，那么sigsuspend从处理程序返回，恢复调用原有的阻塞集合。



Sigsuspend函数和上述代码等价，quiz:如果第一行和第二行之间子进程终止不就永久休眠了吗？答：这其实是个不可中断版本代码，原子属性保证第一行和第二行调用是一起发生的，这样就消除了潜在的竞争。

**设计过程**详见代码和注释：



剩下的是三个信号处理函数（sigchld\_handler，sigint\_handler，sigtstp\_handler）



1. **sigchld\_handler函数**

**作用：**实现一个SIGCHLD信号处理函数，唤醒父进程回收所有僵死进程。

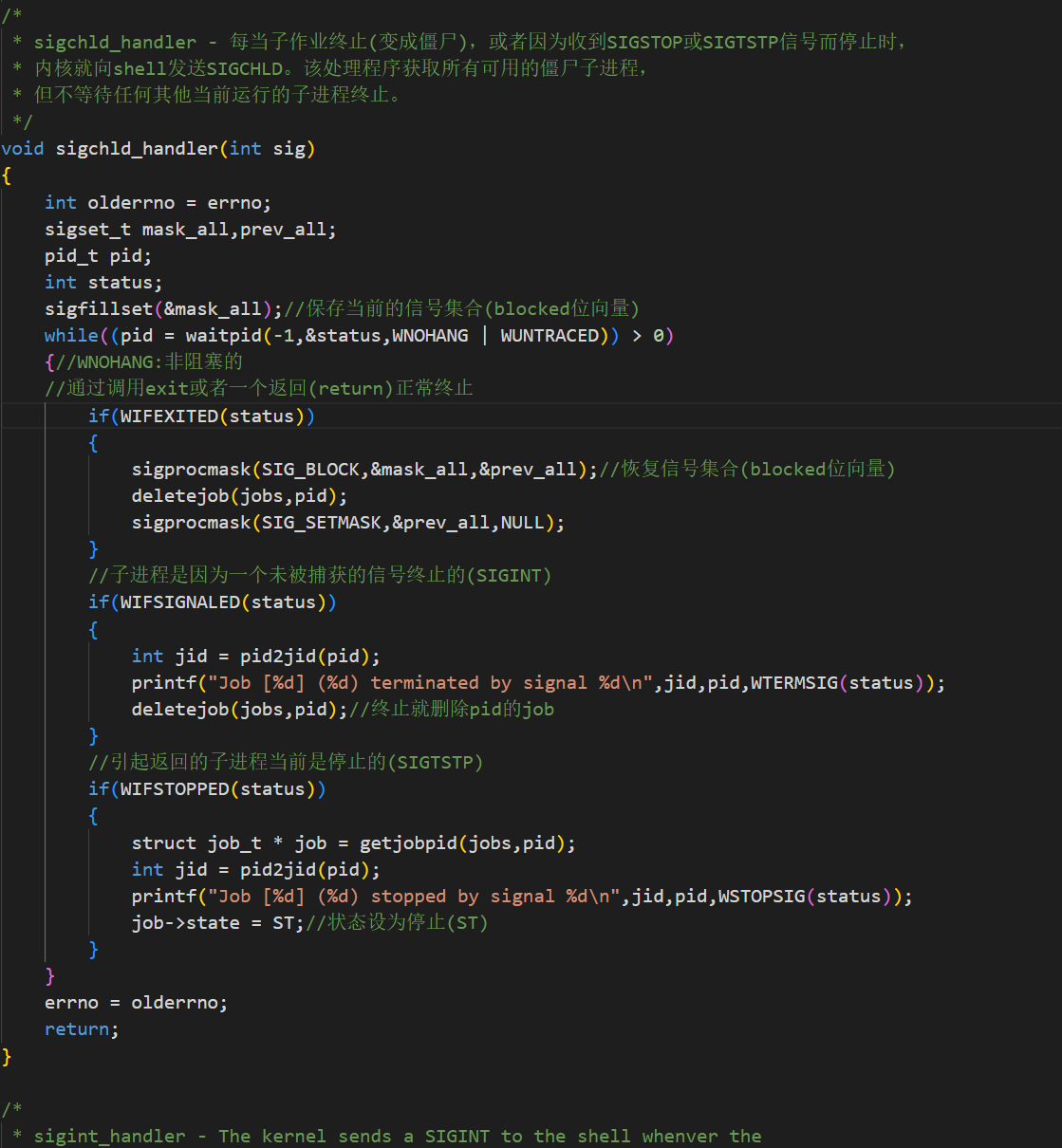
**收获的知识点：**书p516回收子进程。进程终止时，内核不会立马清除它，而是将进程保持在一个已终止的状态中，直到被回收。一个进程可以通过调用waitpid函数来等待它的子进程终止或者停止。对于waitpid函数而言：如果pid>0，回收成功，进程ID等于pid；如果pid=-1，那么父进程所有的子进程组成等待集合。

并发编程原则：注意恢复和保存errno，用局部变量保存errno，运行完成后再恢复。

**设计过程：**在SUGTSTP函数的基础上加上对SIGTSTP的判断和信息显示。多了“引起返回的子进程当前是停止的）”这一内容和WNOHANG|WUNTRACED。WNOHANG：挂起调用进程，直到有子进程终止，若子进程仍然在运行，则返回0。WUNTRACED：挂起调用进程，直到等待集合中的第一个进程变成已终止或者被停止。如果子进程由于传递信号而停止，则返回。WNOHANG|WUNTRACED可以理解为WNOHANG接收终止，WUNTRACED接收停止。

先用while循环调用waitpid函数直到它所有的子进程终止。其次检查退出状态：1、WIFEXITED:通过调用exit或者return正常终止。2、WIFSIGNALED：因为一个未被捕获的信号终止。3、WIFSTOPPED：引起返回的子进程当前是被停止的（注意将状态设置成ST）。

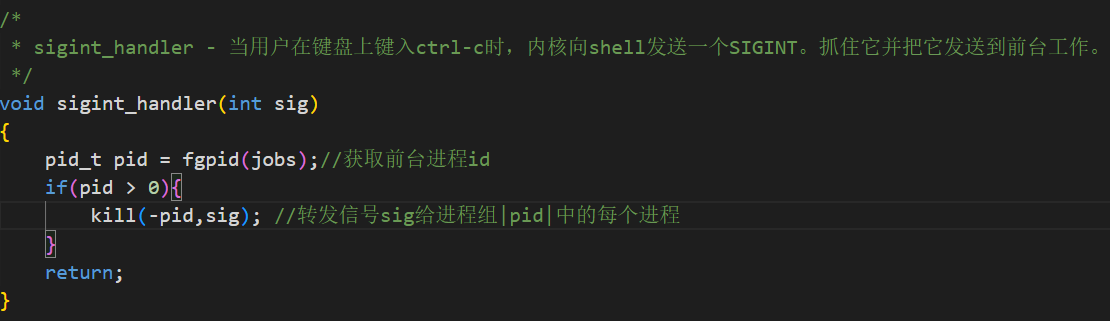
代码和注释如图：



1. **sigint\_handler函数**

**作用：**将SIGINT信号传送给前台进程

**设计过程：**调用fgpid函数返回前台进程id，如果不为0就调用kill函数发送SIGINT信号给前台进程组。代码和注释如下：



1. **sigtstp\_handler函数**（与上一个函数几乎完全相同）

**作用：**将SIGTSTP信号传送给前台进程

**设计过程：**调用fgpid函数获取前台进程pid，判断是否有前台进程，如果没有则直接返回，有则调用kill函数发送SIGTSPT信号给前台进程组。

代码和注释如下：

