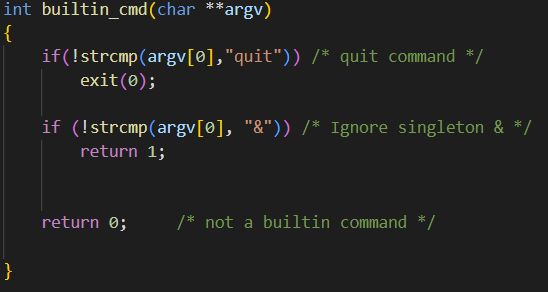
Csapp shell lab实验报告

10215501412 彭一珅

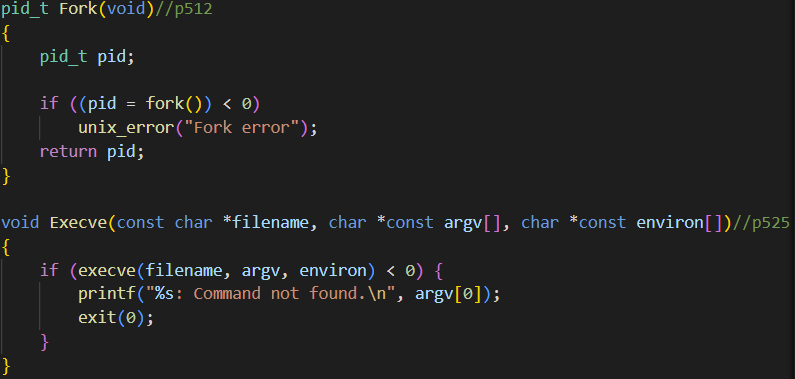
Shell lab的实验目的是实现一个shell程序，执行一系列读/求值步骤，然后终止。读步骤读取来自用户的一个命令行，求值步骤解析命令行，并代表用户执行程序。

先阅读原始给出的main函数，可以看出，main函数在while循环中接续读取文件中的cmdline，并交给eval函数进行解析和执行。因此，我从eval函数入手。将课本p525页的eval和builtin\_command函数写进程序里：

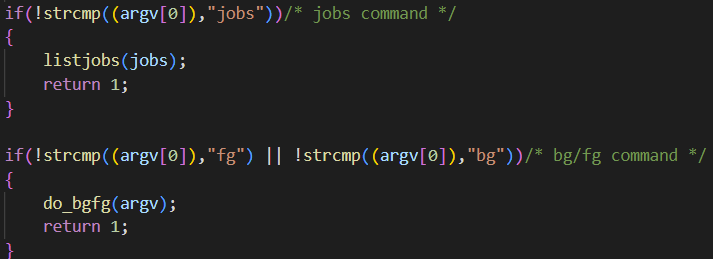




与此同时，将fork，execve函数按照书中方法进行封装，便于后续处理错误。同时被封装的还有kill函数，和后续可能会用到的sigemptyset，sigaddset，sigprocmask，sigfillset，setpgid函数：



接下来对初始版本的eval函数进行解释。首先将输入的指令行拷贝到buf字符串中，用parseline解析，parseline函数可以在argv中填充指令中用空格（可以多个）分割的参数，并返回该指令在后台还是前台运行（由句尾&符号决定）。然后对空行进行处理，直接忽略。接下来判断指令是否是内置的。该实验要求实现的内置指令有“quit”“jobs”“fg”“bg”四个，因此在builtin\_cmd中加入相关的判断：

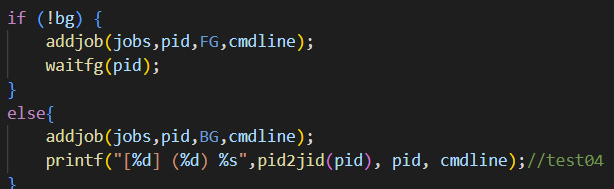


如果输入的指令不是内置指令，那么shell创建一个子进程，并在子进程中执行所请求的程序。然后进行前台后台的判断，如果前台运行，那么就使用waitpid函数等待程序执行完毕，后台运行则直接接收下一条指令。

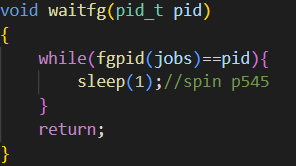
再阅读trace01文件，可以看到只有CLOSE和WAIT两条命令。而trace0203使用了quit内置命令进行退出。此时这三个测试用例都可以正确执行。然后执行make test04时发现输出和rtest04有所不同，于是对eval中的格式化输出进行修改：



trace05在后台执行了myspin程序，但是没有将jobs的列表打印出来，这是因为原本的程序没有将对可执行程序的执行加入到作业列表里，因此需要对eval函数进行修改。

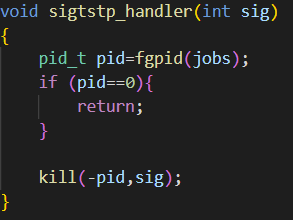
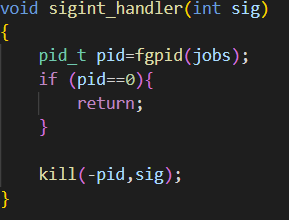


加入addjob函数，使添加的子进程出现在作业列表里，此外，将waitpid改为waitfg，顺便实现waitfg函数：

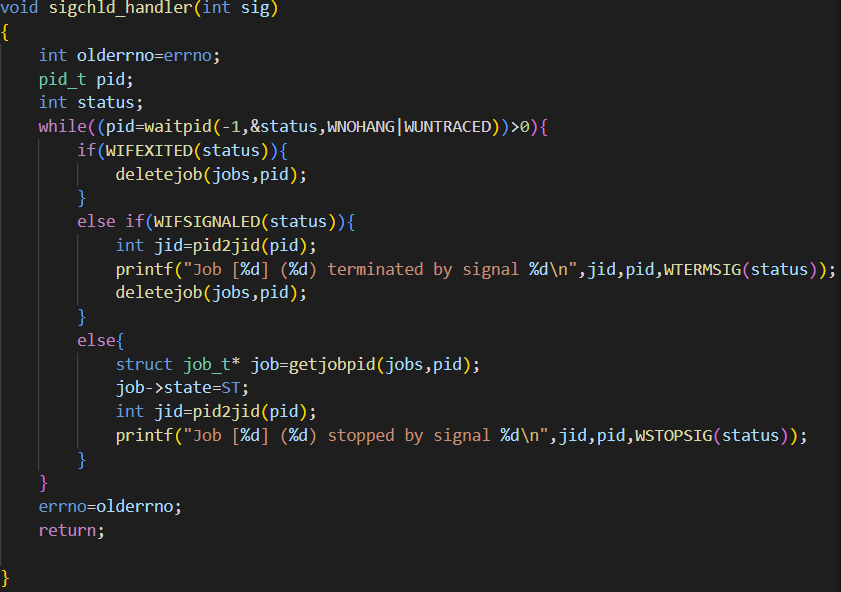


这样可以保证等待结束的子进程只有前台程序。此时可以正确执行test05.

再执行test06，与rtest06相对比，job无法被信号正确终止，这是因为缺少sigint\_handler程序的实现。根据对注释的解读可以知道，sigtstp和sigint函数都是将sigint或sigtstp信号传递给前台进程。因此，只需要对其进行如下相同的实现：



此时给子进程发送了sigint和sigtstp的信号。但这还没有对作业的状态进行输出，于是按照rtest中的格式，在sigchld\_handler中进行输出：

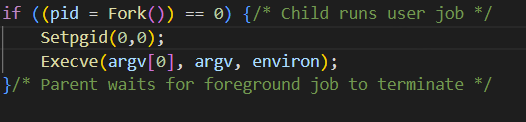


接下来对sigchld\_handler进行解释。首先保存errno的原始状态。在while循环中，waitpid函数立即返回，如果所有的子进程中任何子进程都没有被停止，那么返回0，如果有一个返回或终止，就返回pid，因此只有在pid大于0的时候进行处理。此时有三种情况需要考虑，第一，如果子进程通过调用exit或者一个return正常终止，就会在WIFEXITED(status)中返回1，此时只需要从工作列表中删除即可。第二，如果子进程是因为一个未被捕获的信号终止的，那么WIFSIGNALED(status)函数返回真，这表示用sigint信号进行了终止，此时打印出终止信号的编号，并且deletejob。第三，如果子进程是停止的，那么WIFSTOPPED(status)函数返回真，这表示进程被停止，此时打印出停止信号的编号。这三种情况与函数中的条件控制的三种情况相对应。最后，恢复原先errno的状态。

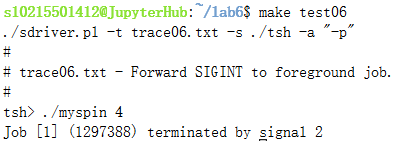
但是此时仍旧无法正确打印如下语句：



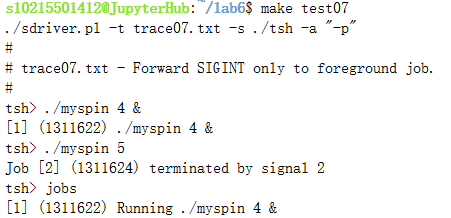
经过探究，发现需要在添加子进程时调用Setpgid函数，创建一个新的进程组，这样在给这个进程组发送信号终止的时候就不会将shell进程也一起终止了。

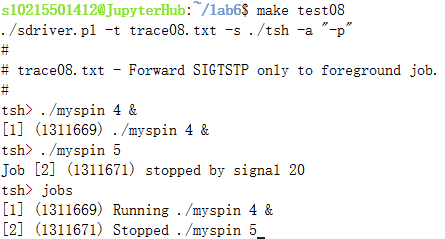


此时test06可以正确打印了

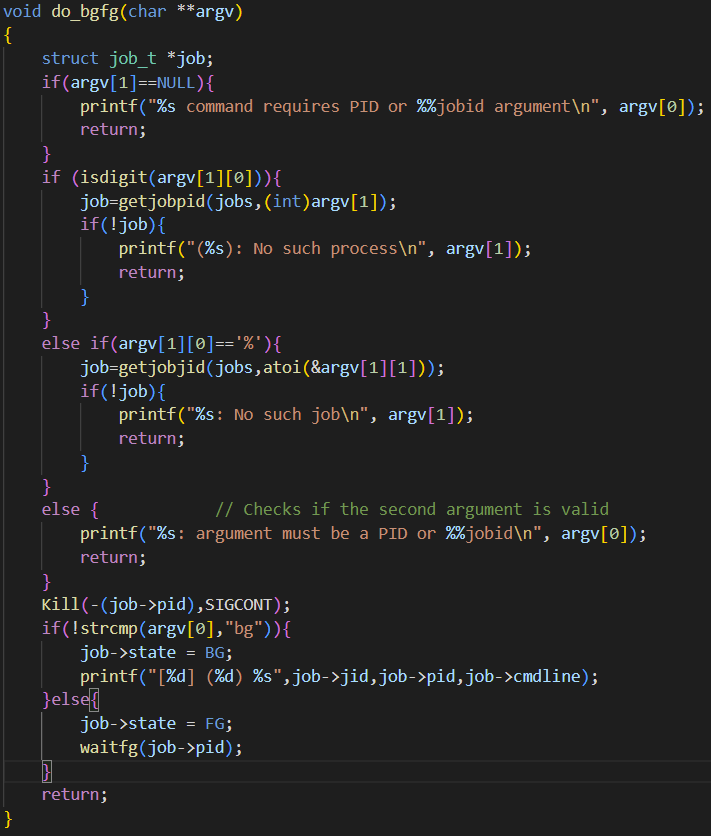


Test0708是为了保证被终止和停止的只有前台进程。由于在handler函数中调用的fgpid函数，因此这一部分没有出错。





Test09要求实现bg转换功能，于是开始实现本实验最后一个函数：do\_bgfg.



基本思路是首先判断输入的指令是否合乎要求。Bg/fg指令形如bg %1或bg 1，分别表示将jid为1的作业转为后台和将pid为1的作业转为后台。因此第二个参数的形式是确定的。所以在第一个if中，先处理没有第二个参数的情况。然后接下来两个条件判断，分别对应pid和jid的情况。基本思路一致：根据jid或pid取得job的指针，然后判断这个作业是否存在，如果不存在，根据rtest中打印的文字进行输出。



在这一句中我曾经遇到bug，为了取argv[1][1]位置的字符串并转换成整数，我一开始使用(int)&argv[1][1]但是转换出的数字并非是我想要的，可能是读取了更多的地址范围。查阅资料得知atoi函数可以直接将字符串转换为整数，因此选用该函数。

在else分支中对第二个参数输入不合法的情况进行了判断，具体语句参考test14中的输出。

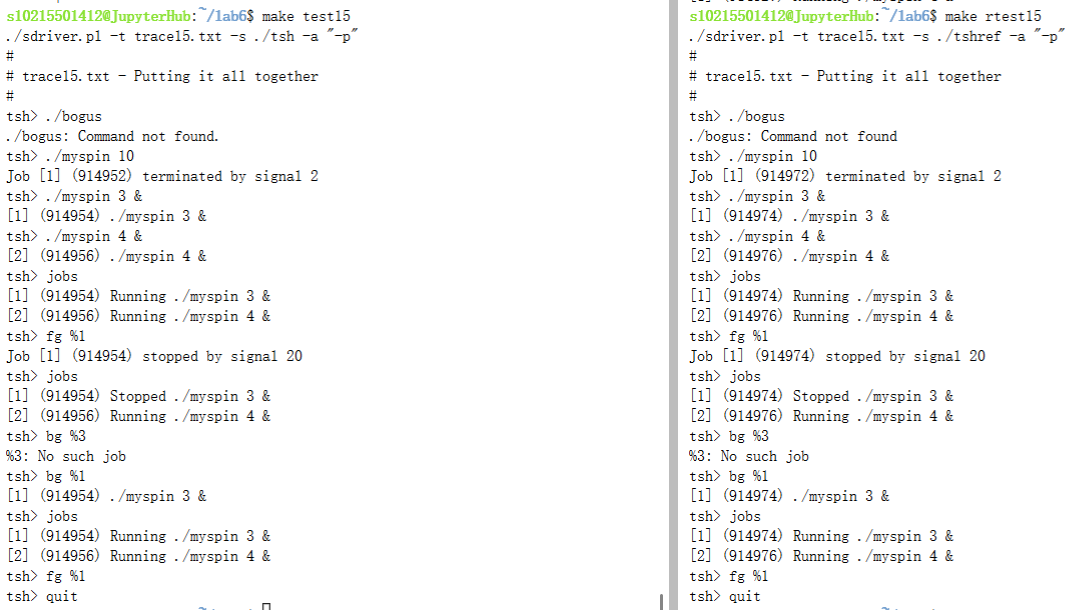
然后将获取到的作业停止，对子进程及其后代输入一个SIGCONT信号，如果停止则继续执行。将state按照前台或后台的要求进行切换，然后根据指定格式进行输出。若转换为前台作业，需要等待前台正在进行的工作结束。

最后对eval函数进行优化，注意到sigchld\_handler中执行的deletejob可能会在addjob之前发生，因此在eval函数中阻塞信号：



设置一个获取信号列表的变量mask，Sigemptyset清空它，Sigaddset函数加入一个SIGCHLD信号，表示子进程被终止，再把这个信号用Sigprocmask函数转移到block中，在调用fork之前实现对信号的阻塞，保证了在子进程被加到作业列表中之后回收该子进程。子进程继承了它们父进程的被阻塞信号集合，所以必须在调用execve之前，解除信号的阻塞。这段内容参考课本p543页的实现，用来同步进程。

接下来所有的test都可以迎刃而解。此处对test15的过程进行详细说明。



由于bogus不在文件夹中，所以执行会报错。在前台执行./myspin 10命令，程序运行10秒，但是中途被SIGINT信号终止。然后在后台执行./myspin 3，./myspin 4命令，用jobs查看正在执行的两个作业。然后将作业号为1的任务转为前台，然后发送一个SIGTSTP信号，发现此时作业1已经被停止。Bg %3指令无法执行，因为没有作业号为3的作业，将作业1转为后台运行，可以看到由于在do\_bgfg函数中发送了SIGCONT信号，因此作业1可以继续执行。最后执行内置指令quit退出程序。