Shell lab实验报告

郭丹琪 2018202067

优秀的地方或提前解决的问题

1. 在eval中处理非内建命令的部分，需要fork出一个子进程，在父进程中要把该子进程加到list中。考虑到竞争的问题，可能在调用addjob把子进程加入list前该子进程就已经结束了并被调用deletejob处理了，这样就会出现问题。所以在这部分加了对sigchld的信号屏蔽，即在fork前开始屏蔽，在addjob结束后解除屏蔽，防止因竞争出现问题。

2. 在waitfg函数中等待前台任务结束中，用的是while循环进行忙等待，并且循环条件是`fgpid(jobs) != 0`，即当前还有前台任务的情况，若`fgpid(jobs)==0`即说明当前没有前台任务了，这样的判断可以概括所有情况，不用再一一区分前台任务是被停止还是结束。在while的循环体中用的是sigsuspend，以防在pause开始前信号就来了，有原子性保障，而且也不会浪费。并且在waitfg中没有其他处理，把处理都留在了sigchld_handler中。

3. 在sigchld_handler中，用的是`while((pid = waitpid(-1, &status, WUNTRACED | WNOHANG)) > 0)`来判断信号。因为可能有多个SIGCHLD并发，所以用的是while循环。且option用的是`WUNTRACED | WNOHANG`，可以立即返回，不用进入挂起状态，而且除了终止的进程外还可以获得停止的进程。因为在处理过程中用到deletejob对全局变量进行处理，担心其他的信号的处理使需要的全局变量发生变化，所以在while的循环体里，一开始就用sigprocmask对所有信号进行屏蔽，直到处理全结束后才解除屏蔽。在处理时用得到的status识别状态，即用`WIFSIGNALED(status)`和`WIFSTOPPED(status)`判断进程是否是因信号而退出或暂停的，如果是的话就打印相关信息。

4. 在do_bgfg中要改变进程的信息，也就是全局变量里的内容，如果有其他信号使其发生改变的话就可能会出现问题，所以在函数的一开始就对所有信号进行屏蔽，在函数返回前结束屏蔽，以防处理完成前进程的状态改变。

遇到的问题及解决方法

1. 在test04里打印出的语句的顺序与正确结果是相反的，说明代码执行的顺序是不正确的，分析是因为waitfg函数还没有写。在waitfg里直接写了个循环条件为`fgpid(jobs) != 0`的while循环进行等待，结果发现虽然正确打印了第一条语句，但是就不再往下运行了，就卡在那里，而原本结果正确的test03也会卡住，但在test03里是没有进行waitfg操作的，只有用到quit，所以考虑可能是sigchld处理的问题。但是反复调整sigchld_handler，确保逻辑没有问题并且没有会停住的地方后，发现测试时还是会卡住。最后只能把原本觉得与问题不相关的eval等函数也理了一遍，发现居然是eval里出现了问题。在父进程里用sigprocmask解除对sigchld的阻塞的时候，没有把SIG_BLOCK改为SIG_UNBLOCK，导致sigchld一直是被屏蔽的，就一直不能回收zombie，所以在test03里会一直卡住。

2. 在test08里发现打印stop信息的语句比打印jobs晚出现，而正确情况应该是stop语句先做完了并把信息打印出来后，再处理jobs那条语句，所以执行的顺序是不正确的。考虑到打印stop信息的语句是在sigchld_handler里处理的，没有在处理jobs前就打印应该是被其他信号打断了。所以在sigchld_handler里，在发现sigchld信号并要开始处理时就把所有信号都屏蔽了，等该条信号处理完再把其他的解除屏蔽。

3. 在test09里，在打印出`tsh> bg %2`后就卡住没有反应了。分析可能是do_bgfg里处理的问题，然后就在do_bgfg里加了信号屏蔽，即在处理bgfg时屏蔽所有信号，然后测试结果正确。但是在test10里，出现了一样的问题，也是在`fg %1`后就卡住没反应了。分析bg和fg的处理上的差别在于fg处理时要有个waitfg的操作，所以判断是waitfg的问题。在waitfg中是个循环条件为`fgpid(jobs) != 0`的while循环，但是while循环里是空的，即不作处理，就一直循环判断fgpid(jobs)的结果。感觉这样会一直占着做判断，太浪费了。所以在while里用了sigsuspend等待信号，这样也不会有race的问题，结果也是正确的。

认为的重难点

1. 判断什么地方需要信号屏蔽是个重点也是难点。因为需要分析清楚哪个部分被信号打断后可能出现问题，得把各种情况都考虑周全，并且根据错误的测试结果来找到是哪里因没有信号屏蔽而出现问题。

2. 在根据测试的结果调整代码的时候，要根据实际shell处理的逻辑来调整，找到实际的问题所在，而不能只是为了测试结果的正确而针对正确结果进行修补拼凑。而要找到实际的问题并且根据正确的shell的逻辑来修改是比较难的也是重要的，难在要能找出是哪里出现问题，正确情况应该是怎样的，要怎么改才能使得运行时按正确的逻辑，而这也重要在只有这样才能正确理解shell实际的操作，才能达到实验的目的。

3. 最难的就是要把运行的逻辑顺序理清楚，还得把各种可能情况考虑周全，分析各种竞争下出现的问题。

反思总结

1. 前期的测试结果正确不代表前期的代码没有问题，后期测试出现问题的时候，有bug的地方不一定就是后期写的代码，问题可能是出在前期写的代码。而且看起来与问题最相关的部分的代码不一定是有bug的地方，出错可能是受其他部分的bug影响。所以在debug的时候，发现相关部分没有问题后，就得去看前期写的看起来没那么相关的部分，有可能问题就出在不直接相关的地方，不能死扣一个地方。

2. 逻辑要先理顺，先理清楚每一种情况要分别怎么处理，把所有情况都考虑周全，然后再开始写。