shell lab 应该是到现在为止我遇到的难度最大的一个lab了——其实现在想想是因为自己虽然明白了进程的内容，对信号还是太不熟悉，而且不认真看书——以后看书的时候一定要注意书上的代码了，这次的教训真的很深刻。

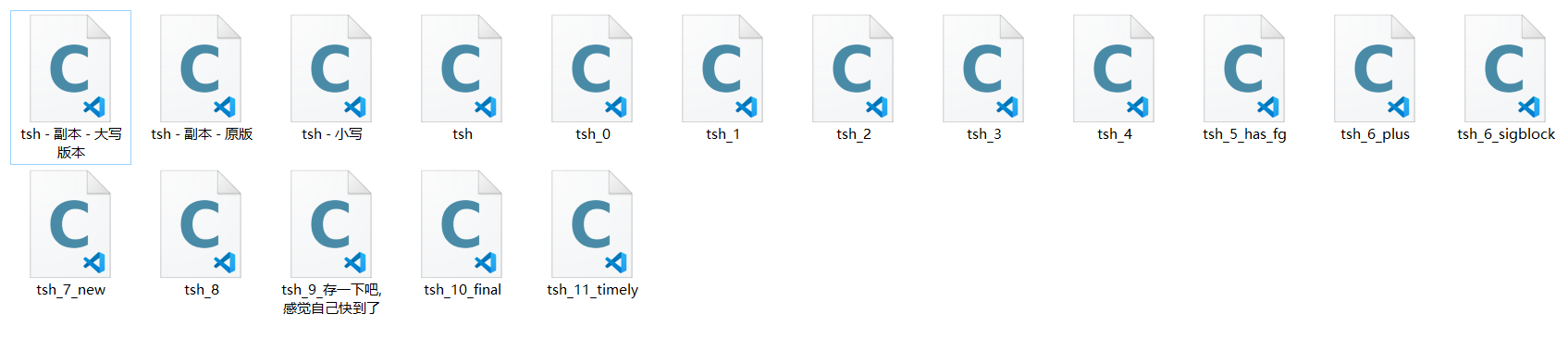
shell lab从周六晚上开始写，周日写了整整一天！我当时经历了这学期以来最为崩溃的一天，比data lab的崩溃程度要大得多，周一又是写了一整天，一直到晚上十二点我才交上去，还经历了100-96-92-0（！）-100的波折，直到周二把所有格式都改清楚，前前后后四天时间——真的是我用时最久的一次，中间甚至掉了眼泪。

现在周四，本来已经把这件事放下了，但是昨天帮室友debug的时候突然发现其实自己还有很大一部分问题没有解决清楚，看来ICS真是博大精深……而且自我感觉用时四天的lab的确不普通，我自认为自己写出了很清晰的逻辑，是我为数不多感到满意的一次了，假装小小骄傲一下。

今天看了看网上的，学长学姐们都是几乎直接把完整的最后解答列了出来，但是我还是决定用更麻烦的方法，把自己从头到尾的几次重要改进加进来——毕竟是要给自己看的，直接列答案我敢保证我以后也不知道为什么了（就是菜的理直气壮！

不说废话，我来记录一下自己在四天里都经历了什么，不管是其他看官还是自己，未来应该都会有一些收获叭！

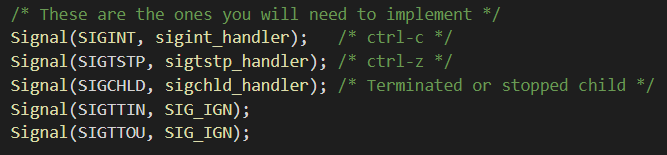
我把自己每一次大修的底稿都留了下来：



看得我都同情当时的自己……

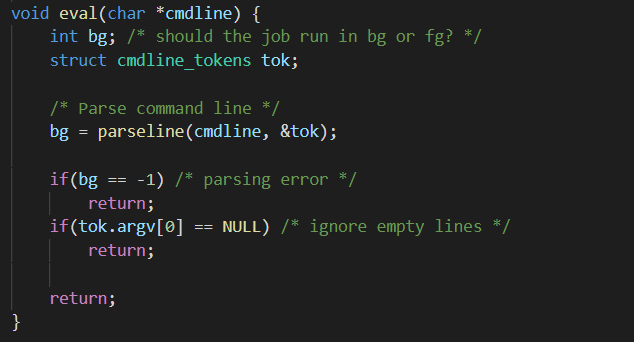
第一阶段：不知所措阶段

刚刚下手，简单看了下main函数在干什么，在这里发现了很重要的一个点！



↑我后来在屏蔽信号部分豁然洞开就是发现了这里，看起来很平常，但是其实这里提醒了我不能仅仅把SIGCHLD加入BLOCK（现在想想当时的自己真的很蠢）。

接着往下，到了第一个需要补充的函数：

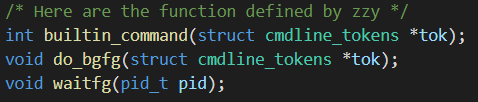


这里有了明显的提示，也就是parseline的使用，parseline在我看来就是一个工具，把命令行里面的信息提取出来了，不再多说。

……

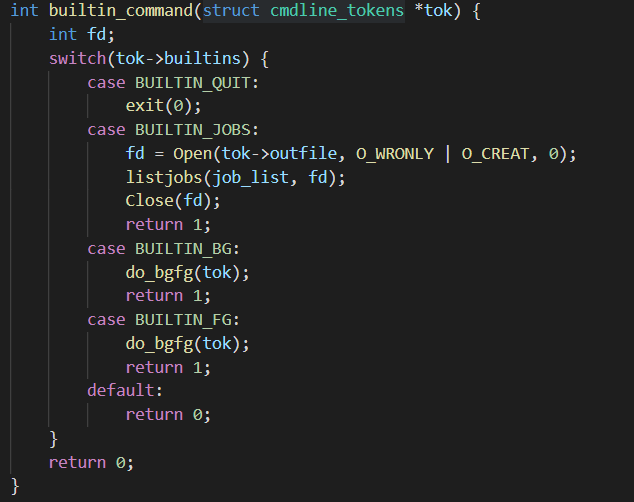
又有谁能想到当时头铁的我直接就开始写了？当然代价是十分惨痛的，我周日一下午都花在改变之前的错误认知上了。

显而易见地，我去抄书了。



为什么这里是waitfg呢？因为回收子进程应该落在SIGCHLD的处理函数里哦！

首先判断是不是内部命令，在这里一定要好好读struct cmdline\_tokens里面都有什么，就可以直接用parseline做好的工程。



这是我的第一版，当时自己还没有弄明白I/O重定向，所以凭借直觉写了下来，但是逻辑还是清楚的，虽然之后找到了好太多的方法hhh

至于不是内置命令，自然就是要开始执行了：

    if(!builtin\_command(&tok)) {

        Sigfillset(&maskall);

        Sigemptyset(&mask);

        Sigaddset(&mask, SIGCHLD);

        Sigprocmask(SIG\_BLOCK, &mask, &prev); /\*block SIGCHLD\*/

        if((pid = Fork()) == 0) {             /\* Child runs user job \*/

            Sigprocmask(SIG\_SETMASK, &prev, NULL);

            /\* Unblock SIGCHLD in child \*/

            Setpgid(0, 0);

            // if(Setpgid(0, 0) < 0) unix\_error("eval: setpgid failed");

            if(execve(tok.argv[0], tok.argv, environ) < 0) {

                printf("%s: Command not found.\n", tok.argv[0]);

                exit(0);

            }

        }

        else {

            /\* parent process block the signal \*/

            if(bg)

                addjob(job\_list, pid, BG, cmdline);

            else

                addjob(job\_list, pid, FG, cmdline);

            Sigprocmask(SIG\_UNBLOCK, &mask, NULL); /\* UNBLOCK SIGCHLD\*/

            /\* Parent waits for foreground job to terminate \*/

            /\* addjob(struct job\_t \*job\_list, pid\_t pid, int state, char

             \* \*cmdline);\*/

            if(!bg)

                waitfg(pid);

            else

                printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);

        }

        return;

    }

在Writeup里面已经写好了要用Setpgid(0, 0)，所以不必多说。

父进程这里还是有点思考的，就是addjob命令，第一次直接看程序其实我不太理解job\_list到底干了些什么，其实就是把所有在工作的job的相关信息给储存了起来，便于查找和组织。

到这里就可以发现我的第一个大错误，致命错误。

我没有屏蔽信号，else里面一次也没有。

对信号的不透彻了解的阴影笼盖着我。

先不管，我们把框架搭完。

上面出现过的两个小函数在这里：

void do\_bgfg(struct cmdline\_tokens \*tok) {

    char \*next\_argv = tok->argv[1];

    struct job\_t \*cur;

    int jid = 0, pid = 0;

    if(next\_argv == NULL) {

        printf("%s command requires PID or JID argument\n", tok->argv[0]);

        return;

    }

    if(next\_argv[0] == '%') {

        jid = atoi(((char \*)next\_argv) + 1);

        cur = getjobpid(job\_list, jid);

        if(cur == NULL) {

            printf("No such job\n");

            return;

        }

    }

    else {

        pid = atoi(next\_argv);

        cur = getjobpid(job\_list, pid);

        if(cur == NULL) {

            printf("(No such process\n");

            return;

        }

    }

    Kill(-(cur->pid), SIGCONT); /\* Later add a judgment of success \*/

    if(tok->builtins == BUILTIN\_BG) {

        cur->state = BG; /\* run in background \*/

        printf("[%d] (%d) %s", cur->jid, cur->pid, cur->cmdline);

    }

    else if(tok->builtins == BUILTIN\_FG) {

        cur->state = FG;

        waitfg(cur->pid); /\*wait until process is completed \*/

    }

    else {

        printf("bg or fg error:%s\n", tok->argv[0]);

    }

    return;

}

void waitfg(pid\_t pid) {

    while(pid == fgpid(job\_list)) {

        sleep(0);

        /\* there may use unsleep() \*/

    }

}

好像也不是很小的啦……

sleep()是明确写过不能使用的，但是因为我对信号的苍白理解，先这样吧，反正我见过别人这样写貌似没问题。

原因是这样的：

**MSDN的说明：指定零 (0) 以指示应挂起此线程以使其他等待线程能够执行。**

**Thread.Sleep(0) 并非是真的要线程挂起0毫秒，意义在于这次调用Thread.Sleep(0)的当前线程确实的被冻结了一下，让其他线程有机会优先执行。Thread.Sleep(0) 是你的线程暂时放弃cpu，也就是释放一些未用的时间片给其他线程或进程使用，就相当于一个让位动作。**原文链接：<https://blog.csdn.net/qiaoquan3/article/details/56281092/>

我们继续，其实到这里就只剩下信号处理函数了。

void sigchld\_handler(int sig) {

    int olderrno = errno;

    pid\_t pid;

    int status; /\* the child process's status \*/

    sigset\_t mask\_all, prev\_all;

    Sigfillset(&mask\_all);

    while((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) {

        /\* if exist child stopped or terminated\*/

        Sigprocmask(SIG\_BLOCK, &mask\_all, &prev\_all);

        if(WIFEXITED(status))

            deletejob(job\_list, pid);

        else if(WIFSIGNALED(status)) {

            printf("Job [%d] (%d) terminated by signal %d \n",

                   pid2jid(pid),

                   pid,

                   WTERMSIG(status));

            sigint\_handler(-pid);

        }

        else if(WIFSTOPPED(status)) {

            printf("Job [%d] (%d) stopped by signal %d \n",

                   pid2jid(pid),

                   pid,

                   sig);

            getjobpid(job\_list, pid)->state = ST;

        }

        Sigprocmask(SIG\_SETMASK, &prev\_all, NULL);

    }

    errno = olderrno;

    return;

}

void sigint\_handler(int sig) {

    pid\_t pid = fgpid(job\_list);

    if(pid != 0) {

        int jid = pid2jid(pid);

        if(sig == SIGINT) {

            Kill(-pid, SIGINT);

            printf("Job [%d] (%d) terminated by signal %d\n", jid, pid, sig);

            deletejob(job\_list, pid);

        }

        else if(pid == -sig) { /\* I'm not sure \*/

            printf("Job [%d] (%d) terminated by signal %d\n", jid, pid, 2);

            deletejob(job\_list, -sig);

        }

    }

    return;

}

void sigtstp\_handler(int sig) {

    pid\_t pid = fgpid(job\_list);

    if(pid != 0) {

        int jid = pid2jid(pid);

        if(sig == SIGSTOP) {

            printf("Job [%d] (%d) Stopped by signal %d\n", jid, pid, sig);

            getjobpid(job\_list, pid)->state = ST;

            Kill(-pid, SIGSTOP);

        }

        else if(pid == -sig) { /\* I'm not sure \*/

            printf("Job [%d] (%d) Stopped by signal %d\n", jid, pid, 20);

            getjobpid(job\_list, pid)->state = ST;

        }  //已经停止的进程通过SIGCHLD发送信息

    }

    return;

}

需要注意的是！再次一个错误理解……果然事后诸葛亮，现在我终于明白了，我把SIGTSTP和SIGINT其实都想复杂了，写的时候我偷偷打开trace08-trace14（15？）看过了，以为发给进程组和发给“CHILD ITSELF”是不同的，于是抖机灵的在SIGCHLD里面加了sigint\_handler(-pid)这样的东西，试图区分两种情况，实际上是不需要的（而且还会错）

接下来开始测试！首先谢谢助教哥哥，真的，不会有人相信我竟然因为一个大小写的问题烦了助教整整一个上午，不说了，今年过年我一定给学长买礼物！

大写是csapp.h和csapp.c中定义的，加好了错误处理，但是很不幸我用不了……

但我的经验是：

sudo cp csapp.h /usr/include

sudo cp csapp.c /usr/include

这样把两个直接放在usr inlude里面，它就会变成正常的头文件一样可以include<>但是我没有成功

还有一种办法，就是不放进去，但是去改MakeFile里面的表项，编译里面直接加上csapp.c 和-lpthread

注意：一定要注意文件名写下时候的顺序！！！

但是因为tsh.c和csapp.c又有重名函数和变量，于是还是不行，而我删改掉之后再次出现各种连接错误，所以……只好乖乖用小写了。

第一次直接./sdriver，32分，我给自己留下了充足的进步空间……

终于开始悲伤的debug时间：



我的一个bug就在这里：

eval里面：

        else {

        sigprocmask(SIG\_BLOCK,&maskall,NULL);

            /\* parent process block the signal \*/

            if(bg)

                addjob(job\_list, pid, BG, cmdline);

            else

                addjob(job\_list, pid, FG, cmdline);

            sigprocmask(SIG\_UNBLOCK, &mask, NULL); /\* UNBLOCK SIGCHLD\*/

            /\* Parent waits for foreground job to terminate \*/

            /\* addjob(struct job\_t \*job\_list, pid\_t pid, int state, char

             \* \*cmdline);\*/

            if(!bg)

                waitfg(pid);

            else

                printf("[%d] (%d) %s\n", pid2jid(pid), pid, cmdline);

        }

但是大家都明白，问题不是这里，程序的主干已经有了（多谢课本），错的是信号处理程序。

第二个bug:

builtin\_command里面：

我显而易见是把所有东西都给重定向掉了，以至于控制台上毛都没有，因为还没有明白输入是怎么样的，先加上判断好了：

   switch(tok->builtins) {

        case BUILTIN\_QUIT:

            exit(0);

        case BUILTIN\_JOBS:

            if(tok->outfile != NULL) {

                fd = open(tok->outfile, O\_WRONLY | O\_CREAT, 0);

                listjobs(job\_list, fd);

                close(fd);

            }

            else{

        //printf("test,no redestinate\n");

                listjobs(job\_list, STDOUT\_FILENO);

        }

            return 1;

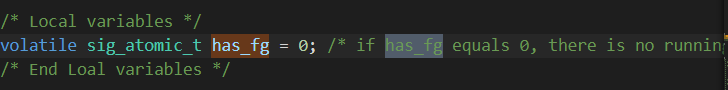
之后我犯了没有好好看要求的错误，做了一个不仅无意义，而且有害处的事，我把sigchld\_handler的输出放在了其他信号处理函数上了……发现了自己的江郎才尽。

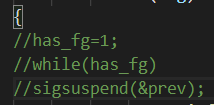
还做了这样一项工作:在sigtstp\_handler和sigint\_handler里面加上了信号屏蔽，这里没有错！因为我要访问全局变量job\_list，唯一的问题是，我一股脑的一进入函数就开始屏蔽信号，出来再解屏……所以我开始疯狂time out。（具体见tsh\_4.c & tsh\_6\_sigbock.c）

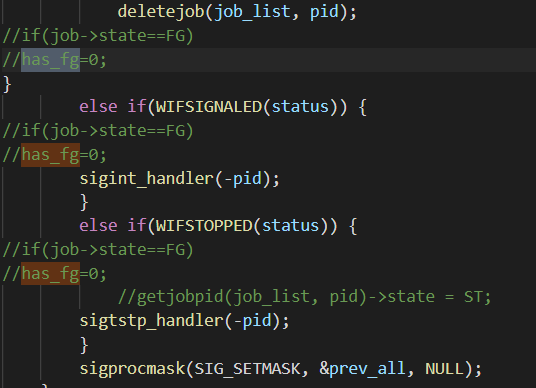
那种感觉真的不怎么好。

就这样，周日一天过去了。

接下来我陷入了慌乱，做了很多无意义的工作，包括加了一个全局变量flag（在这里直接放代码）：







其实脱离苦海的我此时可以发现，自己当时其实已经意识到信号处理函数的问题了，但是惯性问题……

后来我把这里改掉了，避开了全局变量的问题，但是室友在昨天也遇到了同样的问题，这里为什么不行，昨天我跟室友讨论到十二点，马上小班课了，我再跟助教讨论一下：

（室友写法详见 “解决trace05的.c”,一样的flag，一样的flag倒了）

/\*补充：

\* 小班课上完了！！！非常非常感谢助教哥哥！我现在已经明白是为什么

\* 了，应该是这里出了问题（摘自室友的代码），

    if(tok->builtins == BUILTIN\_BG)  //如果是bg job

    {

        job->state = BG;

        printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);

    }

    else if(tok->builtins == BUILTIN\_FG) {

        job->state = FG;

        flag=1;

        wait\_foreground(job->pid);

    }

\* 这里可能出现flag=1还没有执行但是本次前台进程已经执行完毕的情况

\* 导致不能被正常回收。

\* 其他班的助教提供了相似的思路，认为这个在父进程中公用的全局变量

\* 可能被不正常按照逻辑修改。

\* 写着心得我又想到一个更好的办法！如果直接用job->state作判断，可

\* 以直接省略fgpid(job\_list)函数的调用！看来写心得体会是真的有用啊

\*/

之后我又经历了一段暴躁时间，胡乱block重出江湖……真是掩盖不了自己是个小垃圾的事实……即便是在写下反思的时候，还是没有明白啊。

更多暴躁操作详见tsh\_7\_new.c & tsh\_8.c，这两版后来都被我废除了，真的完全是在胡写。

但实际上，虽然做了很多无用功，但是在这里我除了time out和重定向的问题，所有问题已经全部解决了，这时我已经过了22个trace（当然是多次运行靠运气避过timeout之后）。

唉好累啊，我不太想写了。

终于在周一晚上！我想起了本周的作业的第一道题，里面有关于SIGCONT的问题，于是调转了builtin\_command中发送信号、输出和修改状态的顺序，把sigtstp\_handler里面对全局变量的操作改掉了（这部分真的完全是没有好好看writeup啊，不会的时候一定要先看writeup！），deletejob的操作绝对不能在sigint\_handler或者sigtstp\_handler中完成。

但是这些都不是关键问题

都不是关键问题

都不是……

问题出在最开始，我缺少了最开始的信号处理过程：

    sigaddset(&mask, SIGCHLD);

    sigaddset(&mask, SIGINT);

    sigaddset(&mask, SIGTSTP);

原本我只有一句：

sigaddset(&mask, SIGCHLD); /\*mask only block SIGCHLD\*/

真的是令人绝望，看到我自己的注释了吗？

only block SIGCHLD？？？

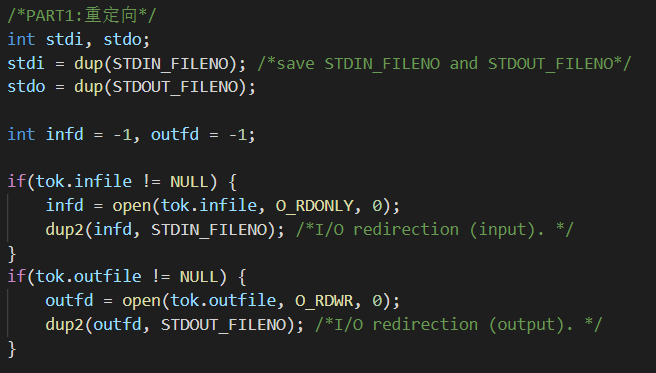
然后问题就解决了！这样完全避免了父进程或者子进程被非正常中断的情况。这样的简单错误浪费了人整整两天，的确令人痛心疾首，不过就在刚刚看树洞的时候，我发现不止一个人跟我一样，微笑🙂

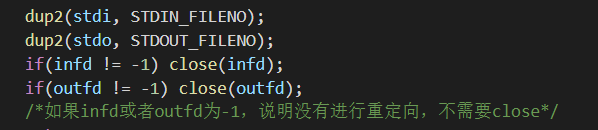


这样就来到了（我自以为的）最精彩的地方——

不是我夸自己，我觉得自己的重定向真的做的非常优雅。为了写出这部分，我真的查了非常非常多的资料。

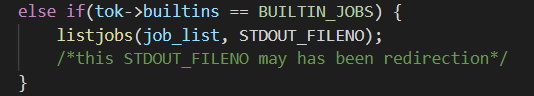
直接贴代码：





可能一下子理解不了为什么这样我认为最好？

当然是——这样在执行输出的时候：



因为重定向，不再需要任何特殊判断。

一开始这只是我的一个设想，因为很明显在这之前我写的也是大堆判断，但是在第一个input redirection出错之后，我开始考虑有没有可能将**STDIN\_FILENO and STDOUT\_FILENO** 直接重定向？

在这里就要再次感谢助教哥哥！他给我们的练习题里出现了这样一个函数：

int dup(int oldfd);

dup用来复制参数oldfd所指的文件描述符。当复制成功时，返回最小的尚未被使用过的文件描述符，若有错误则返回-1。错误代码存入errno中。返回的新文件描述符和参数oldfd指向同一个文件，**这两个描述符共享同一个数据结构，共享所有的锁定，读写指针和各项全现或标志位**。

更重要的是，它使用的头文件是unistd.h，已经给好的，我知道这是正常的，但在当时的我看来，这简直说明我找到了标准答案！

这让我注意到直接复制然后重定向的可能性，简单翻一下之前的函数——看看我发现了什么！

    /\* Redirect stderr to stdout (so that driver will get all output

     \* on the pipe connected to stdout) \*/

    dup2(1, 2);

stderr可以重定向到stdout上！再查一下资料，原来STDIN\_FILENO和STDOUT\_FILENO重定向是正常操作！那还等什么！

于是lab好像就……轰轰烈烈的完结了？

实际上还没有，有很多地方做了无用功，甚至一些多余的sigprocmask语句导致了偶尔出现的time out问题（至少我以为是这样）。语句也有大量赘余，导致在这时我的程序可读性非常差，也就是tsh\_6\_plus.c

但是我已经100/100了！接下来不过是追求完美罢了！

1. 检查的时候我发现自己根本没有充分利用tok->builtins = BUILTIN\_NONE;这句话，于是果断合并builtin\_command；
2. 原本的jid=atoi((tok.argv[1])+1)被我改成了jid=atoi((tok.argv[1])+sizeof(char))，这样就避免了助教说的magic number（不过今天我知道1真的不能算是magic number，所以无所谓的）；
3. 为了追求美观，我把sigchld\_handler里面条件语句的顺序改变了，我倾向于将比较少执行过程的语句往后放，这样貌似会有提高速度的倾向，虽然在本次lab中我没有实验；
4. 删掉了我自己认为没有用的mask，这样真的会对提高速度有效！亲测我修掉两个mask之后，好几个trace都不再timeout了（不过自己其实并没有太明白这里具体的意义，哭哭）；
5. 利用tok里面的东西，我还合并了两次addjob，这样做的目的还是为了能够跑的快一点，这种就是数据传送而不是条件传送了嘛

就是这里：

  if(bg)

                addjob(job\_list, pid, BG, cmdline);

            else

                addjob(job\_list, pid, FG, cmdline);

认真读parseline，我直接用一个bg来记录BG或者FG，直接：

               addjob(job\_list, pid, bg + 1, cmdline);

是不是很漂亮！这也是我很喜欢的一个地方，虽然只是一个小改动，但是用到了ICS之前的知识！

因为已经过去了两三天，可能有一些慌乱情况下的所谓改进我没有一一写下，总之我从

tsh\_6\_plus.c

来到了

tsh\_9\_存一下吧,感觉自己快到了.c

之后的修修补补就不多说了，我直接交了上去。

最后一个小插曲：这也是我本次lab最后一个没有解决完毕的问题：

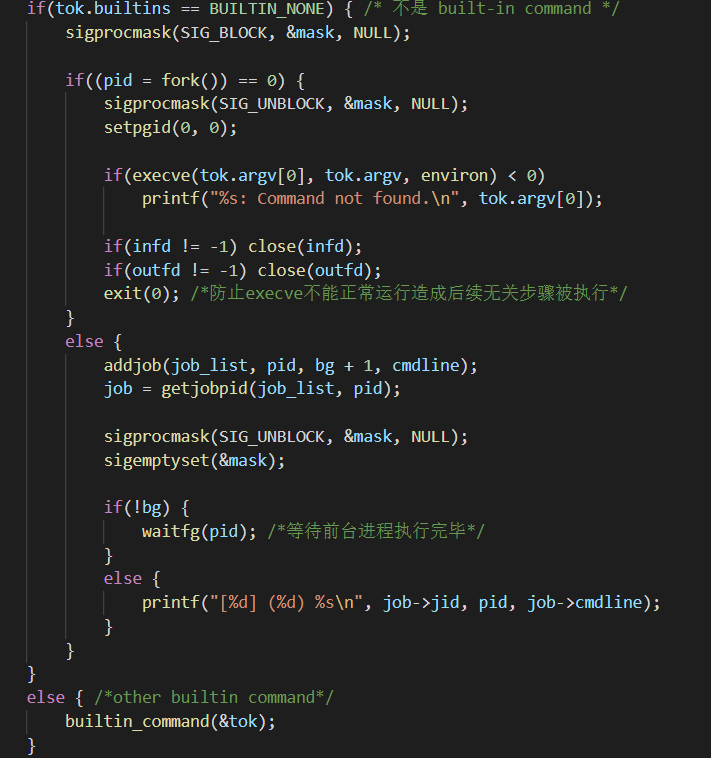
我交上去的程序使用的等待前台进程执行完毕的语句使用的是sleep()而不是sigsuspend(&mask)，但是在尝试改成后者之后，我的分数马上掉到了92，之后又因为一次手滑，还收获了一次0分（因为没有正确编译），最后我放弃了

sleep(0)就sleep(0)吧，大不了扣两分style分……

printf记得换成sio\_put，errno记得交接一下，然后凭喜好加一些错误状态的输出，差不多就这样了

哈哈哈哈哈哈哈哈，写到这里好像又把lab成功了一次！成就感真的爆炸！

这次lab满意就满意在，我超越了自己原来的代码风格——就是拖拉分散，最后修成的程序感觉还是比之前自己写的都要紧凑很多，比如eval这里：

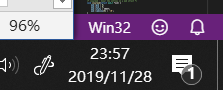


就比最初版本少了很多行。

最后的遗憾就是sleep(0)了……可能自己写的程序还是有问题吧……不过能达到现在我还是很满足了，之后自己重改的版本又出现了新的问题，就不列出了。

同时也是第一次跟室友一起讨论bug！真的是很不一样的感觉

正好，停笔的时候离lab截止只有2分钟了



那就——Everything has the beginning has the end

Malloc lab加油吧！！！

zzy于2019年11月28日星期四，23：58