**操作系统课程设计实验报告**

——实验一：shell实验

负责人姓名：王震

学号：14061131

日期：2016.4.1

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 王震 | 14061131 | 实验一 |
| 2 | 刘铮 | 14061139 | 实验二 |
| 3 | 周子杨 | 14061126 | 实验三 |
| 4 | 曾祥楷 | 14061138 | 实验四 |

目录

[1.实验目的 4](#_Toc446710167)

[2.需求说明 4](#_Toc446710168)

[2.1基本要求 4](#_Toc446710169)

[2.2 提高要求 4](#_Toc446710170)

[2.3 完成情况 4](#_Toc446710171)

[3.设计说明 5](#_Toc446710172)

[3.1 程序流程图 5](#_Toc446710173)

[3.2基本要求实现说明 5](#_Toc446710174)

[3.3 提高要求实现说明 5](#_Toc446710175)

[4.收获和感想 5](#_Toc446710176)

[5.收获和感想 5](#_Toc446710177)

# 1.实验目的

1. 学习Linux相关软件工具的使用（如gcc、gdb和make）
2. 熟悉使用Linux中YACC工具进行语法分析的基本方法
3. 运用man帮助手册查询相关命令
4. 理解并发程序的同步问题
5. 学习POSIX/UNIX系统调用的使用
6. 掌握进程控制和进程间通信的方法

# 2.需求说明

## 2.1基本要求

1.程序能够正常运行

2.能够执行fg、bg、cd、history、exit等内部命令

3.能够执行外部程序命令，命令可以带参数

4.使用I/O重定向

5.支持前后台作业，提供作业控制功能，包括打印作业的清单，改变当前运行作业的前台/后台状态，以及控制作业的挂起、中止和继续运

## 2.2 提高要求

1.尝试对YACC语法分析的文法进行进一步的修改与完善

2.尝试在Linux下将Lex和YACC结合起来使用进行词法和语法分析

3.对其他常用的内部命令进行实现，并可以尝试考虑对通配符的支持与实现

4.实现对管道的支持

5.考虑并实现组合键ctrl+c命令

## 2.3 完成情况

【简述实验完成过程】。完成了以下功能：

【简述实验完成过程】。完成了以下功能：

1. 能够执行fg、bg、cd、history、exit、echo、which等内部命令

2. 能够执行外部程序命令，命令可以带参数

3. 使用I/O重定向

4. 支持前后台作业，提供作业控制功能，包括打印作业的清单，改变当前运行作业的前台/后台状态，以及控制作业的挂起、中止和继续运行

5. 用yacc代替了handleSimpleCmdStr函数，添加myShell.lex代替了yylex函数

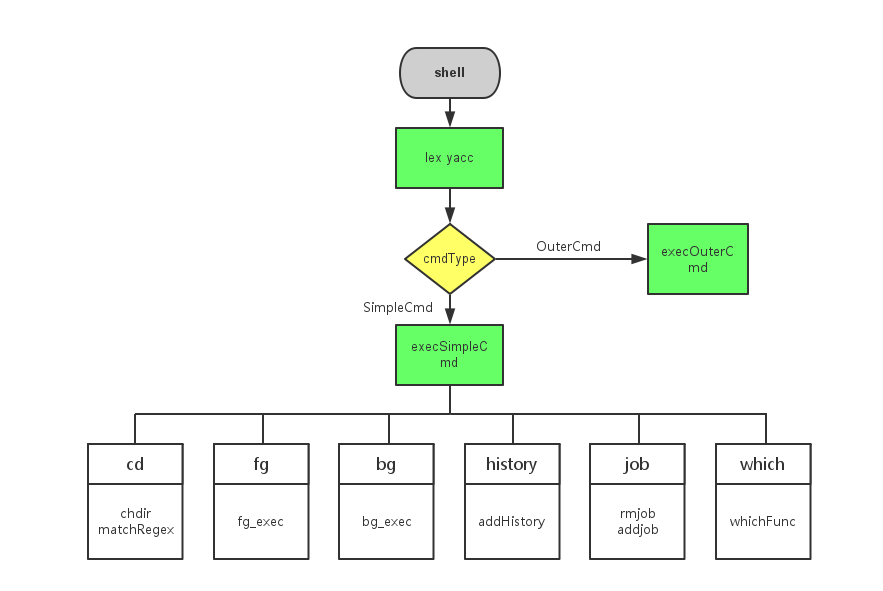
6. 对cd命令实现了\*通配符支持

7. 实现对管道的支持

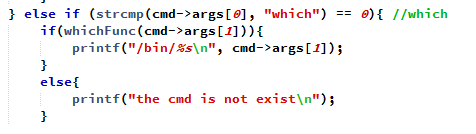
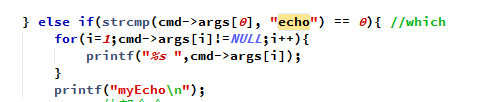
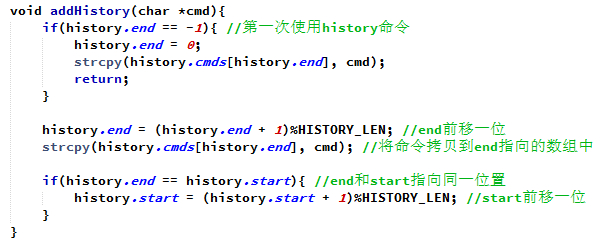
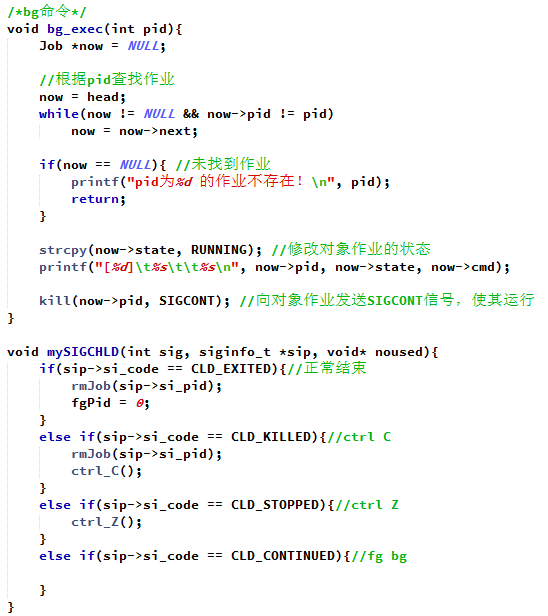
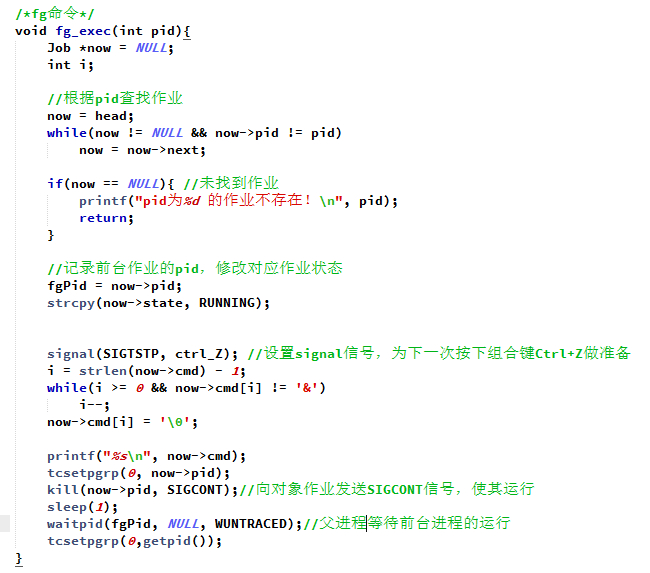
8. 组合键ctrl+c命令

# 3.设计说明

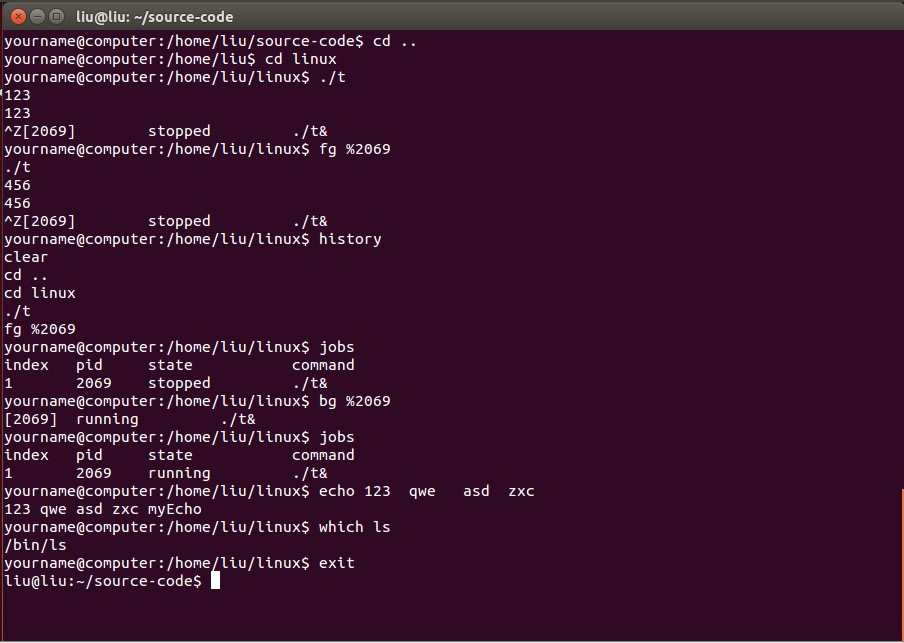
## 3.1 程序流程图



## 3.2基本要求实现说明



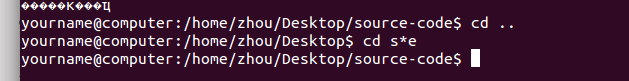
命令测试：



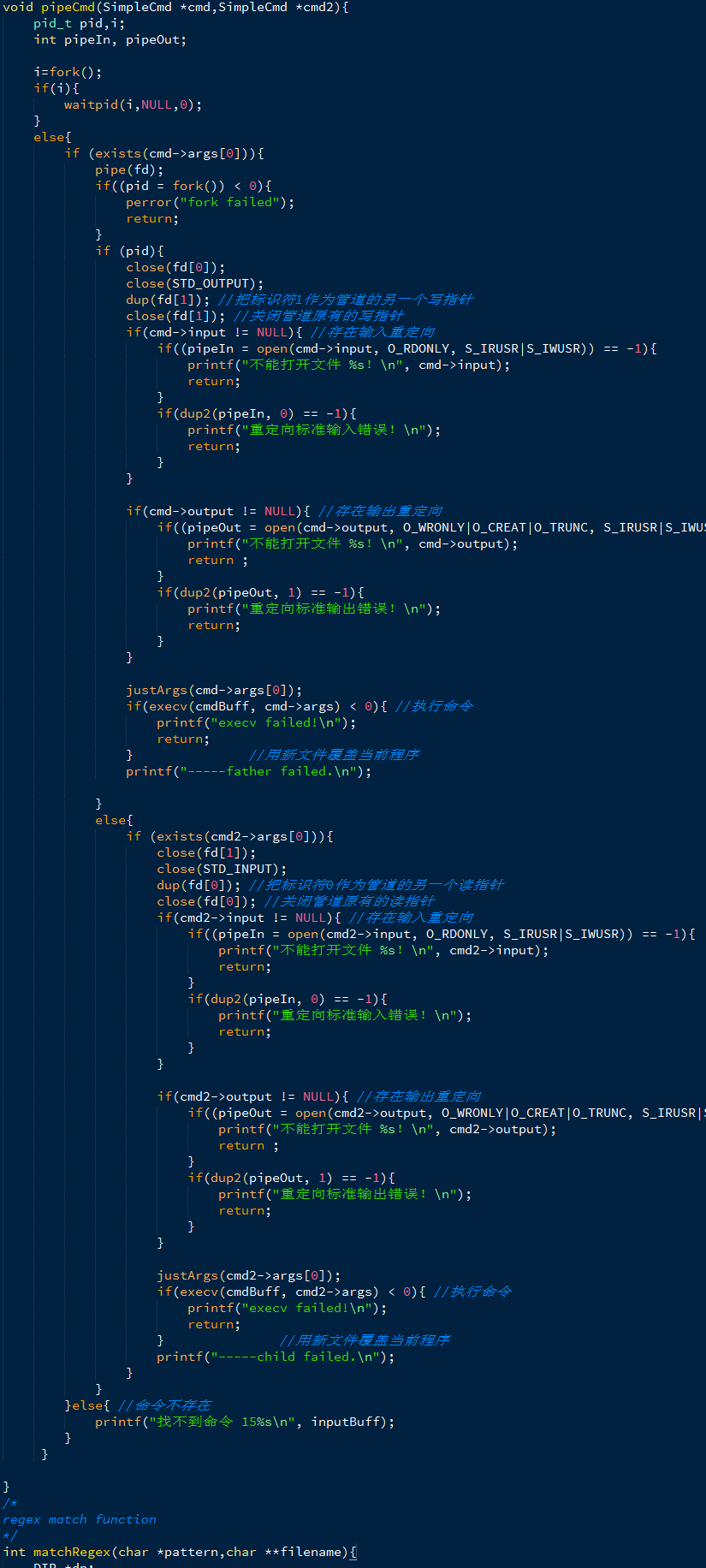
## 3.3 提高要求实现说明

6. cd通配符

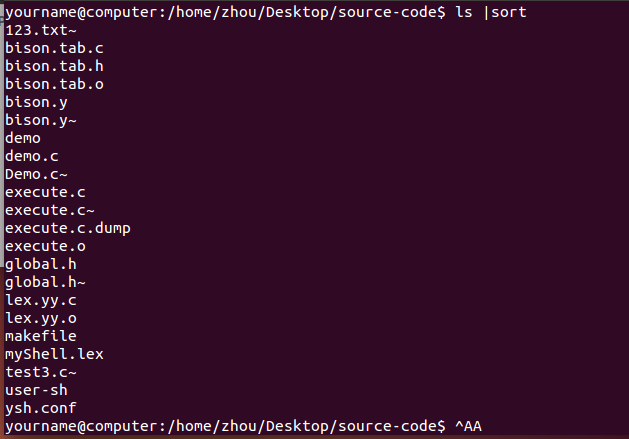
代码图



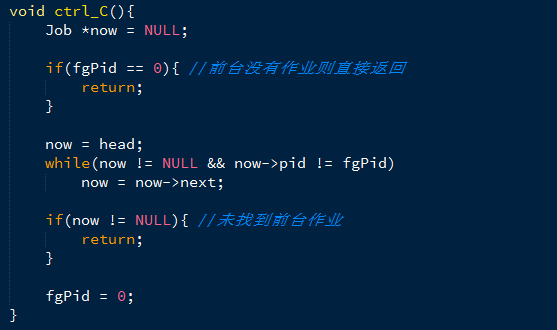
实现图

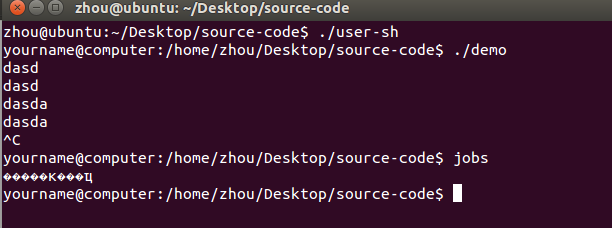
 7. 管道

代码图



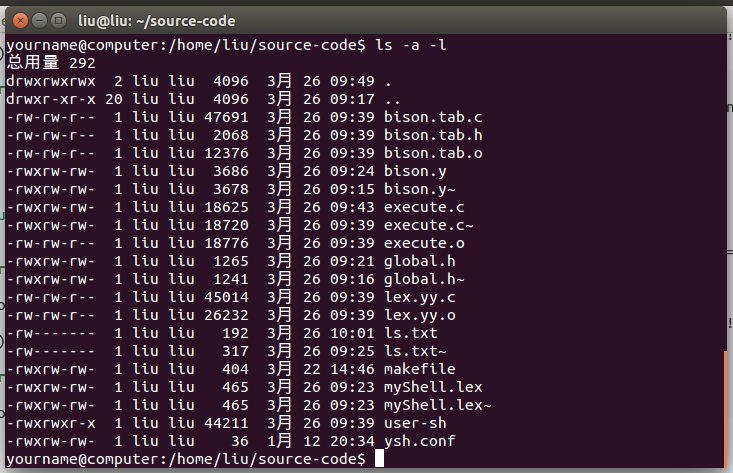
实现图

8. 组合键ctrl+c命令

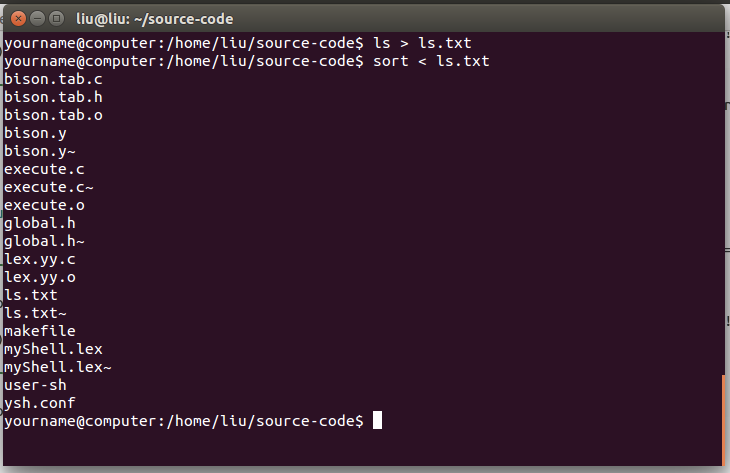
代码图

实现图

9.执行外部命令带参数

：

10.使用I/O重定向：



11.yacc与lex：

lex是语法分析器，其实是使用正则表达式来代替手写C代码分析词法，达到简化分析的目的。

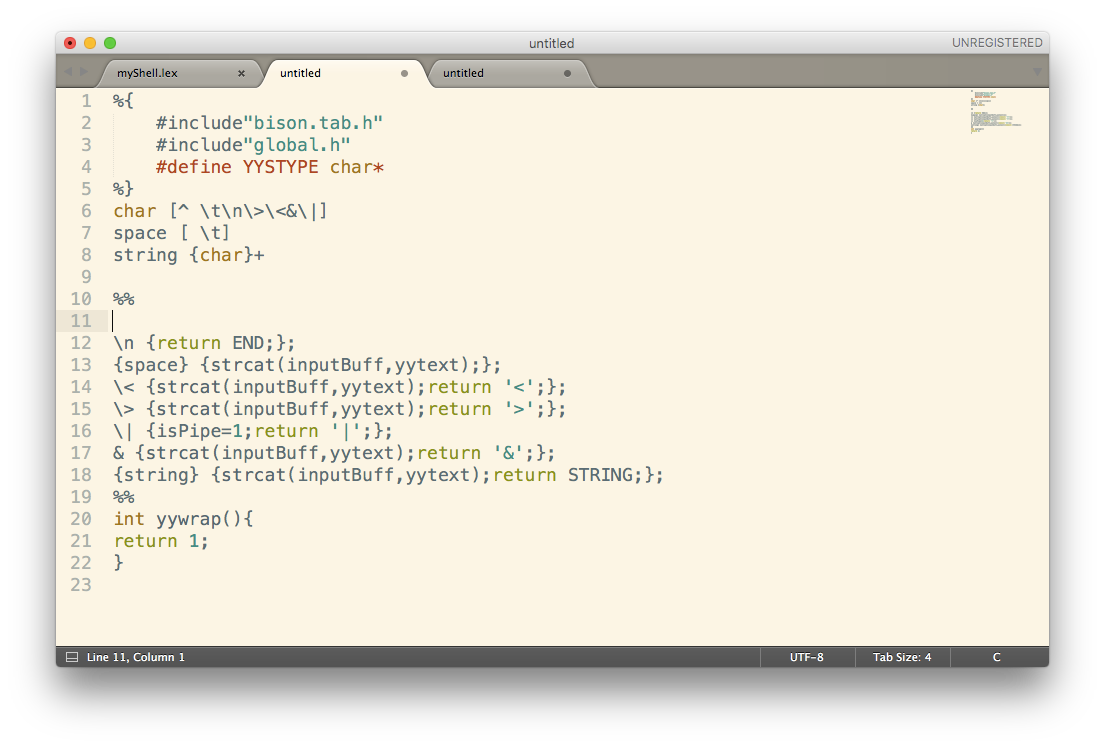
首先是lex，最一开始能想到的就是把空字符串略去，特殊字符（’>’ ‘<‘ ‘|’ ‘&’)分开处理，其它作为字符串输出 。



这样处理之后，发现匹配不能停止。原因是.y 里面的args递归定义，是可以一直匹配STRING的，而原来的.y因为分析的是字符串，在字符串结尾之后就返回了0，结束了分析。因为我们的每一行命令是以回车结束的，因此可以想到当lex读到’\n’时返回END来结束。

可以正常匹配正常结束了。然后又发现一个问题，execute()里面执行命令是要靠inputBuffer的一个数组传进去所有读进去的字符串，然而这里在标准输入里面读进去之后给lex进行分析，inputBuffer里面是没有东西的。这时候就要相办法把所有读到的东西放到inputBuffer里面。

lex里面有一个变量yytext，在每次匹配成功后，会把匹配到的字符都存入yytext里面。由此可想到，只要每次把匹配成功的字符都接到inputBuffer后面，inputBuffer即存入了所有输入的字符串。



至此lex添加完毕，.y文件见**附图1**。shell可以运行。代码中.y的error标记是为了匹配错误时不重复输出错误信息和提示符设置的。输出提示符后的 c=getchar()!=-1是为了去掉后台执行一些打印命令后，下一次读入会失败。usleep是为了等待一些输出较快的后台命令（虽然去掉没有错，但是输出对于较快的后台命令，比如ls &可以保证在其输出完后再打印提示符，好看一些）。

这里又注意到.y里面虽然已经在词法分析方面分开了命令、输入输出重定向、前后台等等，但是.y里面实际上没有进行任何操作，命令的分析最终还是在execute.c 里面的handleSimpleCmdStr函数里实现的，因此就想到可以改进.y代替冗长的handleSimpleCmdStr。

首先是按照handleSimpleCmdStr里面的思路正常地把管道、重定向、命令及参数分开来。.lex中用yylval返回STRING的值，并在读到’|’是置管道标记位Pipe为1。的Pipe标记是为了区分管道指令与普通指令。添加的见**附图2**。

写完后运行，发现正常指令可以执行，但是管道指令无法执行。调试后发现cmd参数域为空。进一步调试发现，管道命令中，对simpleCmd的两次匹配，是在“simpleCmd '|' simpleCmd ”这条规则匹配完成之后，再进行的两次匹配，即匹配成功后的规则执行是在上面匹配成功后，下面再从底向上执行的。即执行Pipe指令时，cmd的构造因为pipe标记位已经被置为1，else部分一直不执行，cmd构造失败。因此Pipe之外，增加一个done标记位表示cmd 已经构造完毕，构造cmd2。更改后如**附图3**

附图3为代码最终版本。

yacc与lex调试技巧：

上述过程看似简单，但每个步骤都是经过了漫长的调试的。总结一些在yacc与lex中的调试技巧。

lex中，如果没有给他写匹配规则（如一开始空格与制表符我都没有写），lex把没有匹配规则的字符放到标准输出。这一点在写代码的时候要保证不想处理的字符也要写匹配规则，要给他一个空的处理方法。调试的时候，这个也是可以加以利用的。

在lex与yacc中，词法或语法匹配成功后，都会执行后面的C代码。可以在C代码中加入printf语句查看某一步骤是否匹配成功。有时，可以通过在lex中打印yytext，或者在lex中给yylval传递不同的值，在.y中，通过$x(x为第x个符号）取回yylval的值并打印来进行调试。

# 4. 实验遇到的问题及解决方法

坑1：

问题：在后台挂起程序之后会卡死

解决办法：shell卡死是因为当父进程发出SIGUSR1信号之后子进程没有及时注册这个信号，导致setgoon()函数没有被执行，导致父进程卡死在while（goon==0）的循环里，所以将子进程的signal(SIGUSR1,setgoon)函数放在刚进入execOuterCmd函数里。

坑2：

问题：运行前台作业挂起之后使用fg放在前台，结果命令提示符先打出来了，没有等在子进程执行完。

解决方法：在fg\_exec()函数里的kill之后加上sleep(1)，使得父进程能够等待到子进程结束才输出



坑3：

问题1：按Ctrl Z的时候会将后台的进程也一起杀死。

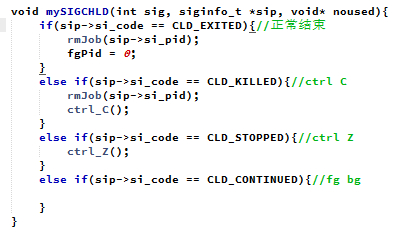
解决办法：用setpgid(0,getpid())将每次fork出来的新的子进程放进新的进程组里，使得对于前台的操作不会影响后台的进程。

问题2：将每个进程独立之后，导致不能获得终端的控制权，无法接受输入。

解决办法：使用tcsetpgrp(0,pid)将终端输入输出的控制权给子进程，然后在子进程结束之后通过tcsetpgrp(0,getpgrp())回收终端的控制权。

问题3：使用tcsetpgrp之后，使用bg命令调到后台之后会导致前台shell崩溃，同时ctrl Z给的是子进程，导致不能使用addJob()。

解决办法：定义一个自己的函数mySIGCHLD()用来统一处理SIGCHLD信号，因为在siginfo\_t这个结构里有一个si\_code信号，当程序正常结束时发出CLD\_EXITED信号，当按ctrl C时发出CLD\_KILLED信号，当按ctrl Z时发出CLD\_STOPPED信号，通过对不同信号的区分处理来判断执行哪一个函数。



同时在init里注册这个信号，在父进程里忽略SIGTSTP和SIGINT信号，在子进程里注册SIGTSTP和SIGINT信号。

# 5.收获和感想

收获：

通过这次shell实验，我们明白了系统的shell是怎么实现的，明白了一些系统调用的使用方法例如signal等等，以及进程之间是如何通信并且共享资源的，也知道了在终端里前后台程序的区别。除了这些，这次也是我们几个同学第一次合作完成一个作业，通过相互之间的合作学习，互相帮助，最终完成了任务，为我们的接下去的几次合作实验打下来良好的基础。同时，这次的实验用到了好多老师上课的时候没有教过的知识，我们通过自己上网搜索答案，自己学习来解决问题，很好地锻炼了独立自主的学习能力。

感想：

以前在Linux里用shell的时候感觉特别神奇，不知道为什么通过简单的终端就可以实验那么多的功能，通过这次实验，我们明白了shell的基本工作原理，明白了无论看起来多么酷炫复杂的功能实现，都是用代码一砖一瓦搭建而成的，而我们也是可以通过学习来实现属于自己的程序。

**会议记录**

我们小组每周开一次会议

第一次会议

阅读源代码，确定每个人的工作，确定这次实验的难点重点进行重点学习。

第二次会议

交流完成的基本功能，发现在fg，bg的时候有一些奇怪的bug，组员们进行集中讨论，面对面的交流debug，最终解决了这些问题，最后将完成的代码进行合并。并且确定将要做哪些提高要求。

第三次会议

交流完成的提高要求，讨论一下具体实现了哪些功能，遇到了哪些调不出来的bug，进行集中处理，合并代码进行测试，讨论并完成实验报告。