**操作系统课程设计实验报告**

——实验一：shell实验

负责人姓名：陈鸿超

学号：14061216

日期：2016.3.23

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 陈鸿超 | 14061216 | 整体分工与策划，Bug修复 |
| 2 | 刘润泽 | 14061120 | 管道，通配符，CTRL+C的添加 |
| 3 | 叶旭诚 | 14061215 | Yacc与lex的编写 |
| 4 | 杨佳琦 | 14061124 | 内部命令的添加 |

目录

[1.实验目的 4](#_Toc446001831)

[2.需求说明 4](#_Toc446001832)

[2.1基本要求 4](#_Toc446001833)

[2.2 提高要求 4](#_Toc446001834)

[2.3 完成情况 4](#_Toc446001835)

[3.设计说明 5](#_Toc446001836)

[3.1 程序流程图 5](#_Toc446001837)

[3.2基本要求实现说明 5](#_Toc446001838)

[3.3 提高要求实现说明 5](#_Toc446001839)

[4.问题与解决 5](#_Toc446001840)

5.收获和感想……………………………………………………………………………………………………………5

# 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、中止和继续运行

## 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：

1.添加CTRL+C命令

**2.添加了管道线，实现管道操作**

3.BUG修复：

(1).修复了无法运行后台命令的BUG

(2).修复了执行fg命令时父进程比子进程先结束导致输出混乱的BUG

(3).修复了CTRL+C没有彻底的删除进程，作业依然残留在jobs中的BUG

(4).修复了fg命令重启的进程无法再被Ctrl Z/C中断的错误

(5).修复了后台命令也会被CTRL+Z/C挂起或终止的BUG

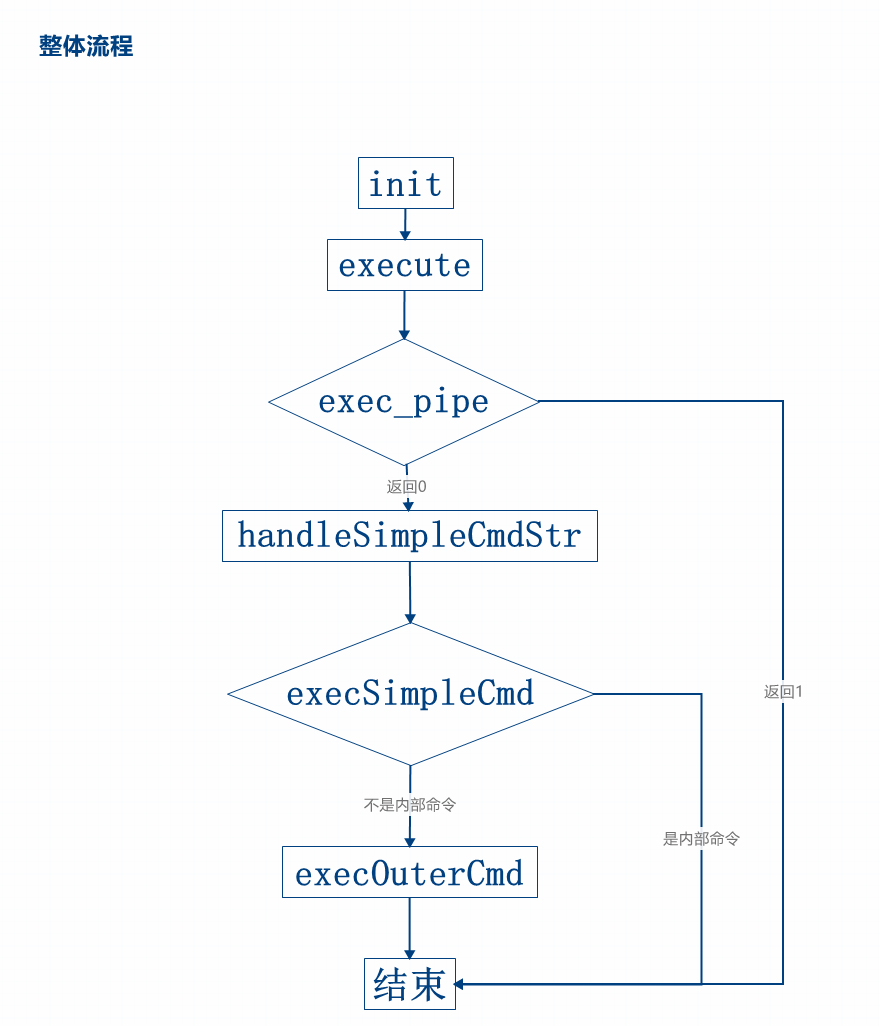
**4.增加了对通配符的支持。**

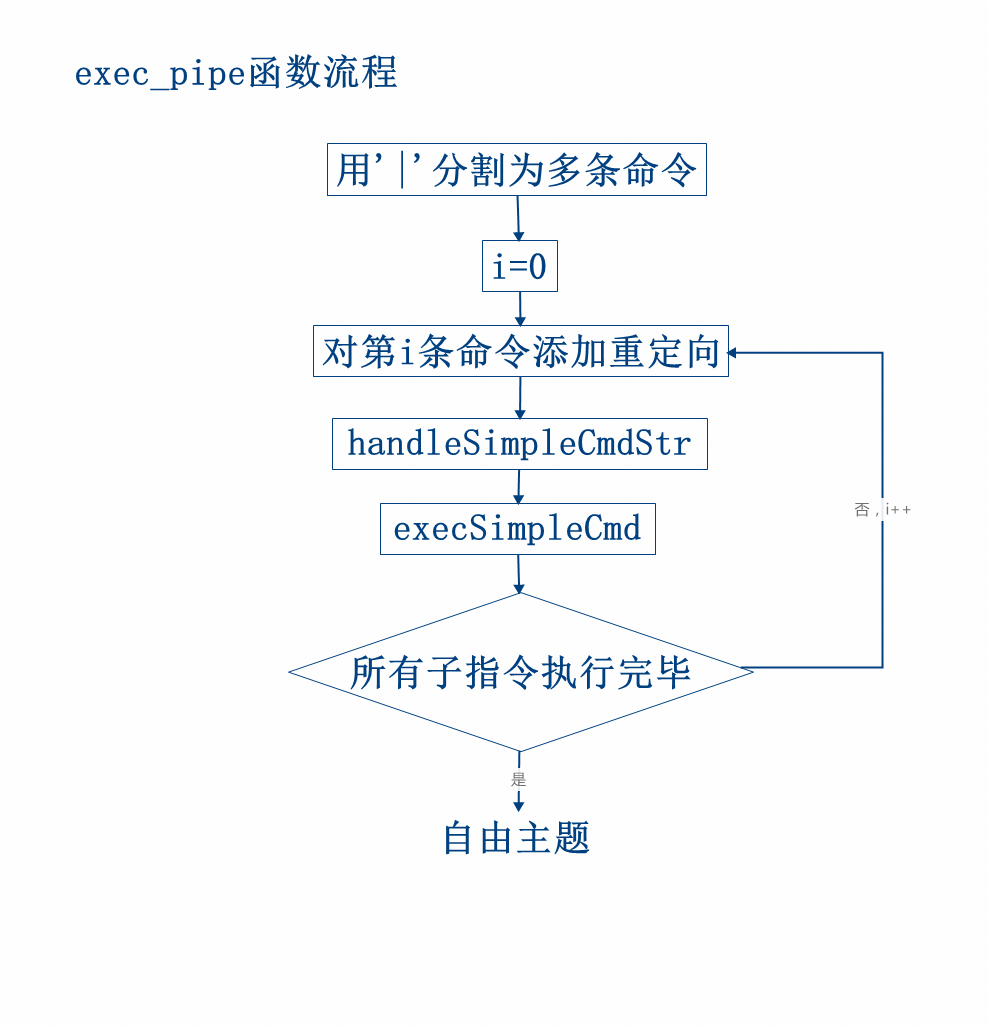
5.添加了一些内部命令。

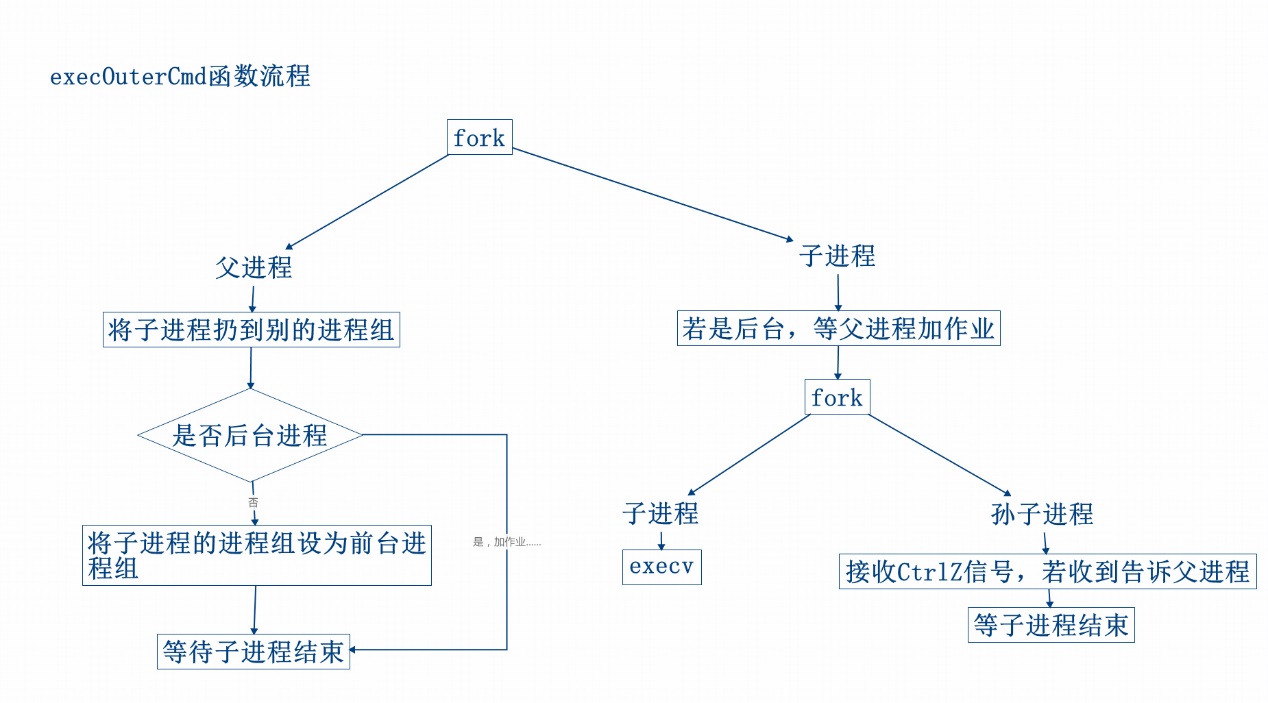
**6.利用yacc和lex重写了语法和词法分析,并对部分错误输入进行了提示。**

# 3.设计说明

## 3.1 程序流程图





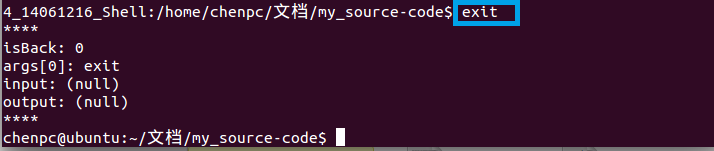


## 3.2基本要求实现说明

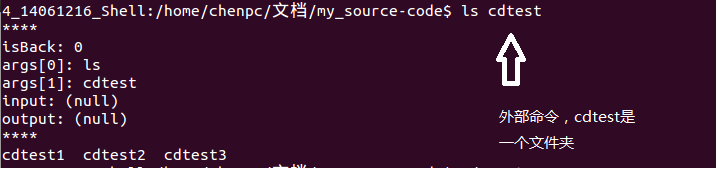
/\*\*\*前面几个要求源代码就可以实现，就不贴代码了\*\*\*/

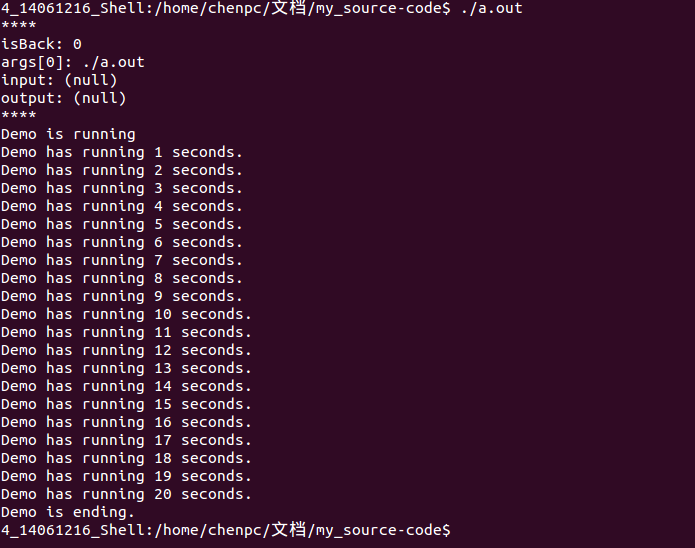
1. 程序能够正常运行

2. 能够执行fg、 bg、 cd、 history、 exit等内部命令

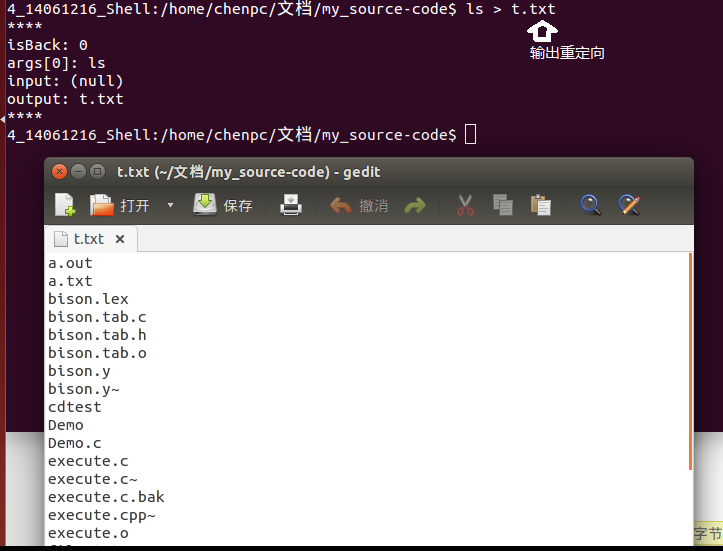


3. 能够执行外部程序命令，命令可以带参数





4. 使用I/O重定向

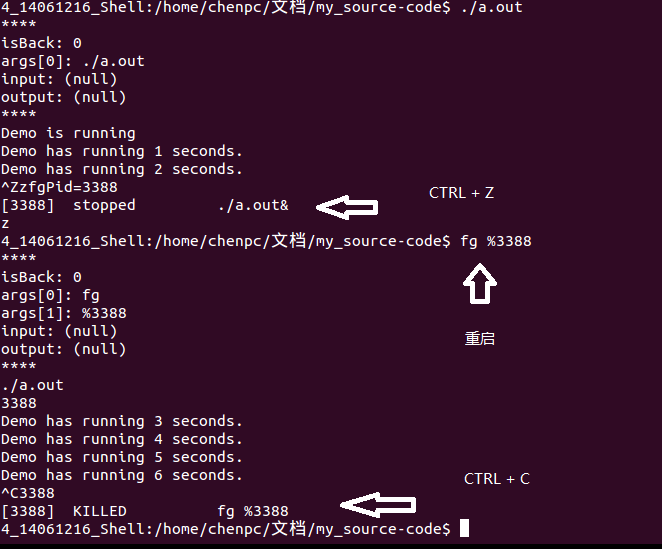


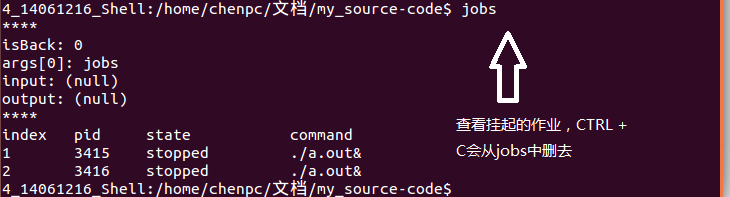
5. 支持前后台作业，提供作业控制功能，包括打印作业

的清单，改变当前运行作业的前台/后台状态，以及控

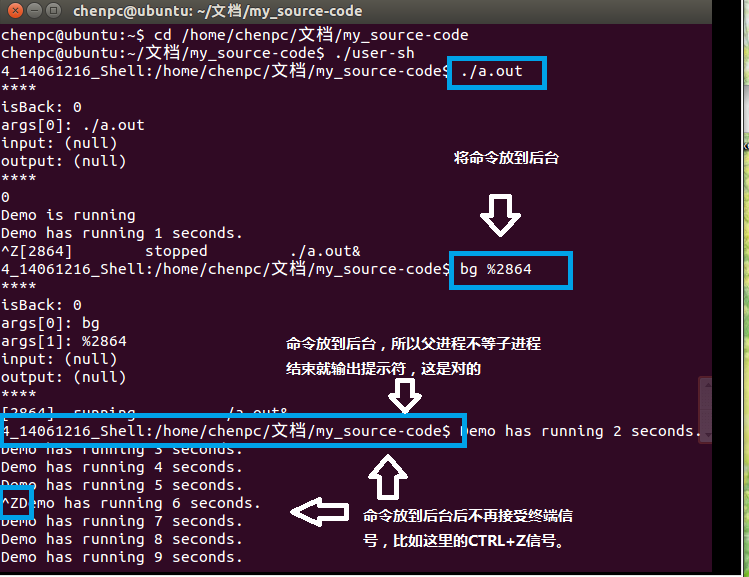
制作业的挂起、中止和继续运行

挂起，重启，终止，查看



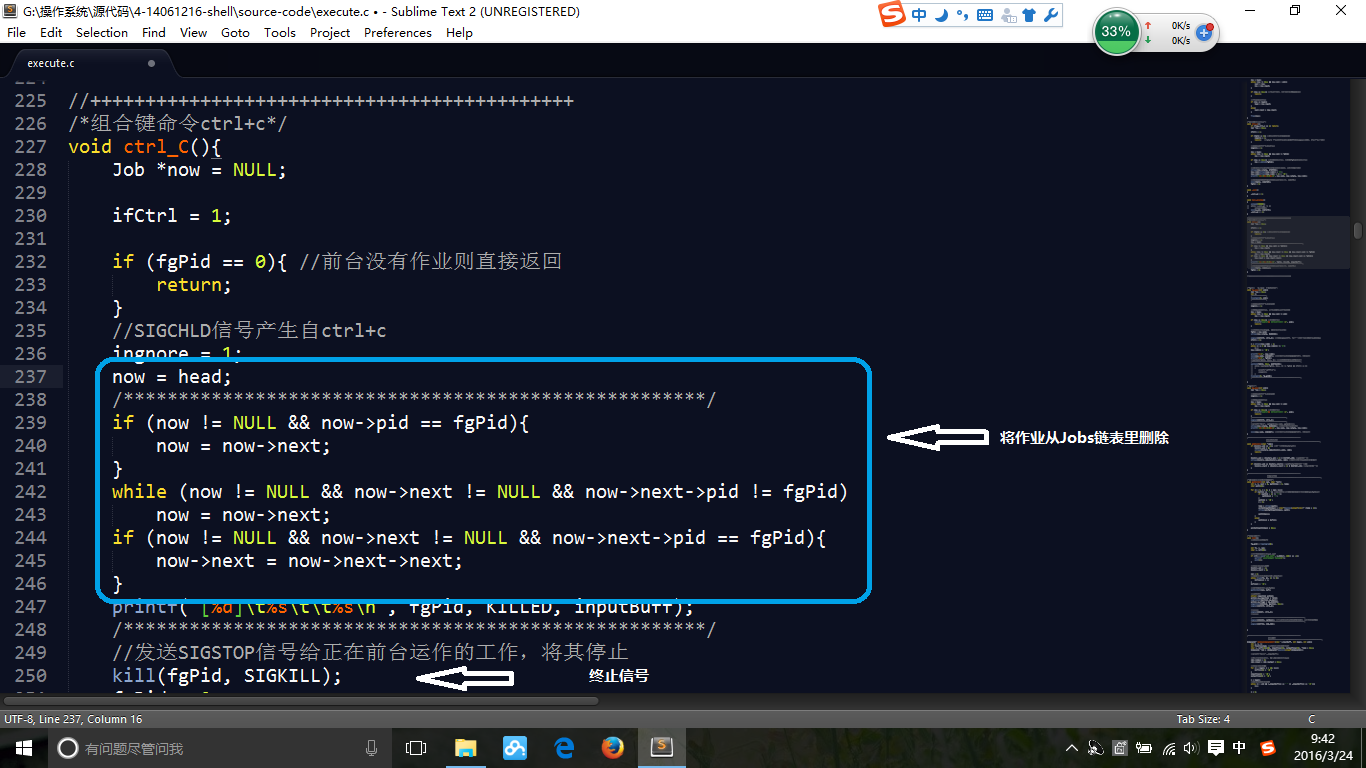


切换后台

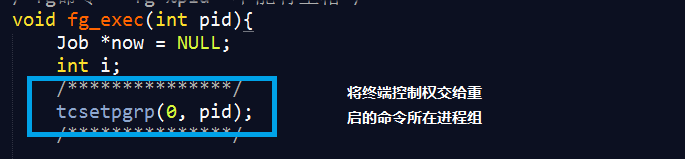


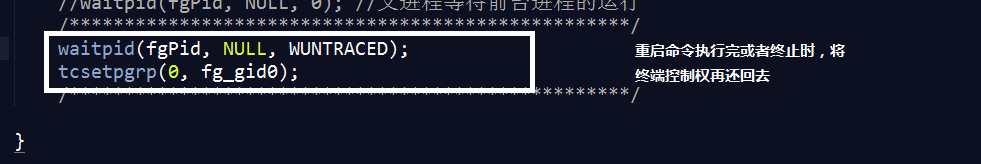
代码：

CTRL + C

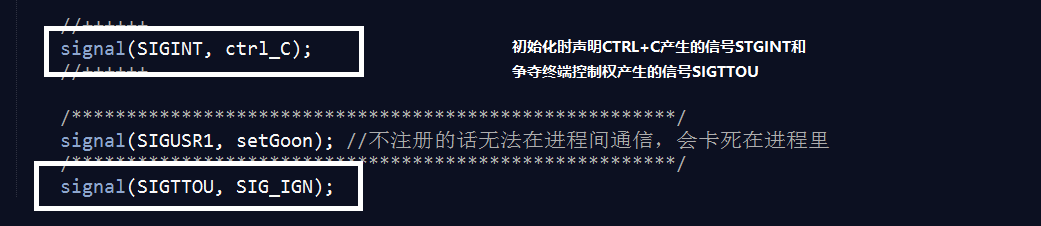


Fg





信号声明：



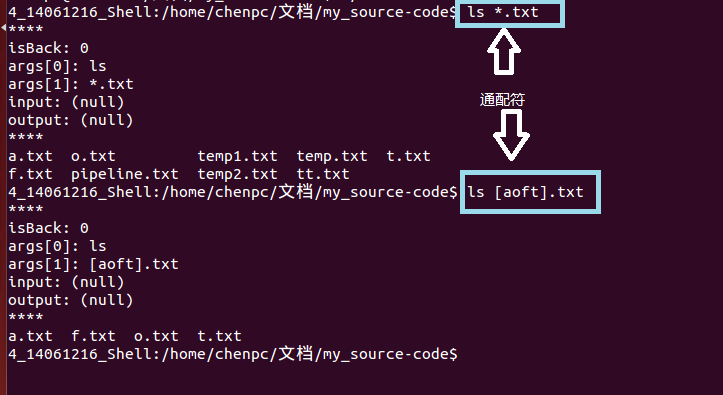
终端控制权切换：

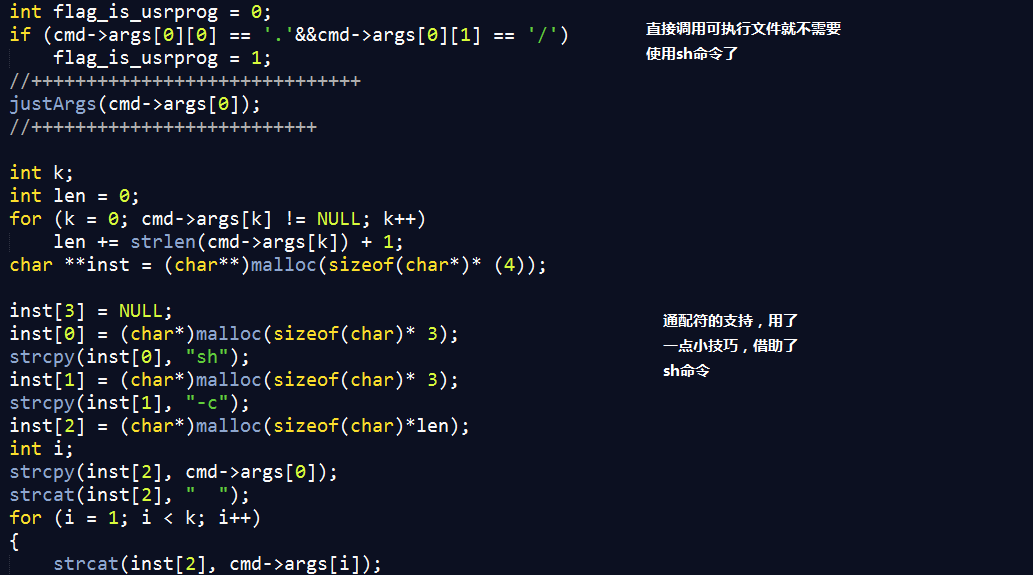


## 3.3 提高要求实现说明

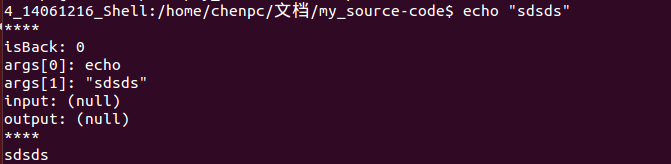
1. 对其他常用的内部命令进行实现，并可以尝试考虑对通配符的支持与实现

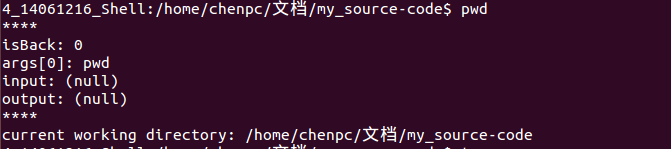
通配符：

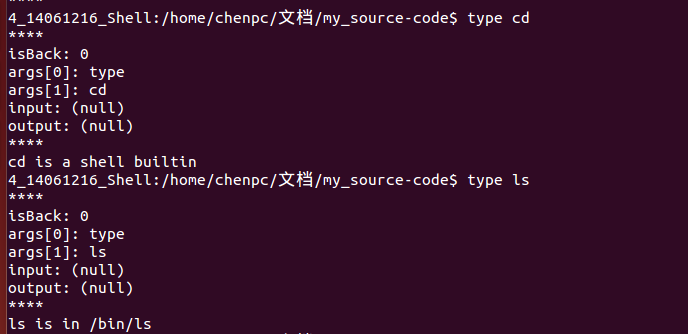


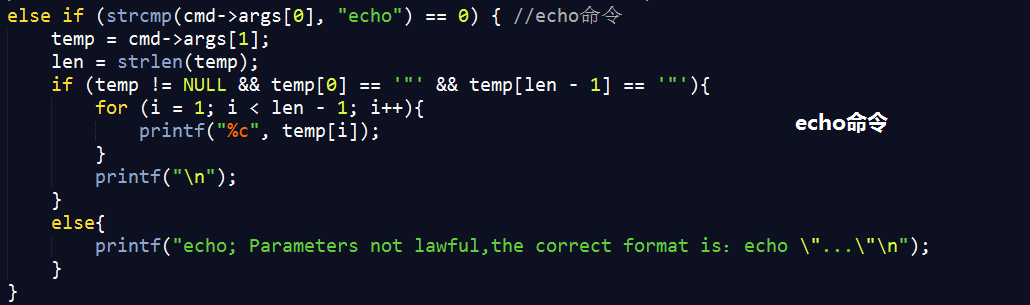


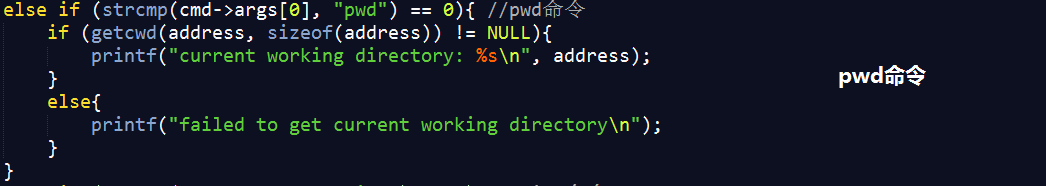
其他内部命令：

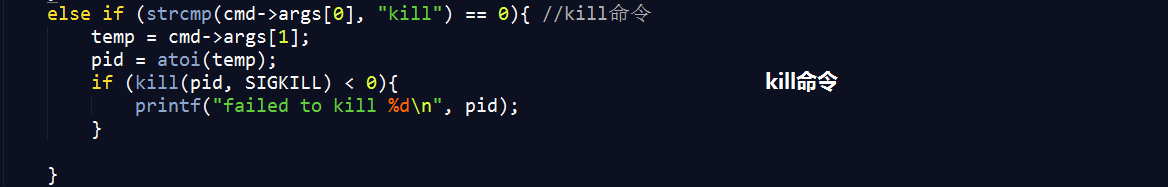


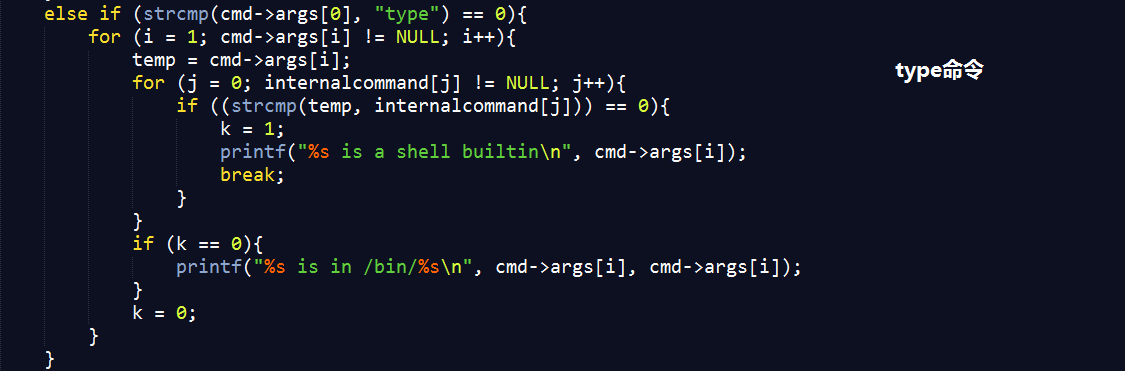




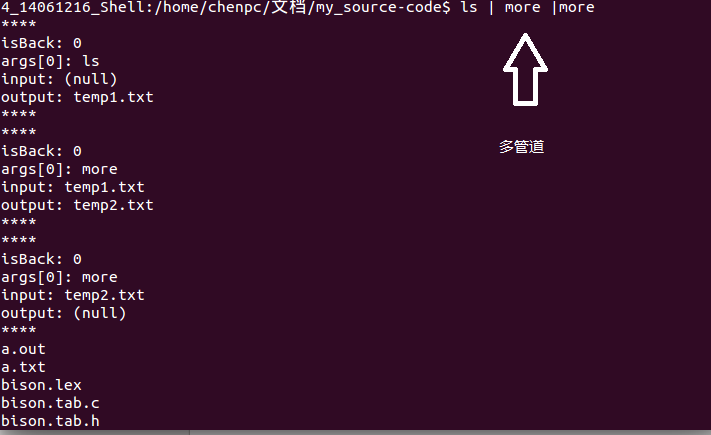
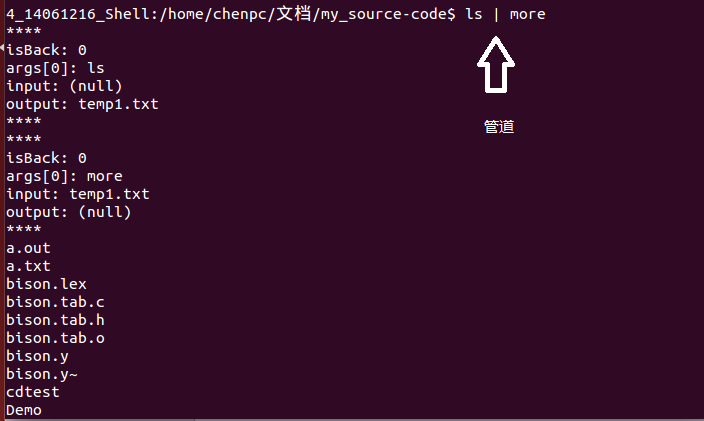




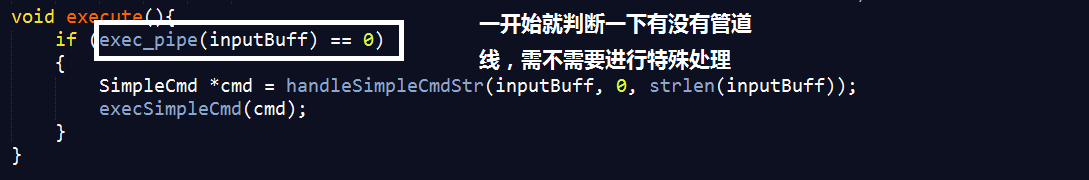


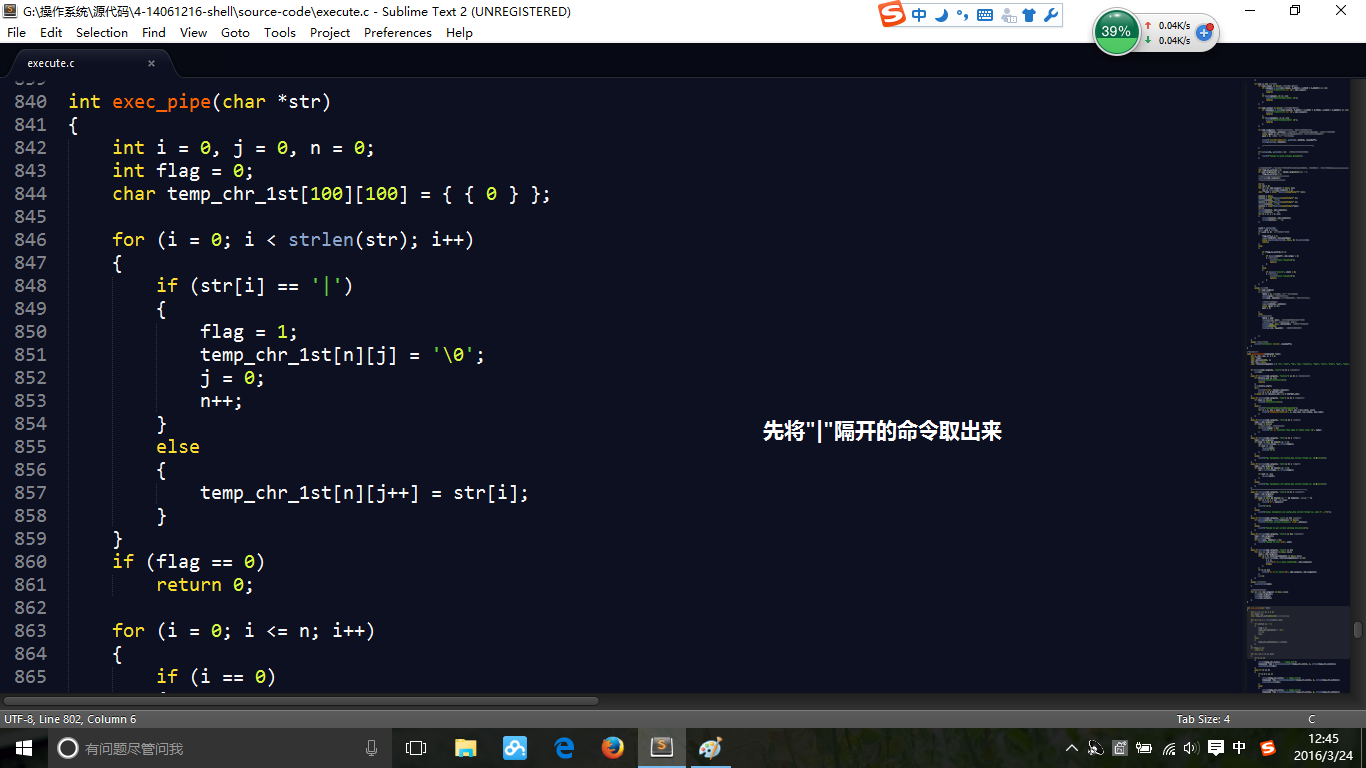


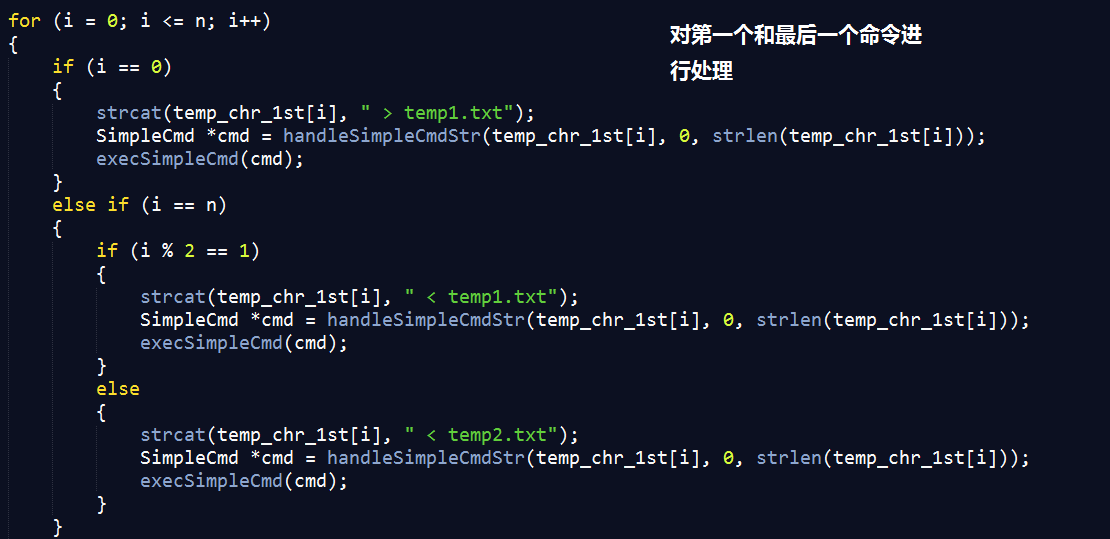
2. 实现对管道的支持

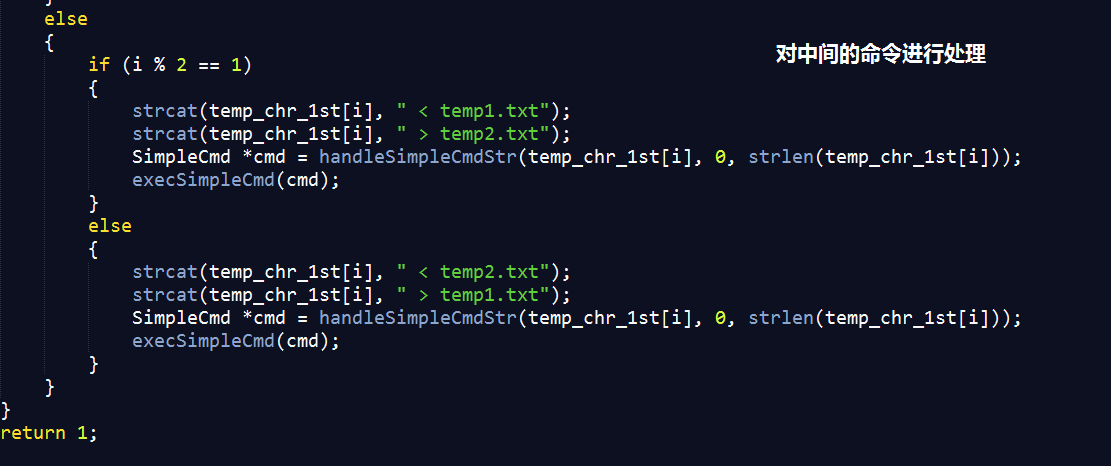


代码：



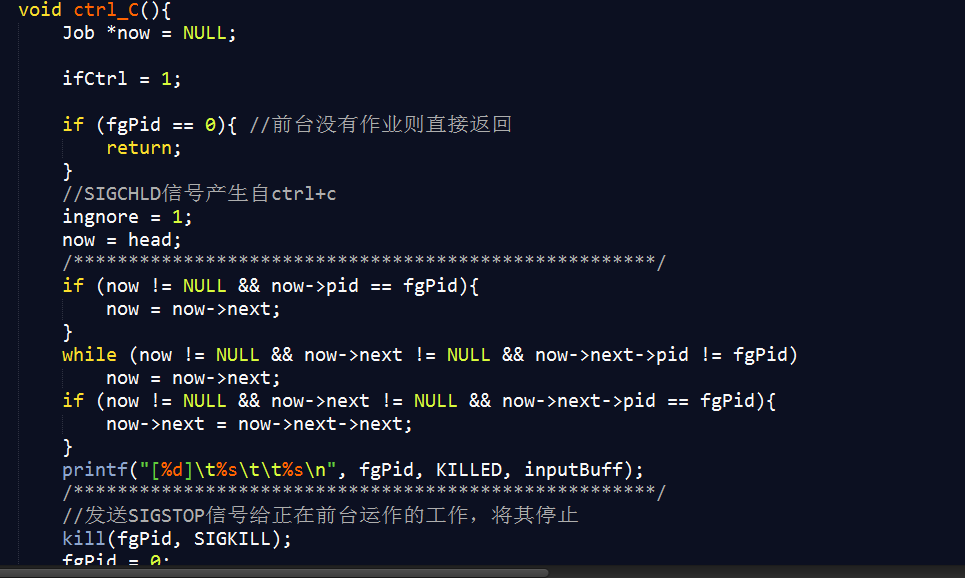






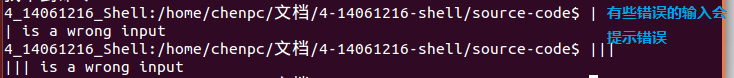
3. 考虑并实现组合键ctrl+c命令





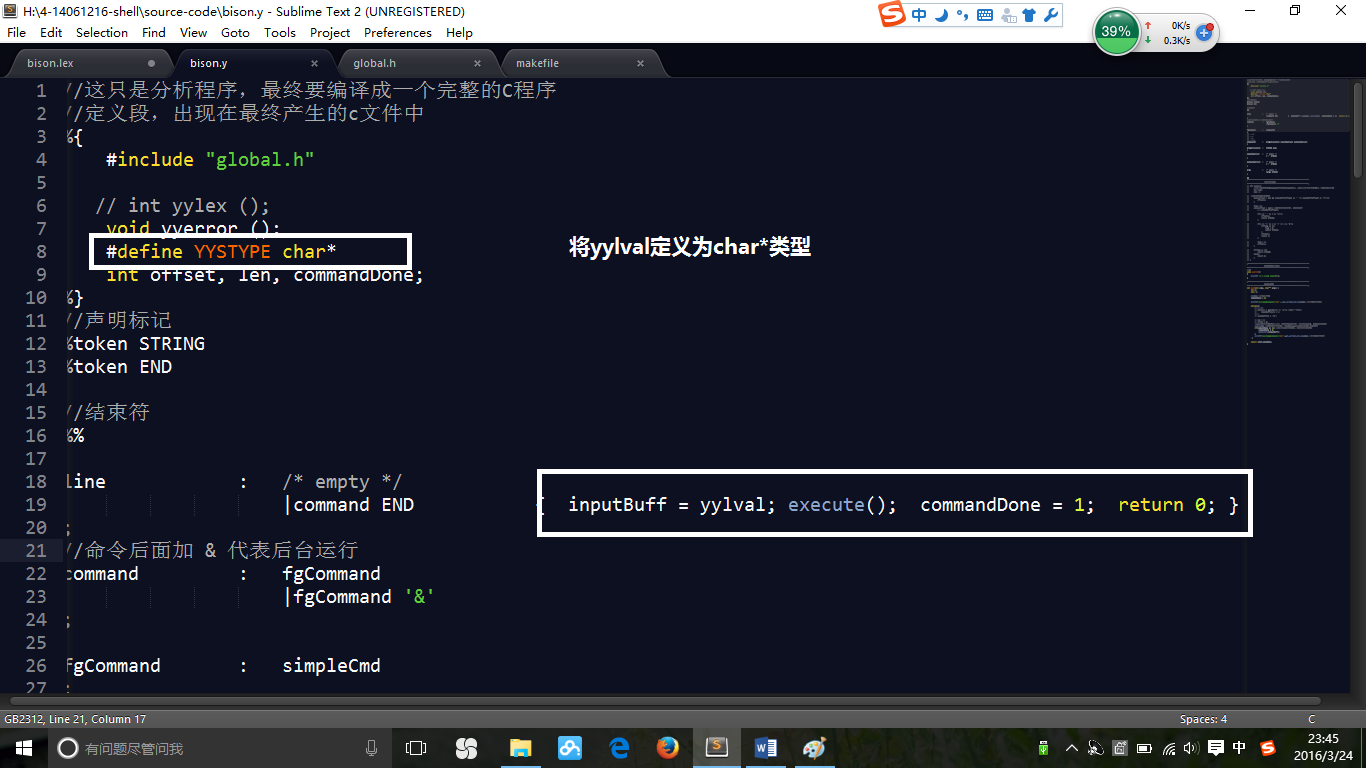
4.yacc与lex的实现





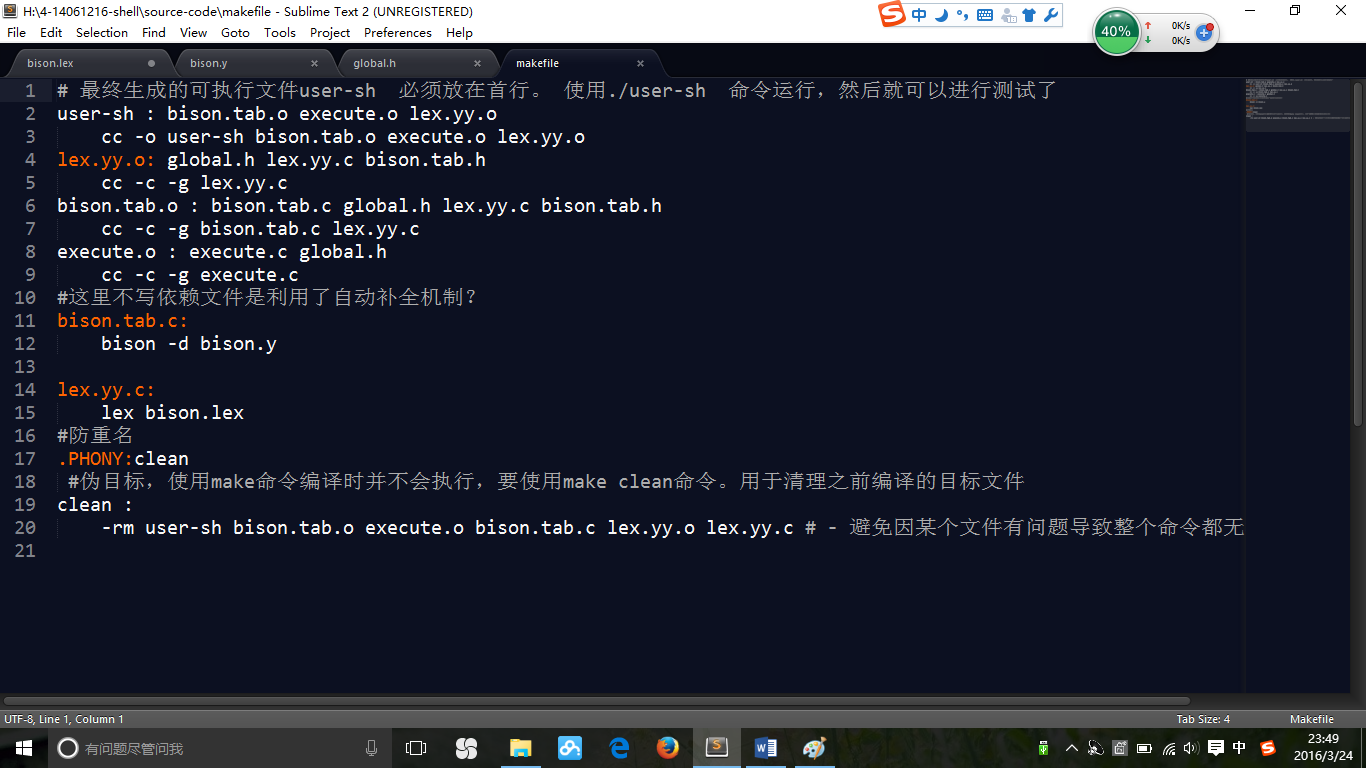
代码：







修改了makefile



# 4.问题与解决

1.后台命令无法运行

原因：

父进程向子进程发送SIGUSR1信号时，子进程还未创建成功，所以子进程会按系统默认的方式响应SIGUSR1信号，而不是setgoon()。而系统默认的响应方式是终止进程，所以程序会卡死在父进程的循环里。

解决方法：

在初始化时就声明一下signal(SIGUSR1, setGoon);这样子进程在创建过程中也可以按我们的要求响应SIGUSR1信号。



2. 执行fg命令时父进程比子进程先结束导致输出混乱。

原因：

Fg程序里waitpid()等待子进程结束的时候，因为子进程还没有完全启动，所以等待失败，返回-1.父进程就不再等子进程执行完。

解决方法：

使用while循环，一直等到子进程结束。



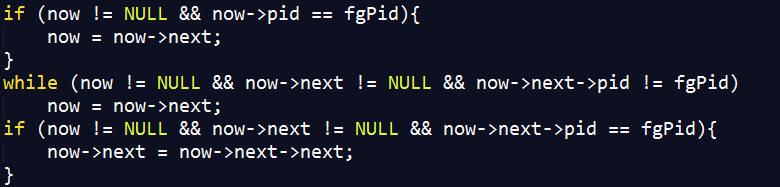
3. CTRL+C没有彻底的删除进程，作业依然残留在jobs中

原因：

写CTRL+C时是按CTRL+Z来写的，所以没有从jobs中把进程号删去。

解决方法：

在CTRL+C中对jobs链表进行遍历判断，从中删去当前的进程。



4. fg命令重启的进程无法再被Ctrl Z/C中断

原因：

因为之前要等待子进程创建成功，我们用了一个while循环。所以在重启的进程运行过程中使用CTRL+Z/C，while依旧在循环.

解决方法：

为CTRL +Z/C设置一个标志信号ifCtrl，以便能从while循环里跳出来。



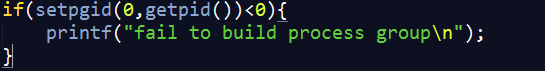
5. 后台命令也会被CTRL+Z/C挂起或终止

原因：

虽然这些命令是后台命令，可他们其实也在前台，都会接到CTRL信号

解决方法：

把后台命令全部放到相应的新进程组里，这样他们就接收不到终端的信号了，也就不会被挂起或终止。



**6.为实现bg功能，进行了多线程调控。(这个调了好久=\_=)**

原因：

Bg里面无法把之前挂起子进程分配到其他进程组中，很奇怪。

解决方法：

(1).把所有的外部命令都各自分配到不同的进程组中。

(2)如果是前台命令，就把终端控制权交给子进程所在进程组，执行完再把控制权拿回来。

(3)这里带来一个问题，也就是执行子进程时，execv会覆盖掉子进程原来的东西，这时用CTRL+Z的话，子进程就会执行默认的处理方式，挂起进程。而父进程因为失去了终端的控制权，所以无法接受到CTRL+Z信号，也就没有把子进程加入到jobs中。

(4)为了解决这一问题，我们又在子进程里fork()了一个孙子进程。孙子进程不会被execv覆盖掉。当CTRL+Z时，孙子进程就给父进程发送一个SIGTSTP信号，父进程就可以执行Ctrl\_Z()函数，把子进程加入到jobs中去了。

(5)这里孙子进程要一直挂在那里，一直到子进程结束，以便子进程多次挂起和重启时都可以向父进程发送SIGTSTP信号。

(6)在孙子进程给父进程发送SIGTSTP信号前，父进程可能已经结束，然后会先输出提示信息，再执行CTRL+Z。这里当然可以父进程<---->子进程<---->孙子进程这样进行传信号来调控一下时间，太繁琐了。因为时间有点紧，我们就直接在父进程的最后sleep一会，等一下Ctrl\_Z()函数执行完就好了。

# 5.收获和感想

这次实验收获最大，当然被坑最惨的就是进程了。与以前写的程序相比，进程多了时序上的不确定性，并不一定会按我们的思路顺序执行。如果不设计好进程间的通信，因为进程的不同步可能会带来很多莫名其妙的Bug。下面就对这次的收获做一些总结，都是经验之谈，可能会有一些纰漏。

1. 进程间通信

关于进程间的通信，最重要的就是两个函数signal和kill。

(一)signal()函数

1. 格式：signal(信号名,处理方式)
2. Signal函数的作用在于可以接受到一些特定的信号，比如CTRL+Z产生的SIGTSTP信号，CTRL+C的SIGINT信号，然后按照第二个参数所指定的方式进行一些响应处理操作。
3. 假如不使用signal声明信号的话，那么当进程收到这些信号时就会按照系统默认的处理方式进行处理，大部分都是终止进程。
4. Signal的第二个参数可以是系统自带的参数，比如SIG\_IGN(忽视这个信号)和SIG\_DFL(按默认方式响应)，更多的则是我们自己定义的函数，这样就可以根据这些信号做我们想做的事。
5. 一定要确保signal信号声明在信号到来前，不然就没有作用了。而且，一次声明只能响应一次信号，所以如果想多次响应的话，每次响应结束都有再次声明。
6. 下面例几个常用的信号

SIGTSTP CTRL+Z

SIGINT CTRL+C

SIGTTOU 后台进程组对终端进行写操作，可以用于拿回终端信号

SIGUSR1 用户自定义信号

SIGUSR2 用户自定义信号

(二)kill()函数

1. 格式：kill(pid,sig)
2. 作用：向任意进程pid发送信号sig。与signal()函数相配合，可以实现进程间的通信。
3. 第一个参数pid

pid>0时，pid是信号欲送往的进程的标识。

pid=0时，信号将送往所有与调用kill()的那个进程属同一个使用组的进程。

pid=-1时，信号将送往所有调用进程有权给其发送信号的进程，除了进程1(init)。

pid<-1时，信号将送往以-pid为组标识的进程。

1. 第二个参数sig与signal()函数的信号参数完全相同，就不在介绍。

不过当sig为0时，不发送信号，但系统会进行错误检查，可用来判断进程是否还存在。

1. 返回值：成功执行时，返回0。失败返回-1。
2. 进程组

(一).对进程组的一些基础理解

1. 进程必定属于一个进程组，也只能属于一个进程组。

2. 一个进程组中可以包含多个进程，进程组的生命周期从被创建开始，到其内所有进程终止或离开该组。

3.同时只有一个进程组可以控制终端，称其为前台进程组。

4.每个进程组都有一个组长进程，也就是进程组的第一个进程。也只有组长进程才有权对进程组进行终端控制权切换操作。

(二)进程与进程组相关函数

1.getpid()

获取当前进程或子进程进程号

2.getppid()

获取父进程进程号

3.getpgrp()

获取所在进程组组号

4.tcgetpgrp(int filedes)

获取前台进程组组号,参数是终端的一些代号，不如终端写就是0或STDIN\_FILENO。

5.setpgid(pid,pgid)

(1)将进程pid放入到进程组pgid中，若进程组不存在就会创建他。

(2)几种特殊用法

setpgid(pid,0) 将进程pid放入到进程组号也为pid的组中

setpgid(0,pgid) 将当前进程组号设为pgid

(3) 一个进程只能为它自己或它的子进程设置进程组号，在它的子进程调用了exec函数后，它就不再能改变该子进程的进程组号。否则函数会执行失败，返回-1.

6.tcsetpgrp(int filedes, pgid)

(1)将进程组pgid设为前台进程组，也就是交给他终端控制权。

(2)只有前台进程组的组长进程才能执行此函数。

(3)后台进程执行此函数争夺终端控制权必须先声明信号signal(SIGTTOU,SIG\_IGN)。

(三)终端控制权与进程组之间的一些关系

1.获得终端控制权的进程组叫做前台进程组。

2.只有前台进程组才能收到终端写入的数据。

3.只有且所有前台进程组中的进程都能收到终端产生的CTRL信号

4.不管前台后台进程，都可以对终端进行输出。

三．makefile

1.何为makefile?

Makefile可以理解为为了完成编译任务所需要进行的操作的一个集合，当使用make命令进行编译时，系统会自动寻找makefile文件，并按照它所指定的命令来进行编译。这样的话只要一个命令就可以完成整个编译，而且也很好进行修改。

2.编写规则

(1)每条命令的格式：

Target : prerequisites

Command

Target: 目标文件，也就是这条命令要生成的文件

Prerequisites：依赖文件，生成目标文件所需的文件

Command：生成目标文件所需的Shell命令

(2)一些规定

最终的可执行文件或最高级别的目标文件一定放在第一行

只要有一个依赖文件比目标文件新，就会重新编译

Command一定要以一个Tab键开头

3.变量

Makefile也支持变量的使用，简化代码，方便修改

例：

objects = main.o kbd.o command.o

edit: $(objects)

cc – o edit $(objects)

这里objects就是变量，他代表着main.o kbd.o command.o。

$(objects)就是对变量的引用，代替了三个.o文件

4.伪目标

(1)Makefile中有些目标文件在使用make命令时并不会执行，只有需要在一些特殊的情况下才需要执行他们，这些目标文件就叫做伪目标。

(2)例如伪目标clean，可以通过make clean命令调用，作用是清除之前编译生成的文件

.PHONY:clean

clean :

-rm user-sh bison.tab.o execute.o bison.tab.c lex.yy.o lex.yy.c

.PHONY 是声明符，声明他是一个伪目标，防止与真的目标重名

rm命令前的减号”-”代表如果命令执行过程中遇到问题，忽略他，继续执行。

四．Yacc和Lex

1.思路

在我们这次的OS实验中，主要是撰写lex文件以代替现有.y文件中提供的yylex()函数，以达到词法匹配的任务。因为有了Lex,所以就可以用yylval来代替原来的inputBuff作为程序的输入。

2.知识点

(1) lex和yacc分析过程都是将输入字符串按程序员预先设计好的正则表达式进行匹配，匹配上则进行相应的操作。

(2) yytext是lex内部已经定义好的指针变量，总是指向当前获得匹配的字符串,即lex词法规则匹配上的值。

(3)yylval也是个指针，与yytext不同的是他指向lex所读取的整个数据，yacc也正是通过它来读取lex传过来的数据进行语法分析的。

(4)yylval默认是int\*类型，而我们需要的是char\*类型。这里可以在开头的定义部分通过 #define YYSTYPR char\* 来定义其类型。

3问题与解决

1. 无法编译通过：编译的过程中总是显示excute.c中的许多函数未声明或者是某些文件不存在。

这里首先是没有先编译生成lex.yy.o文件，其次是要把生成的三个.o文件一起编译生成最终的可执行文件，当然还要注意一下makefile中编译目标文件的顺序。

1. lex无法判断输入的结束，一直都在往里面读值。

既然我们不知道他是怎样才能停止读值，那我们就给他设置一个读值结束的标志”\n”。一旦读到这个值，就代表这一次读值结束了。

1. 无法将yylex()读到的值交给main函数。

这个也很好解决，既然之前读的值都保存在inputBuff中，而现在读的值都保存在yylval中，那我们只要把它们俩的类型改成一样的char\*，然后在yacc匹配到一组完整的数据时将yylval中的值赋给inputBuff就可以了。