**操作系统课程设计实验报告**

——实验一：shell实验

负责人姓名：韩慧敏

学号：14061218

日期：2016.3.25

**小组成员**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 姓名 | 学号 | 实验分工 |
| 1 | 韩慧敏 | 14061218 | 实验一 |
| 2 | 张然殊 | 14061146 | 实验二 |
| 3 | 张慧昕 | 14061117 | 实验三 |
| 4 | 李雨霜 | 14061177 | 实验四 |

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 5](#_Toc446001835)

[3.设计说明 6](#_Toc446001836)

[3.1 程序流程图 6](#_Toc446001837)

[3.2基本要求实现说明 7](#_Toc446001838)

[3.3 提高要求实现说明 7](#_Toc446001839)

[4.收获和感想 10](#_Toc446001840)

# 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。

1. **需求说明**

## 2.1基本要求

## 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。

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 
2. 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

完成了以下功能：

## 程序能够正常运行。

## 能够执行fg、bg、cd、history、exit等内部命令。

对于cd、history保留原代码内容，对于fg、bg、exit进行了部分修改。

## 能够执行外部程序命令ctrl+c和ctrl+z。

对于ctrl+z进行修改，并参照ctrl+z完成了ctrl+c部分的代码。

## 4.支持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。

## 6.对YACC语法分析的文法进行了进一步的修改与完善。

## 7.在Linux下将Lex和YACC结合起来使用，进行了词法和语法分析。

## 8.对其他常用的内部命令进行了实现(包括kill, type, echo)。

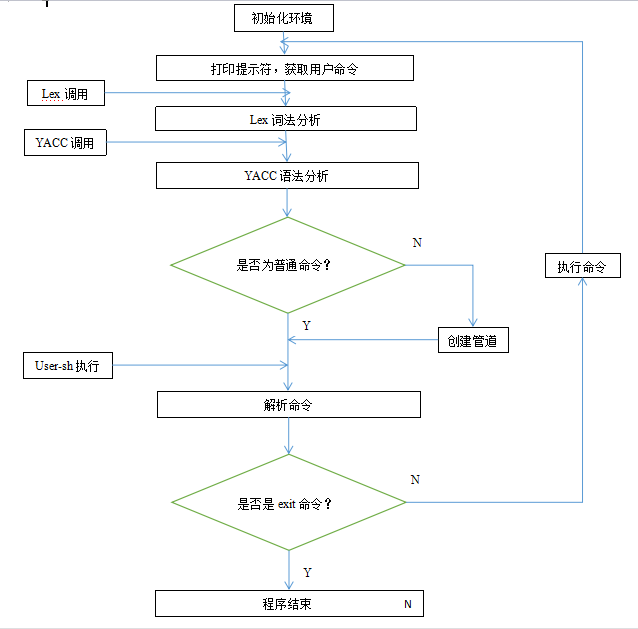
## 9. 实现对管道的支持。

## 10.考虑并实现了组合键ctrl+c命令。

参照ctrl+z完成了ctrl+c部分的代码。

# 3.设计说明

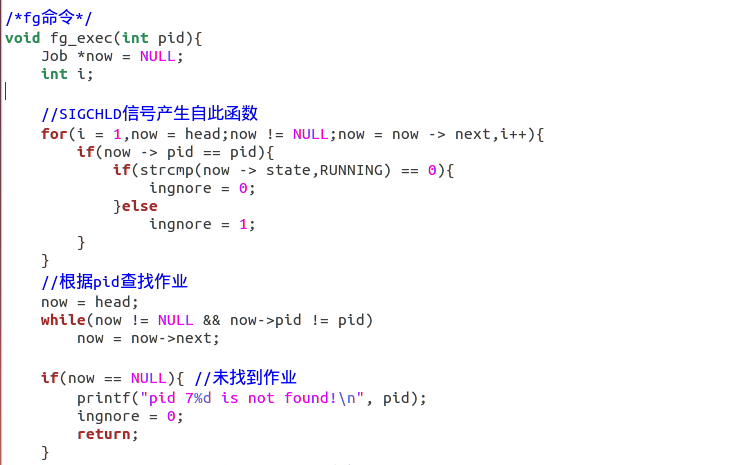
## 3.1 程序流程图

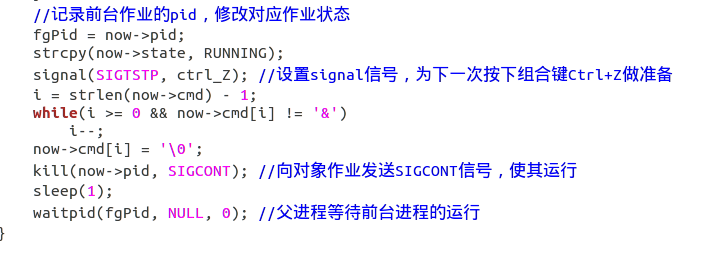


## 3.2基本要求实现说明

## 程序能够正常运行。 2. 能够执行fg、bg、cd、history、exit等内部命令。

## fg代码实现部分：





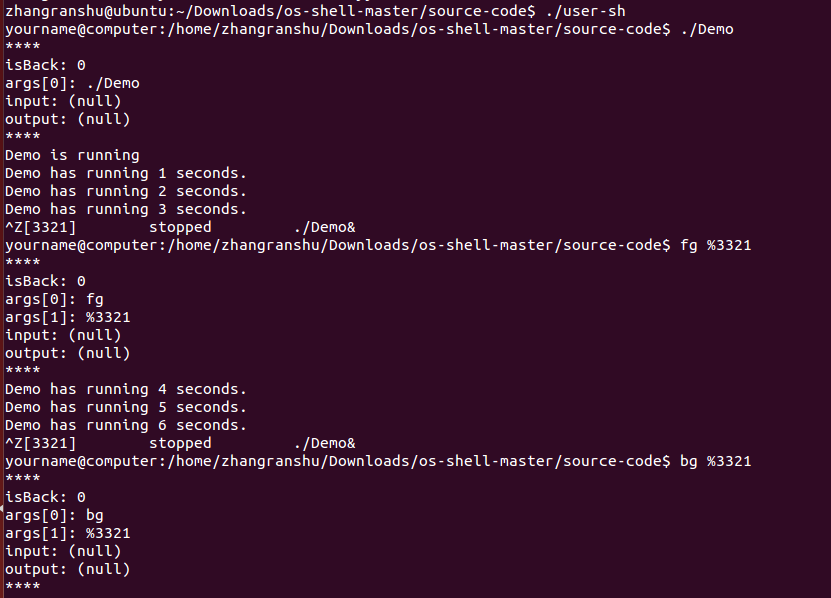
## bg代码实现部分：

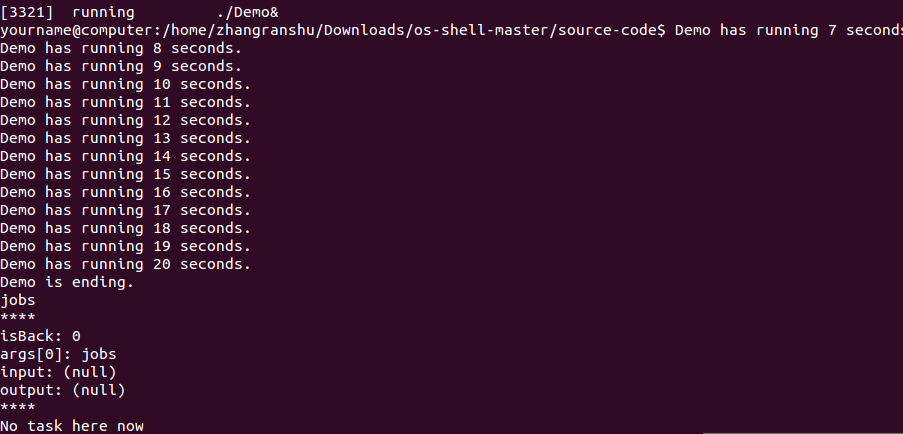
## bg

## exit代码实现部分：

## exit 3. 能够执行外部程序命令ctrl+c和ctrl+z。 4. 支持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。

（下图给出了一个简单的实现：运行Demo，挂起一个作业，把作业放到前台运行，把作业放到后台运行，全部运行结束后，后台无执行的作业，无被挂起的作业，jobs无打印的内容）





## 3.3 提高要求实现说明

## 尝试对YACC语法分析的文法进行进一步的修改与完善。

## yacc的修改与完善：在测试重定向的时候，发现并不支持后台程序重定向，因此在原bison.y文件里添加了一行代码（Line28），来支持后台重定向。 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。

## yacc与lex：首先写词法分析文件lextest.l。lex文件最重要的就是规定如何识别各种对象，因为对于作业要求的一般指令来说，字母、数字、%、./、都是合法的输入，所以用正则表达式[-a-zA-Z\_0-9./%]×来匹配，输入中的空格用[ \f\r\t\v]+来匹配。为了方便，在定义段还定义了由这两种正则表达来组合形成的重定位符号（inputRedirect/outputRedirect）和管道命令标识（pipeCmd）。对于空格字符的处理，是直接跳过的；对于字符串的处理，是存入inputBuff中；对于\n，返回0代表一行的结束。

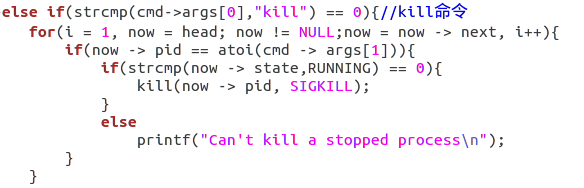
## 在lex中，还做了从字符串进行匹配的转变，把原本的yy\_input给undef掉了，重新define里一个函数来得到inputBuff，再在execute.c里对inputBuff进行处理。

## 我们查询、参考了许多做法，最后决定还是把语法的分析放在yacc文件里，而不在lex里全部匹配完，这样才能体现lex和yacc各自的特点。所以在lex中采取里strcat的拼接方法，把每次得到的字符串拼接起来，直到换行才清空。

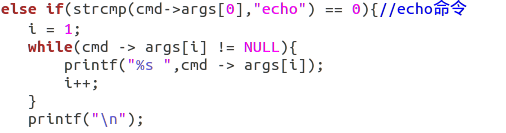
## 在语法分析文件yacc中，我们尽力分析了命令行输入可能出现的各种情况，构成了yacc中的语法规则。比较困难的是，增加管道后，管道的命令格式与从前的有些不同，造成了语法上制定规则的困难，最后我们决定放宽在yacc中对管道命令的检验，只要符合基本的格式的，就不在yyerror中报错，再在execute.c中进行进一步的判断与处理。（这里和fg、bg的处理相似，只要有fg，就不会在yyerror报错，而如fg %<这样的错误输入，会在执行文件中分析命令时报错。）

## 最后是通过改写makefile使得yacc和lex文件编译在一起，完成了整个词法语法分析器。 3. 对其他常用的内部命令进行实现。

## kill代码实现部分：



## echo代码实现部分：



## type代码实现部分：

## type 4. 实现对管道的支持。

## 由于管道命令解析时需以“|”为分割符分成不同命令，所以在execute（）中更改了一下框架，无“|”时同之前，有“|”时进入pipebegin（）函数，在该函数内定义fd二维数组，每次用fd一行创建管道，首先创建一条管道，如果成功，则解析第一条命令，然后执行命令（此处添加执行命令的参数，分别为输入输出的文件标识符，在外部命令执行判断重定向时用，第1条输入参数为0），然后进行循环，每次循环中创建管道，每条命令执行时的输入标识符为上条命令时创建的fd[0],输出标识符为本次创建的管道的fd[1],循环至最后1条之前，最后1条的输出参数为0。以此将两条命令的输入输出通过管道连接。除此还需更改外部命令执行时的重定向判断，在参数为0时即当作不需重定向，若需要重定向，则在参数不为0执行操作与源码相同，若不为0则不需打开文件，且把0或1标识符指向该参数，以此实现重定向。

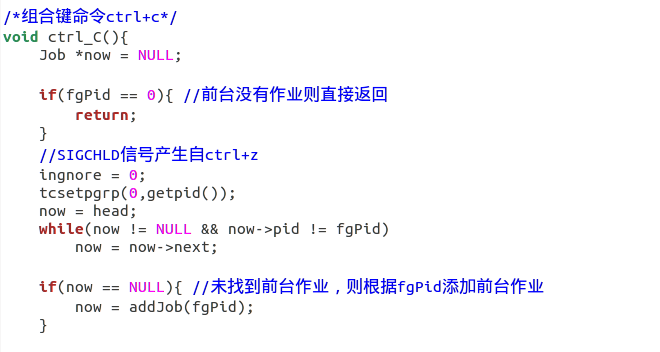
## pipe1

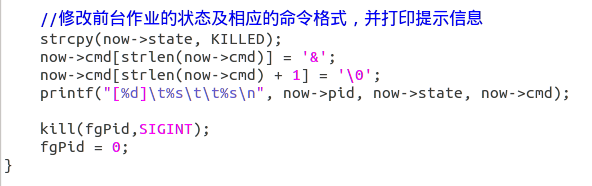
## pipe2

## 5. 考虑并实现组合键Ctrl+C命令。

## Ctrl+C最初考虑参考Ctrl+Z去实现，后来发现还是有很多不同的（何况sourceCode中CtrlZ本来有错）。首先要清楚的是，CtrlC是一种InterruptKey，会产生中断信号SIGINT和SIGCHLD，观察本来的代码，发现设置的是通过SIGCHLD信号进入函数rmjobs。所以CtrlC应当也通过这个信号自然地去进入rmjobs，来实现将“待杀死”的前台任务从jobs列表中移除。所以一方面，模仿源代码中的“Signal（SIGTSTP，Ctrl\_Z）”(SIGTSTP由CtrlZ产生),通过Signal（SIGINT，Ctrl\_C）去设定，一旦按下Ctrl\_C，有了SIGINT信号，就进入CtrlC函数去操作，改变任务的状态并输出，另一方面，通过CtrlC产生的SIGCHLD信号，进入rmjobs。

## 接下里为了区别进入rmjobs时进程不同的状态，就要分析SIGCHLD信号具体是由什么产生的。如果是CLD\_STOPPED,表明由CtrlZ产生，就由CtrlZ函数处理并返回；如果是CLD\_KILLED，就是由CtrlC产生的，由CtrlC函数处理，接着继续remove这项任务，就完成里前台任务的杀死。当然其中还包含了ingnore信号等的具体判断，但主要思想就如上述了。在具体的调试过程中，还涉及到了tcsetpgrp、waitpid等函数的问题，比较复杂，调了很久才好，实际上就是父进程、子进程并发时前后台的状态比较难以理解。





# 4.收获和感想

4.1重要问题及解决

1：kill()函数

解决方案：保证子进程在父进程执行kill函数之前运行，在execOuterCmd函数的父进程的kill函数前加上sleep(1).

2：wait()函数

解决方案：wait函数要等待子进程收到开始启动进程的信号后运行才能得到正确的结果，需要在fg\_exec函数中的kill函数后加上sleep(1).

3：Ctrl\_z杀死后台运行的作业

解决方案：在execOuterCmd函数的子进程调用execv函数之前加上setpgid(0,pgid)，为子进程设定单独的进程组。

解决过程中出现问题：调用了setpgid函数之后，子进程就不在前台进程组了，失去了对终端的控制权，无法执行需要输入的命令。

解决方案：在子进程和父进程中都为子进程调用setpgid设立单独的进程组，然后在父进程的waitpid函数之后，调用tcsetpgrp函数将终端控制权转移给子进程，因为此时的终端控制权在父进程，所以该函数写在父进程中。

解决过程中再次出现问题：父进程把终端控制权交给了子进程，当子进程运行结束时，终端控制权无主，程序出现异常。

解决方案：在子进程的末尾，也就是子进程运行结束之后，再次调用tcsetpgrp函数将终端控制权转交父进程。

4：exit -- 要在结束父进程时结束所有的子进程

解决方案：在父进程的exit(0)前加上一个for循环，访问jobs链表，向链表中的所有子进程发送一个SIGKILL信号。

解决过程中遇到问题：如果没有执行任何进程时，即now==NULL，无法进入for循环，程序无法退出

解决方案：在for循环外再写一个exit(0);

5：rmjob -- 后台程序运行过程中输入一个无关指令，如格式不正确或无法处理的指令，会导致程序运行结束以后不会从作业链表中移除，即ingnore被这些无关指令赋为1，导致不能remove。

解决方案：在fg\_exec,bg\_exec函数判断指令是否有效的时候，若无效，再次将ingnore赋为0。还有在bg命令中存在的不是挂起状态的命令时，也视为无关指令，将ingnore赋为0.

解决问题过程中遇到问题：fg命令把./Demo&放在前台运行时，程序运行结束后不会从jobs链表里删除，因为rmjobs这个函数需要子进程结束返回一个信号才能删去进程。

解决方案：在fg\_exec函数里加上如下内容:

for(i = 1,now = head;now != NULL;now = now -> next,i++){

if(now -> pid == pid){

if(strcmp(now -> state,RUNNING) == 0){

ingnore = 0;

}

else

ingnore = 1;

}

}

4.2收获与感想

通过本次实验，我们熟悉了Linux系统下的基本编程环境和开发工具的使用，明确了shell程序的实现机制以及shell与系统内核的相互关系，初步了解了Linux系统调用的使用方法。

韩慧敏：作为此次OS实验的负责人，可能我的感触要比其他的组员更深刻。最开始的时候，我以为团队完成任务必定要比个人完成来得轻松许多，大家心里并没有很强烈的紧迫感，但是安装虚拟机和双系统的频频崩溃使我们心中开始有些惶恐，又加上一堆从未使用过的系统调用，闻所未闻的语法，词法分析器的连番轰炸，都让我们预感这次的作业将是一个巨大的工程。

最怕的不是作业多，而是一头雾水，无从下手。团队的第一次会议是在作业发布后三天召开的，但是这的确是一场尴尬的会议，大家回去后仔细阅读了实验要求和源代码，但是需要自学的东西太多，不知道该从哪里找到突破口，甚至于连分配任务都做不到。再加上平时的课业压力，我们的内心几乎是崩溃的，思考再三，我们决定先找出源代码中已经实现的指令的漏洞，将找出的bug规整出来，按bug分配任务。找出实现有问题的命令之后，我们再次陷入僵局，所幸第二次实验课上，助教给我们提醒了几处易错点，我们以此为突破口，加上之前发现的bug，将任务分配给组员，总算是有了下手的契机。

万事开头难，一切开始之后，心里总算有些踏实了，但是问题也随之而来。大量的知识都要靠自学，组内四个女生，对于Linux系统都是第一次接触，完全没有经验，网上百度得来的讲解良莠不齐，想要求助于其他同学，其他同学的时间也十分紧迫，无法为我们详细的讲解，只好自己一点一点的摸索，一点一点的尝试，阅读一篇又一篇的参考资料，走过无数弯路，才慢慢的有所掌握，很多次深夜看到组员们还在群里提出疑问，分享经验，或者是又查到了一个漏洞，是一种心酸和喜悦杂糅着的心情。从最开始的一头雾水到如今的逐渐入门，从一开始的单枪匹马，各自为政，到如今热火朝天的讨论，模块化工作的有条不紊，我们牺牲了多少休息时间，又付出了多少艰苦的努力，到终于把程序大概实现的那一刻，我们不是各自单独的快乐，而是一起从电脑前抬起头来的相视一笑。以前各做各的，觉得这样的字眼矫情又尴尬，真的当大家学会分工，学会合作，学会一起讨论，一起分享，并最终克服重重困难达到目的，这种感动是不言而喻的。

## 第一次作业或许完成的不是那么的尽如人意，一方面是刚刚入手的生涩，另一方面也是大家不断磨合的过程，成果或许不是那么完美，但是相信跌跌撞撞之后，我们可以越做越好。

## 张慧昕：在这次实验中感受到了合作的重要性，对于一项任务，尤其是完全不知道怎么做的任务，且其中包含需要学习的各不相同的部分，此时进行合作，分而治之，每个人干一块，各学各的相关内容，可大量提高效率，如果在可有人讨论的情况下效率应该会更高一些，而各学各的也会导致对于另外部分的内容一知半解。

## 张然殊：对于我个人来讲，这次OS实验不仅仅是学习方面的挑战，更是心理层面上的挑战。由于不懂的东西太多，做起来简直可以用一头雾水来形容。再加上时间紧迫，每个人都在努力地完成自己的任务，并没有很多精力给我们解答问题和困惑，所以我们花费了很多时间去讨论和调试。但是这带给我的精神财富却是十分巨大的。在这个过程中，我掌握了很多shell程序的相关知识，对Linux这个新鲜的事物有了更深一层的了解，这在我的学习生涯中无疑是全新的一页。虽然前路漫漫，但我们小组勠力同心，合作无间，一定可以有更多更长足的进步。

## 李雨霜：第一次OS实验给我最大的体会就是自学能力和合作能力的重要性，从最初源代码都看不明白，到最后自己写出正确的shell程序，中间确实经历了很多，实践出真知，有时候看了很久的资料，还是一知半解，只有不停地写、不停地调试，最终才掌握了这些内容。我个人觉得OS的入门还是比较艰难的，所以一个人做实验确实会有困难，大家分工协作，各自专注一部分，很能节省时间，而且OS实验又是一个整体，在衔接时需要大量的讨论，也使得我们对不是自己任务的、其他组员做的内容有了比较详尽的了解。