**实验一——Shell程序**

小组成员：钱程，管文翰，贾焕庭，叶强

总负责：管文翰

测试人员：钱程，贾焕庭，叶强。

报告作者：钱程

润色：管文瀚

**实验目的：**

1. 学习Linux相关软件的使用。
2. 熟悉使用Linux中的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. 掌握进程控制和进程间通信的方法。

**实验要求**：

基本要求：

1. 能够执行fg、bg、cd、history、exit等内部命令。
2. 能够执行程序外部命令，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：打印作业清单，改变当前运行作业的前后台状态，控制作业挂起、终止和继续运行。

**提高要求**：

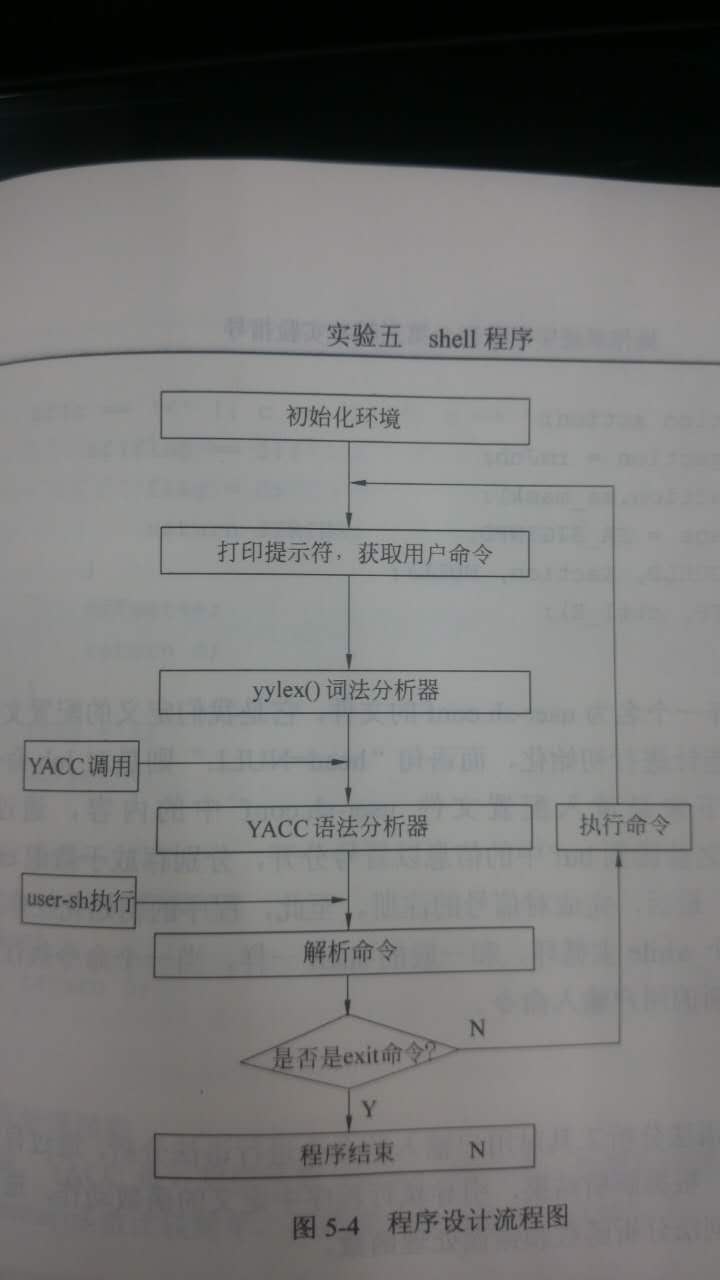
1. 对YACC语法分析的文法进行进一步的修改和完善
2. 尝试在Linux下将Lex和YACC结合起来使用进行词法和语法分析。
3. 考虑通配符的实现。
4. 实现对管道的支持。
5. 考虑实现组合键Ctrl+C命令。

**实现情况：**

基本要求可以实现，提高要求中除通配符和管道以外的功能可以实现。

**实验设计**：

1.程序设计流程图



2.基本要求实现说明

实验源码基本可以实现基本要求，基本要求部分以调试程序和修改bug为主。

3.提高要求实现说明

1）ctrl\_C实现

在init()中注册signal(SIGINT, ctrl\_C)，在ctrl\_C函数中向当前前台命令发送SIGKILL信号以结束之，无前台命令则不动作。

2）lex &YACC

使用lex进行词法分析，抛出特殊字符（<>|&\n)token，忽略空白符（实际上起到了分隔符的作用）以及除特殊字符和空白符外的其他字符组成的字符串token。

使用yyac进行语法分析，在逐层分析的同时构造出将要执行的命令，并在最后一层分析到line时调用execute()执行命令，执行完毕后结束本次分析，并重置命令。

3）管道的实现

由于时间问题，使用stupid pipe实现对管道符的支持（即新建文件作为读写中介的方法），为求实现简便要求文件夹下不能有以.pipe作为文件名开头的文件。（一般也不会有这种文件）

4.过程中发现的问题

1：setpgid(pid, pgid)函数作用对象是整个进程组，导致一种实现前后台进程组交换的设想（设定两个进程为全局进程，其中shell进程设定为前台组组长（它永远不被结束），同时在init()中fork出一个shell进程的子进程，它无限循环且忽略各种信号，让他成为一个新的进程组的组长，将其gid作为后台组gid，然后所有外部命令进程在需要时使用setpgid加入前台组或后台组）无法实现（前台组中包括shell进程，因此前台外部命令进程使用setpgid加入后台组会失败）。

解决：使用tcsetpgrp函数实现前后台进程组交换，使每个后台进程新建一个进程组，将后台进程移到前台进行的时候，将终端交给这个后台进程所属进程组（该组进包含一个进程）使用。

2：在ctrl+z时，使用setpgid使前台进程新建一个进程组并成为该组组长，失败。

解决：ctrl+z时由shell（父进程）对子进程setpgid，失败（按照网上关于setpgid的描述应该不会失败，或许在exec函数调用后进程父子关系被破坏），故只能在子进程建立之时让子进程自己对自己setpgid，成功。

3：使用tcsetpgrp函数时，后台进程转到前台执行完毕后shell被退出。此问题原因在于进程组没有进程时生命周期结束，而shell进程组此时没有与终端相连。

解决：记录shell进程的组号，每当前台进程结束或挂起之前将终端返还给shell进程组。

4：在yacc中进行语法分析的同时集成了内存处理的工作，free内存的时候与execute.c中释放命令占用空间的free操作冲突，导致多次free一个非NULL指针。

解决：将execute.c的释放命令空间的函数删除。

5：ctrl\_Z，fg\_exec等函数触发时会自动调用rmjob（由于SIGCONT也会触发rmjob，因此使用ingnore来控制rmjob在某些情况下会失效），导致非预期的作业移除。

解决：修改rmjob函数使其忽略某些与子进程结束无关的信号。

6：后台进程结束时会自动在标准输入中添加EOF，导致语法解析不正常。

解决：重定义YYINPUT函数，处理输入时将EOF忽略。（本身程序没有必要以EOF作为输入结束的标志，永远不结束输入也没有问题，解析到exit命令时自然就能结束了）

7：解析命令时偶尔进入不明死循环。原因在于yacc中巴克斯范式编写的疏漏（line标记应为非终结符，否则解析完一次命令之后所有命令均不能匹配）。

解决：仍将line作为终结符，但是每次匹配line成功就让解析函数返回一次。

8：实现stupid pipe时，最初的设想是只使用两个中介文件，进程读pipein，写pipeout，结束后将pipein删除，pipeout改名为pipein，然而执行完一个进程后发现给pipeout改名时出现了无法打开文件的错误（显示乱码，猜测是这个错误）。

解决：最初设想是进程结束时调用close，然而close的时机和close参数如何获得的问题比较烧脑，因此转念采用建立命令数 - 1数量的中介文件的方法，最后使用rm -f命令强制删除之。

9：ctrl+c（其实是空行）导致输入解析异常。

解决：最初设想ctrl+c时退出yyparse或重启shell，然而浪费大量时间仍没能找到解决方法，

最后发现问题是空行导致的orz。不过ctrl+c时与bash一致的处理方法就应该是退出yyparse或重启shell，不知怎么实现（尽管无伤大雅）。

坑1：

解决：后台进程执行的时候，使用两个信号量SIGUSR1和SIGUSR2实现父子进程间的多次同步

坑2：

解决：在父进程发出信号后sleep一秒。（这个sleep不太影响效率，因为即使父进程sleep，子进程也能运行）

坑3：

解决：利用进程组机制，非前台的进程组无法接收到前台发出的信号，因此前台使用组合键不会影响非前台进程。

收获及体会

开发的时候一定要统一运行环境，我们组就确实遇到了总负责人代码汇总之后，并测试运行之后，再回到我们各自的电脑上不能运行的情况，可能还是和不同系统的编译环境相关。就是盯着代码没法改，生生看着terminal报错。

大量的时间浪费在了调试上，完成此实验时难以使用gbd对程序进行调试（gdb时断点设在main中意义不大，而yyparse函数由于是由yacc生成，也难以进行追踪，且进入yyparse后单步命令next会失效。也许和我操作不当也有关系吧），而图形化调试环境由于种种系统问题和依赖问题没能很顺利地搭建起来（最后由于时间紧张而在折腾一天后放弃），最后采用的是最原始的printf调试法，一开始每次调试都要输入很多行printf和重复率很高的字符串，经过无数次的make clean 和 make后，逐渐熟悉了printf大法后，编写了调试函数a1,2,3...，配合while(1)手工设置断点，调试效率才有了比较可观的提高。如果有时间的话，还是希望能搭建图形化调试界面，可以给调试很多方便。