实验三：实现Linux命令解释器实验报告一、实验任务要求

（1）分析且运行myshell。  
 （2）扩充myshell功能，使其支持以下内部命令：

① cd<目录>———更改当前二点工作目录到另一个<目录>。如果<目录>未指定，输出当前工作目录。如果<目录>不存在，应当由错误信息提示。

② echo<内容>——显示echo后的内容且换行。

③ help————简短概要地输出myshell的使用方法和基本功能。

④jobs——输出myshell当前的一系列子进程，必须提供子进程的命名和pid号

（3）添加重定向和管道功能

### 二、实验原理和方法设计

本实验原理是通过学习明白shell的工作原理，达到实现一个自己的shell，满足指定要求的效果。所采取的主要方法是自主学习shell工作原理，利用C语言补充实现myshell。

### 三、实验工具和环境配置说明

VMware15.1

Fedro7.0

### 四、实验步骤

第一个实验十分简单，只需按照书本上一步步进行即可。这里就只大概叙述一下过程，并将代码贴出：

首先，该程序需要通过启动一个主程序myshell.c，该程序通过无限循环的读取指令，并将指令进行分发至yylex()函数完成。代码如下：

#include<stdio.h>

#include<string.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<dirent.h>

#include<fcntl.h>

#include<time.h>

#include<ftw.h>

#include<unistd.h>

#define PROMPT\_STRING "[myshell]$"

#define QUIT\_STRING "exit\n"

static char inbuf[MAX\_CANON];

char\* g\_ptr;

char\* g\_lim;

extern void yylex();

int main()

{

while(1)

{

if(fputs(PROMPT\_STRING,stdout)==EOF)

continue;

if(fgets(inbuf,MAX\_CANON,stdin)==NULL)

continue;

if(strcmp(inbuf,QUIT\_STRING)==0)

break;

g\_ptr=inbuf;

g\_lim=inbuf+strlen(inbuf);

pid\_t pid;

pid=getpid();

printf("%d",pid);

yylex();

}

return 0;

}

以上为该程序的主体框架，真正去执行指令的为yylex()函数。

yylex()是由专业的语法分析器flex根据输入文件parse.lex生成。

parse.lex文件主要由三部分构成，包括所使用的头文件、数据变量和函数原型。每个部分由%%进行分割。所有的函数实现就写在第三部分中，该实验下，我将所有功能写于exec\_simple\_cmd()函数下。下面对实现功能分开叙述，最后再附上全部代码。

1. cd

该功能是非常好实现的，因为C中已封装好有chdir()函数，直接调用即可。具体代码为：

if(strcmp(g\_argv[0],"cd")==0)

{

if(chdir(g\_argv[1])==-1)

{

printf("Dir or file is not exit!\n");

}

else

{

printf("Path has changed!\n");

}

}

1. echo

该功能也是非常简单，检测第一个参数为echo，直接将后续内容输入即可。

if(strcmp(g\_argv[0],"echo")==0)

{

for(int i=1;i<=argc-1;i++)

{

printf("%s",g\_argv[i]);

}

printf("\n");

}

1. help

只需将各个功能输出即可。

if(strcmp(g\_argv[0],"help")==0)

{

printf("Welcome to use myshell!\n");

printf("How to use:\n");

printf("1.cd <path>\n");

printf("2.ls\n");

printf("3.jobs\n");

printf("4.echo <file>\n");

printf("5.help\n");

printf("6.exit\n");

}

1. jobs

这部分我没找到合适的方法，所以在实现时，我直接使用的shell指令，因为myshell中所有创建的子进程均名为myshell，所以我直接使用了如下代码：

if(strcmp(g\_argv[0],"jobs")==0)

{

system("ps | grep myshell");

}

首先是进行命令分解。

int splitcommand(char cmd[])

{

int num=0;

int I,j;

int len = strlen(cmd);

for( i=0,j=0;i<len;i++)

{

if(cmd[i]!=’ ’)

{

cmds[num][j++]=cmd[i];  
}

else

{

if(j!=0)

{

Cmds[num][j]=’\0’;

++num;

J=0;  
}

}

}

if(j!=0)

{

cmds[num][j]=’\0’;

++num;

}

}

然后就是之前的匹配指令，运行相关结果。

后面主要变更的是将输出结果使用sprintf+strcat的方式存储起来。就两行代码。这里就不列出来了。

最后是实现重定向和管道功能。这两个功能的话就我之前的代码时需要大改的。因为这两个功能都需要对函数只需结果进行进一步的操作。那么就需要将前期得到的结果进行存储，放置在一个字符串里。再根据命令解析后的内容决定是输出还是存储亦或是作为一个输入进行下一步操作。

首先重定向首先查找重定向符号’>’，找到后按照其进行划分，fopen打开重定向文件。

int in=0,out=0;

char \*infile=NULL,\*outfile=NULL;

FILE \*p;

int end=right;

for(int i=left;i<right;i++)

{

if(strcmp(cmd[i],'>')==0)

{

++in;

if(i+1<right)

infile=cmds[i+1];

else

{

printf("Format error\n");

return -1;

}

}

}

p=fopen(infile,”w”);

然后执行内容，将格式化后的结果字符串写入即可。

else if(pid==0)

{

if(in==1)

{

freopen(infile,"r",stdin);

}

char com[1024];

for(int i=left;i<endindex;i++)

{

cmd[i]=cmds[i];

}

cmd[endindex]=NULL;

out(cmd[left]);

}

最后是实现管道功能。首先判断是否含有管道命令’|’，含有的话执行管道命令。方法与重定向类似，这里我是基于重定向，使用的临时文件作为传递的媒介，再第二个指令使用后将该文件删除，以做到用户不可见。

代码和重定向的基本一致，就是增加了对已写文件的操作，作为第二参数进行使用。因为在实现的功能中只有三个功能可能用到改功能，而这三个功能的参数形式都是一样的，所以只要进行sprintf格式化构造即可。即：

sprintf(cmd1,”%s %s”,cmd,tmp\_file);

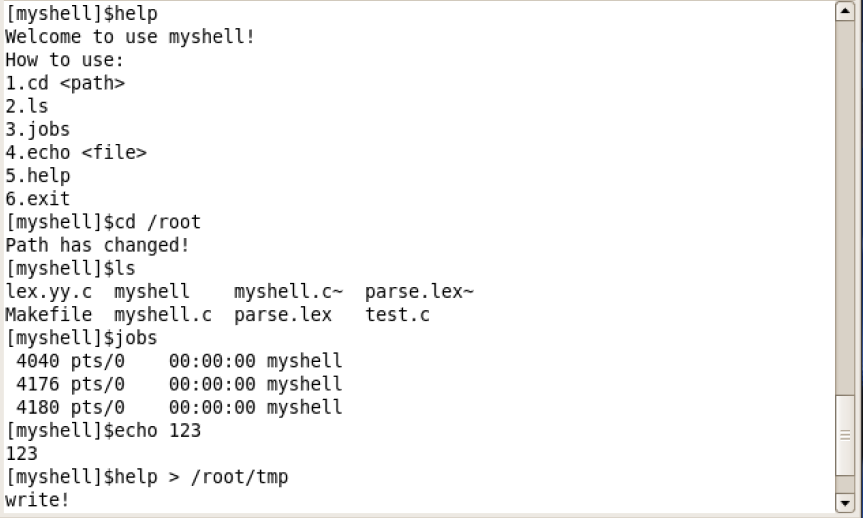
这里我实际上并不是十分理解到底是要以什么样的形式进行输出。所以我定义的管功能举例为：

echo /root | cd

就是先输出/root后执行cd，切换至/root文件夹。

这样就简单实现了该功能。

### 五、实验结果呈现与说明（数据或程序执行结果载图等）



图表 1：部分结果



图表 2：部分结果

### 六、实验结论与分析

通过该实验，我大概理解了shell的执行模式，基本上就是由一个进程分发程序，加上指令处理程序组成。shell是多个进程的祖先，不断着安排着子孙进行工作。这样看来，shell本事并不是一个很难得机制，当然，系统自带的shell是否也是这个机制我还不是很清楚，但估计八九不离十。

这个实验对我而言就像解谜了一般，使原本很神奇的命令行结构、工作机制变得清晰了许多。