/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project Name: myshell

\*Description: a reduced shell program implemented by C

\*Auther：lishichengyan

\*Student ID：omitted

\*Last Modified： 2017.08.03

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*必要的头文件包含\*/

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<memory.h>

#include<string.h>

#include<sys/types.h>

#include<sys/wait.h>

#include<sys/stat.h>

#include<sys/param.h>

#include<pwd.h>

#include<errno.h>

#include<time.h>

#include<fcntl.h>

#include<dirent.h>

#include<signal.h>

/\*和长度有关的宏定义\*/

#define MAX\_LINE 80//最大命令长度

#define MAX\_NAME\_LEN 100//最大用户名长度

#define MAX\_PATH\_LEN 1000//最大路径长度

/\*全局变量申明\*/

extern char \*\*environ;//必须用extern申明，否则会报错

char \*cmd\_array[MAX\_LINE/2+1];//保存命令输入，就是原框架的char\* args[]

int pipe\_fd[2];//和管道有关的数组，作为pipe()的参数

int cmd\_cnt;//命令中字符串的个数

/\*老师框架里的函数（有改动）\*/

void readcommand();//读取用户输入

int is\_internal\_cmd();//处理内部命令

int is\_pipe();//分析管道命令

void do\_redirection();//分析重定向，对内部命令无效

/\*自己定义的函数\*/

void welcome();//打印欢迎信息，带有颜色

void printprompt();//打印提示符，必须包含当前路径名

int getcommandlen();//计算命令长度

void do\_pipe(int pos);//执行管道命令

void run\_external\_cmd(int pos);//执行外部命令

int is\_bg\_cmd();//判断是否有后台运行符号&

void myquit();//quit，退出myshell

void myexit();//exit，直接退出

void myclr();//clr，和BatchShell下的clear一样

void print\_continue\_info();//打印"continue"相关信息

void mypwd();//pwd，打印当前工作目录

void myecho();//echo，必须支持重定向

void myecho\_redirect();//带重定向的echo

void mytime();//time，和"date"类似

void myenviron();//environ，和env一样,必须支持重定向

void myenviron\_redirect();//带重定向的environ

void mycd();//cd，切换到某个目录

void myhelp();//help，必须支持重定向

void myhelp\_redirect();//带有重定向的help

void print\_manual();//打印用户手册，是myhelp()的子函数

void print\_cmdinfo(char\* cmdname);//打印每个命令的帮助信息，是myhelp()的子函数

void myexec();//exec，开启一个新进程并替换当前进程

void mytest();//test，检查文件类型，支持-l,-b,-c,-d四个选项

void myumask();//umask，查看默认的umask值或者重置umask

void myjobs();//jobs，查看正在运行的=进程

void myfg(pid\_t pid);//fg，切换进程到前台

void mybg(pid\_t pid);//bg，切换进程到后台

void mybatch();//实现命令批处理，一次性执行保存在文件里的命令

void mydir();//dir，显示当前目录下的所有文件

void mydir\_redirect();//带有重定向的dir

/\*老师所给函数的实现（框架有调整）\*/

/\*实现顺序和定义顺序相同\*/

void readcommand(){

//这个函数用来读取用户输入

int cnt=0;//记录cmd\_array[]中字符串的个数

char str[MAX\_LINE];

char\* helper;

memset(cmd\_array,0,MAX\_LINE/2+1);//每次必须清空！

fgets(str,MAX\_LINE,stdin);//用fgets代替gets，因为gets不检查溢出，比较危险

if(str[strlen(str)-1]=='\n'){

str[strlen(str)-1]='\0';//fgets会补'\n'，这里必须把'\n'替换成'\0'

}

helper=strtok(str," ");//用空格分割这个命令

while(helper!=NULL){//将分割后得到的结果写进cmd\_array

cmd\_array[cnt]=(char\*)malloc(sizeof(\*helper));

strcpy(cmd\_array[cnt++],helper);//注意：即便直接回车cmd\_cnt也是1

helper=strtok(NULL," ");

}

cmd\_cnt=cnt;//cmd\_cnt的值就是cnt的值

}

int is\_internal\_cmd(){

//这个函数用来解析内部命令

//根据不同的结果来调用不同函数来达到目的

if(cmd\_array[0]==NULL){//如果没有命令（只是回车）

return 0;//返回0使得主函数的continue不执行

}

else if(strcmp(cmd\_array[0],"quit")==0){//quit，退出之前有“Thank you...”提示信息

myquit();

}

else if(strcmp(cmd\_array[0],"exit")==0){//exit，直接退出

myexit();

}

else if(strcmp(cmd\_array[0],"clr")==0){//clr，清屏

myclr();

return 1;

}

else if(strcmp(cmd\_array[0],"continue")==0){//continue命令，它只在循环里有效

print\_continue\_info();

return 1;

}

else if(strcmp(cmd\_array[0],"pwd")==0){//pwd，打印当前工作目录

mypwd();

return 1;

}

else if(strcmp(cmd\_array[0],"echo")==0){//echo

int i;

for(i=1;i<cmd\_cnt;i++){//从第二个字符串开始分析

if(strcmp(cmd\_array[i],">")==0||strcmp(cmd\_array[i],">>")==0){//如果有重定向符号

myecho\_redirect();//调用带有重定向的echo

return 1;

}

}

myecho();//如果没有重定向，直接echo

return 1;

}

else if(strcmp(cmd\_array[0],"time")==0){//time，显示当前时间，和date命令相似

mytime();

return 1;

}

else if(strcmp(cmd\_array[0],"environ")==0){//environ

if(cmd\_array[1]!=NULL&&(strcmp(cmd\_array[1],">")==0||strcmp(cmd\_array[1],">>")==0)){//带有重定向

myenviron\_redirect();//调用带有重定向的environ

return 1;

}

else{

myenviron();//没有重定向

return 1;

}

}

else if(strcmp(cmd\_array[0],"cd")==0){//cd，切换目录

mycd();

return 1;

}

else if(strcmp(cmd\_array[0],"help")==0){//help

if(cmd\_array[1]!=NULL&&(strcmp(cmd\_array[1],">")==0||strcmp(cmd\_array[1],">>")==0)){//带有重定向

myhelp\_redirect();//注意：格式是 "help > filename"或者"help >>filename"

return 1;

}

else{

myhelp();//没有重定向

return 1;

}

}

else if(strcmp(cmd\_array[0],"exec")==0){//exec，开启一个新进程替换当前进程

myexec();

return 1;

}

else if(strcmp(cmd\_array[0],"test")==0){//test，用来查看文件属性，支持[-l]，[-d]，[-c]，[-b]四个选项

mytest();

return 1;

}

else if(strcmp(cmd\_array[0],"umask")==0){//umask，查看或者设置umask值

myumask();

return 1;

}

else if(strcmp(cmd\_array[0],"jobs")==0){//jobs，查看运行的进程

myjobs();

return 1;

}

else if(strcmp(cmd\_array[0],"fg")==0){//fg，将进程切换到前台

pid\_t pid;

if(cmd\_array[1]!=NULL){

pid=atoi(cmd\_array[1]);//用atoi转换，获取pid

}

else{//如果只有一个fg

printf("myshell: fg: no job assigned\n");//打印提示信息

return 1;

}

myfg(pid);

return 1;

}

else if(strcmp(cmd\_array[0],"bg")==0){

pid\_t pid;

if(cmd\_array[1]!=NULL){

pid=atoi(cmd\_array[1]);//用atoi转换，获取pid

}

else{//只有一个bg

printf("myshell: bg: no job assigned\n");//打印提示信息

return 1;

}

mybg(pid);

return 1;

}

else if(strcmp(cmd\_array[0],"myshell")==0){

if(cmd\_cnt==1){//只有一个myshell命令

printf("myshell: myshell: too few arguments\n");//打印提示信息

return 1;

}

else if(cmd\_cnt==2){//输入格式是：myshell [filename]

mybatch();

return 1;

}

else{//参数过多的情况

printf("myshell: myshell: too many arguments\n");

return 1;

}

}

else if(strcmp(cmd\_array[0],"dir")==0){//dir

if(cmd\_array[1]!=NULL&&(strcmp(cmd\_array[1],">")==0||strcmp(cmd\_array[1],">>")==0)){//有重定向，格式是: dir > filename或者dir >> filename

mydir\_redirect();//调用带有重定向的dir

return 1;

}

else{//没有重定向

mydir();

return 1;

}

}

else if(strcmp(cmd\_array[0],"set")==0){//I'll try latter

printf("myshell: set: not supported currently\n");

return 1;

}

else if(strcmp(cmd\_array[0],"unset")==0){//I'll try latter

printf("myshell: unset: not supported currently\n");

return 1;

}

else if(strcmp(cmd\_array[0],"shift")==0){//I'll try latter

printf("myshell: shift: not supported currently\n");

return 1;

}

else{

return 0;//返回0使得主函数的continue不执行

}

}

int is\_pipe(){

int i;

for( i=1;i<cmd\_cnt;i++){//从第二个字符串开始分析

if(cmd\_array[i]!=NULL&&strcmp(cmd\_array[i],"|")==0){

cmd\_array[i]=NULL;//把管道符替换成NULL，因为已经不再需要，避免对命令执行造成影响

return i+1;//返回下一个命令的位置

}

}

return 0;//没有pipe，返回0

}

void do\_redirection(){

//这个函数仅用来实现外部命令的重定向

//对于：dir, environ, echo, help命令

//有专门的函数体执行它们的重定向

int i;

for( i=1;i<cmd\_cnt;i++){

if(cmd\_array[i]!=NULL){

if(strcmp(cmd\_array[i],">")==0){//>:重写文件

int output=open(cmd\_array[i+1],O\_WRONLY|O\_TRUNC|O\_CREAT,0666);//必须用O\_TRUNC

dup2(output,1);//把stdout重定向到output

close(output);

cmd\_array[i]=NULL;//把>替换成NULL

i++;

continue;//跳过

}

if(strcmp(cmd\_array[i],">>")==0){//>>:在文件内容后追加

int output=open(cmd\_array[i+1],O\_WRONLY|O\_APPEND|O\_CREAT,0666);//必须用O\_APPEND

dup2(output,1);//把stdout重定向到output

close(output);

cmd\_array[i]=NULL;//用NULL代替>>

i++;

continue;//跳过

}

if(strcmp(cmd\_array[i],"<")==0){//<:输入重定向

int input=open(cmd\_array[i+1],O\_CREAT|O\_RDONLY,0666);

dup2(input,0);//把stdin重定向到input

close(input);

cmd\_array[i]=NULL;//用NULL替换<

i++;

}

}

}

}

/\*自己定的函数的实现\*/

/\*实现顺序和定义顺序相同\*/

void welcome(){

//如下是欢迎信息

//为了是程序更友好，加入了颜色

//颜色是紫色，背景色与shell相同

printf("\e[35mwelcome to myshell\e[0m\n");

printf("\e[35mit's a unix-like shell program made by WuYusong\e[0m\n");

printf("\e[35mhope you have a good time with it :-)\e[0m\n");

}

void printprompt(){

//这个函数用来打印命令提示符

//为了使程序更友好，加入了颜色

//颜色是蓝色

char hostname[MAX\_NAME\_LEN];

char pathname[MAX\_PATH\_LEN];

struct passwd \*pwd;

pwd=getpwuid(getuid());//通过pid获取用户信息

gethostname(hostname,MAX\_NAME\_LEN);//取得hostname

getcwd(pathname,MAX\_PATH\_LEN);//获取绝度路径把它储存到第一个参数pathname[]

printf("\e[34mmyshell>%s@%s:\e[0m",pwd->pw\_name,hostname);//打印提示符，颜色是蓝色

if (strncmp(pathname,pwd->pw\_dir,strlen(pwd->pw\_dir))==0){//比较两条路径

printf("~%s",pathname+strlen(pwd->pw\_dir));//打印路径

}

else {

printf("%s",pathname);//打印当前工作路径

}

if (geteuid()==0) {//函数返回有效用户的id

printf("#");//如果是root用户，打印#提示符

}

else {

printf("$");//普通用户打印$提示符

}

}

int getcommandlen(){

int tot\_len=0;

int i;

for(i=0;i<cmd\_cnt;i++){

tot\_len+=strlen(cmd\_array[i]);//注意:空格没有算进去

}

return tot\_len+cmd\_cnt-1;//因此这里要把空格的长度加进去，直接回车返回-1

}

void do\_pipe(int pos){

int pid;

if(pos==0){//没有pipe

return;

}

if((pid=fork())==0){//子进程

close(pipe\_fd[1]);//关闭写

dup2(pipe\_fd[0],0);//重定向stdin到pipe\_fd[0]

run\_external\_cmd(pos);//执行外部命令

}

else{//父进程

close(pipe\_fd[1]);//关闭写

waitpid(pid,NULL,0);//阻塞父进程等待子进程

}

}

void run\_external\_cmd(int pos){

int res;

res=execvp(cmd\_array[pos],cmd\_array+pos);//用execvp执行命令

if(res<0){//如果执行失败

printf("myshell: command not found\n");//打印提示信息

}

}

int is\_bg\_cmd(){

int i,lastpos;

if(cmd\_cnt==0){//直接回车的情况

return 0;

}

for(i=0;i<cmd\_cnt&&cmd\_array[i]!=NULL;i++){}//什么都不做，只是为了得到i

lastpos=i-1;//最后一个位置

if(strcmp(cmd\_array[lastpos],"&")==0){//命令最末有&

cmd\_array[lastpos]=NULL;//用NULL替换&

cmd\_cnt--;//必须减掉1，因为&已经被替换了

return 1;//返回1表示这个命令需要在后台执行

}

else{

return 0;//否则返回0

}

}

void myquit(){

printf("Thanks for your using，bye-bye！\n");

sleep(1);//暂停1s，看上去视觉效果好一些

exit(0);

}

void myexit(){

exit(0);//直接退出

}

void myclr(){

printf("\033[2J");//清屏

printf("\033[H");//把光标移动到合适的位置

}

void print\_continue\_info(){

//对于只在脚本循环中有效的continue

//在终端下给出提示信息

printf("myshell: continue: only meaningful in a 'for','while' or 'until' loop\n");

}

void mypwd(){

char pathname[MAX\_PATH\_LEN];

if(getcwd(pathname,MAX\_PATH\_LEN)){//获取路径名

printf("%s\n",pathname);

}

else{//如果出错

perror("myshell: getcwd");//报错

exit(1);

}

}

void myecho(){

int i;

for(i=1;i<cmd\_cnt;i++){

printf("%s",\*(cmd\_array+i));

if(i==cmd\_cnt-1){//打印最后的字符时不要空格，直接break

break;

}

printf(" ");

}

printf("\n");//然后换行

}

void myecho\_redirect(){

//echo的内容可以被重定向到文件

//这个函数虽然长但是并不复杂

//核心是使用dup2()来进行重定向

//重定向以后的标准输出（stdout）就会被输出到指定的文件

int fd;//文件描述符

pid\_t pid;

char filename[MAX\_NAME\_LEN];//用来保存文件名

for(int i=1;i<cmd\_cnt;i++){

if(strcmp(cmd\_array[i],">")==0||strcmp(cmd\_array[i],">>")==0){

if(cmd\_array[i+1]==NULL){//如果在>和>>之后没有路径名

printf("myshell: syntax error\n");//如果出现错误

}

else{

strcpy(filename,cmd\_array[i+1]);//获取文件名

}

if(strcmp(cmd\_array[i],">")==0){

fd=open(filename,O\_CREAT|O\_TRUNC|O\_WRONLY,0600);//因为要重写，所以必须用O\_TRUNC

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//把stdout重定向到fd

for(int j=1;j<i;j++){//开始把内容写进文件

printf("%s ",cmd\_array[j]);//打印的内容其实去到了文件（因为已经重定向了）

}

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{//如果出现错误

perror("myshell: fork");

exit(1);

}

close(fd);

}

else{//如果是>>，实现追加的重定向

fd=open(filename,O\_CREAT|O\_APPEND|O\_WRONLY,0600);//这样一来就必须用O\_APPEND

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//重定向stdout到fd

for(int j=1;j<i;j++){//向文件写内容

printf("%s ",cmd\_array[j]);

}

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{

perror("myshell: fork");

exit(1);

}

close(fd);//最后不要忘了关闭文件

}

}

}

}

void mytime(){

int weekday;

int month;

time\_t tvar;

struct tm \*tp;

time(&tvar);

tp=localtime(&tvar);//获取本地时间

weekday=tp->tm\_wday;

switch(weekday){//根据不同的值打印不同的星期

case 1:

printf("Mon ");

break;

case 2:

printf("Tues ");

break;

case 3:

printf("Wed ");

break;

case 4:

printf("Thur ");

break;

case 5:

printf("Fri ");

break;

case 6:

printf("Sat ");

break;

case 7:

printf("Sun ");

break;

default:

break;

}

month=1+tp->tm\_mon;//必须要加1，经过查阅资料：tm\_mon比实际的值少了1

switch(month){//根据不同的值打印月份名

case 1:

printf("Jan ");

break;

case 2:

printf("Feb ");

break;

case 3:

printf("Mar ");

break;

case 4:

printf("Apr ");

break;

case 5:

printf("May ");

break;

case 6:

printf("Jun ");

break;

case 7:

printf("Jul ");

break;

case 8:

printf("Aug ");

break;

case 9:

printf("Sep ");

break;

case 10:

printf("Oct ");

break;

case 11:

printf("Nov ");

break;

case 12:

printf("Dec ");

break;

default:

break;

}

printf("%d ",tp->tm\_mday);//日期

printf("%d:",tp->tm\_hour);//小时

printf("%d:",tp->tm\_min);//分钟

printf("%d ",tp->tm\_sec);//秒

printf("CST ");//CST，意思是China Standard Time

printf("%d\n",1900+tp->tm\_year);//必须加上1900，返回的值并不是完整的年份，比真实值少了1900

}

void myenviron(){

//用environ[]来实现全局变量的打印

//"environ" 必须实现被申明：exetern char\*\* environ

for(int i=0;environ[i]!=NULL;i++){

printf("%s\n",environ[i]);

}

}

void myenviron\_redirect(){

//把环境变量重定向到文件

//思路是用dup2()

//这和上面的重定向相似，但是一些关键的代码是不一样的

int fd;//文件描述符

pid\_t pid;

char filename[MAX\_NAME\_LEN];

if(cmd\_array[2]==NULL){//如果出现错误

printf("error occurs when redirecting\n");

exit(1);

}

else{

strcpy(filename,cmd\_array[2]);//获取文件名

}

if(strcmp(cmd\_array[1],">")==0){//>：重写文件内容

fd=open(filename,O\_CREAT|O\_TRUNC|O\_WRONLY,0600);//必须用O\_TRUNC

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//把stdout重定向到fd

for(int i=0;environ[i]!=NULL;i++){//向文件写内容

printf("%s\n",environ[i]);

}

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{

perror("myshell: fork");

exit(1);

}

close(fd);

}

else{//如果是>>

fd=open(filename,O\_CREAT|O\_APPEND|O\_WRONLY,0600);//必须用O\_APPEND

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//把stdout重定向到fd

for(int i=0;environ[i]!=NULL;i++){

printf("%s\n",environ[i]);

}

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{

perror("myshell: fork");

exit(1);

}

close(fd);

}

}

void mycd(){

struct passwd \*pwd;//用来获取参数pw\_dir

char pathname[MAX\_PATH\_LEN];//储存路径名

pwd=getpwuid(getuid());

if(cmd\_cnt==1){//如果只有一个cd

strcpy(pathname,pwd->pw\_dir);//获取pathname

if(chdir(pathname)==-1){//如果有错

perror("myshell: chdir");//报错

exit(1);

}

}

else{//如果有路径

if(chdir(cmd\_array[1])==-1){//如果chdir执行失败

printf("myshell: cd: %s :No such file or directory\n",cmd\_array[1]);//打印提示信息

}

}

}

void myhelp(){

if(cmd\_cnt==1){//如果是不带参数的help

print\_manual();//调用子函数print\_manual打印用户帮助手册

}

else if(cmd\_cnt==2){//如果格式是"help [command]"

print\_cmdinfo(cmd\_array[1]);//打印单个命令的帮助信息

}

else{//如果有错

printf("myshell: help: Invalid use of help command\n");//打印提示信息

}

}

void myhelp\_redirect(){

//重定向帮助信息到文件

//这个函数并不支持"help [command] > filename"这样的格式

//因为其实和help直接重定向比起来是大同小异

int fd;//文件描述符

pid\_t pid;

char filename[MAX\_NAME\_LEN];

if(cmd\_array[2]==NULL){

printf("error occurs when redirecting\n");

exit(1);

}

else{

strcpy(filename,cmd\_array[2]);//获取文件名

}

if(strcmp(cmd\_array[1],">")==0){//>：重写文件内容

fd=open(filename,O\_CREAT|O\_TRUNC|O\_WRONLY,0600);//必须用O\_TRUNC

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//把stdout重定向到fd

print\_manual();//打印内容重定向到了文件

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//阻塞父进程，等待子进程

}

else{

perror("myshell: fork");

exit(1);

}

close(fd);

}

else{//如果是>>重定向

fd=open(filename,O\_CREAT|O\_APPEND|O\_WRONLY,0600);//必须用O\_APPEND

if(fd<0){

perror("myshell: open");

exit(1);

}

if((pid=fork())==0){//子进程

dup2(fd,1);//重定向stdout到fd

print\_manual();//调用print\_manual,信息打印到文件

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{

perror("myshell: fork");

exit(1);

}

close(fd);//不要忘记关闭文件

}

}

void print\_manual(){

//这个函数很“无聊”但是很重要

//它为用户打印myshell命令的帮助信息

//直接使用help就可以进行查看

printf("welcome to the manual of myshell, hope it's useful for you\n");

printf("the following are the BUILT-IN commands supported by myshell\n");

printf("\n");

printf("NAMES FORMATS DESCRIPTIONS\n");

printf("bg: bg [job\_spec] execute commands in background\n");

printf("cd: cd [dir] go to a specified directory\n");

printf("continue: continue [n] valid only in for, while, or until loop\n");

printf("echo: echo [arg ...] print strings after echo,redirection is supported\n");

printf("exec: exec [command] execute a command and replace the current process\n");

printf("exit: exit quit the shell directly\n");

printf("fg: fg [job\_spec] execute commands in foreground\n");

printf("jobs: jobs check the processes running in the system\n");

printf("pwd: pwd print the current working directory\n");

printf("set: set [-$] [command or arg ...] set shell variables\n");

printf("shift: shift [n] shift user's inputs\n");

printf("test: test [arg ...] check file attributes, 4 options are supported so far\n");

printf("time: time show the current time in an elegant format\n");

printf("umask: umask [-p] [-S] [mode] change the value of umask\n");

printf("unset: unset [name] unset shell variables\n");

printf("clr: clr clear the screen\n");

printf("dir: dir [dir] list the file names in the target directory\n");

printf("environ: environ list all the environment variables\n");

printf("help: help/help [command] show the manual of help/get help info of a sepcified command\n");

printf("quit: quit quit the shell with thank-you information\n");

printf("myshell: myshell [filename] execute a batchfile\n");

printf("for more information, use help [command] to see diffirent options of each command\n");

fflush(stdout);

}

void print\_cmdinfo(char\* cmdname){

//这个函数显示myshell命令的option

//需要说明的是[command] --help这样的格式是无效的

//因为这会带来分析命令是不必要的麻烦

//正确的格式是help [command]

if(strcmp(cmdname,"bg")==0){

printf("usage:execute commands in background\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"cd")==0){

printf("usage:go to a specified directory\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"continue")==0){

printf("usage:valid only in for, while, or until loop\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"echo")==0){

printf("usage:print strings after echo,redirection is supported\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"exec")==0){

printf("usage:execute a command and replace the current process\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"exit")==0){

printf("usage:quit the shell directly\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"fg")==0){

printf("usage:execute commands in foreground\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"jobs")==0){

printf("usage:check the processes running in the system\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"pwd")==0){

printf("usage:print the current working directory\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"set")==0){

printf("usage:set shell variables\n");

printf("options descriptions\n");

printf("/ not supported currently\n");

}

else if(strcmp(cmdname,"shift")==0){

printf("usage:shift user's inputs\n");

printf("options descriptions\n");

printf("/ not supported currently\n");

}

else if(strcmp(cmdname,"test")==0){

printf("usage:check file attributes, 4 options are supported so far\n");

printf("options descriptions\n");

printf("[-l] test if the file is a symbolic link\n");

printf("[-b] test if the file is a block device\n");

printf("[-c] test if the file is a character device\n");

printf("[-d] test if the file is a directory\n");

}

else if(strcmp(cmdname,"time")==0){

printf("usage:show the current time in an elegant format\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"umask")==0){

printf("usage:change the value of umask\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"unset")==0){

printf("usage:unset shell variables\n");

printf("options descriptions\n");

printf("/ not supported currently\n");

}

else if(strcmp(cmdname,"clr")==0){

printf("usage:clear screen\n");

printf("options descriptions\n");

printf("none see the manual,pls\n");

}

else if(strcmp(cmdname,"dir")==0){

printf("usage:list the file names in the target directory\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"environ")==0){

printf("usage:list all the environment variables\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"help")==0){

printf("usage:show the manual of help/get help info of a sepcified command\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"quit")==0){

printf("usage:quit the shell with thank-you information\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"myshell")==0){

printf("usage:execute a batchfile\n");

printf("options descriptions\n");

printf("none see the manual,plz\n");

}

else if(strcmp(cmdname,"mylove")==0){

printf("wish you find your Mr/Miss.Right :-)\n");

}

}

void myexec(){

//这有点像daemon（守护进程）

//但是两者是有区别的

//在这里我们要替换的是父进程

int res;

pid\_t pid;

if(cmd\_cnt==1){//如果只有一个exec，return

return;

}

else{

pid=fork();//fork一个进程出来，方便模拟exec

if(pid<0){

perror("fork");

exit(1);

}

else if(pid==0){//子进程

exit(0);//直接退出

}

else{//父进程

res=execvp(cmd\_array[1],cmd\_array+1);//用execvp()来执行这个命令

if(res<0){//如果执行失败

printf("myshell: command not found\n");//打印提示信息

}

exit(0);

}

}

}

void mytest(){

if(cmd\_cnt!=3){//命令中的字符串格式只有3个

printf("myshell: test: incorrect number of arguments\n");

printf("the format is 'test [op] [filename]'\n");

printf("for more information: use 'help test'\n");

}

else{

FILE\* fp;

struct stat buf;

char filename[MAX\_NAME\_LEN];

int mode;//用来保存调用stat()以后的返回结果

strcpy(filename,cmd\_array[2]);//获取文件名称

fp=fopen(filename,"r");//以只读方式打开文件

if(fp==NULL){//如果打不开

printf("myshell: test: file named %s doesn't exist\n",filename);//文件不存在

}

else{//如果文件存在

stat(filename,&buf);//储存相关信息到buf

mode=buf.st\_mode;

if(strcmp(cmd\_array[1],"-l")==0){//如果是符号链接（symbolic link）

mode=mode&S\_IFLNK;//位与操作

if(mode==S\_IFLNK){//检查得到的新值和S\_IFLNK是否匹配

printf("yes,this is a symbolic link\n");

}

else{

printf("no,this is NOT a symbolic link\n");

}

}

else if(strcmp(cmd\_array[1],"-b")==0){//检查是不是块设备（block device）

mode=mode&S\_IFBLK;//位与操作

if(mode==S\_IFBLK){//检查与S\_IFBLK是否匹配

printf("yes,this is a block device\n");

}

else{

printf("no,this is NOT a block device\n");

}

}

else if(strcmp(cmd\_array[1],"-c")==0){//检查是不是字符设备（character device）

mode=mode&S\_IFCHR;//位与操作

if(mode==S\_IFCHR){//检查与S\_IFCHR是否匹配

printf("yes,this is a character device\n");

}

else{

printf("no,this is NOT a character device\n");

}

}

else if(strcmp(cmd\_array[1],"-d")==0){//检查是不是目录文件（directory）

mode=mode&S\_IFDIR;//位与操作

if(mode==S\_IFDIR){//判断是不是和S\_IFDIR匹配

printf("yes,this is a directory\n");

}

else{

printf("no,this is NOT a directory\n");

}

}

else{//其他的非法输入

printf("myshell: test: only 4 options are allowed:\n");

printf("[-l],[-b],[-c],[-d]\n");

printf("for more information: use 'help test'\n");

}

}

}

}

void myumask(){

int bit1,bit2,bit3,bitsum;

mode\_t new\_umask,old\_umask;

if(cmd\_cnt==1){//只用一个umask可以查看默认值

printf("myshell: default umask value: %o\n",2);

return;

}

if(strlen(cmd\_array[1])!=4||cmd\_array[1][0]!='0'){//对于不正确的格式打印提示信息

printf("myshell: umask: the format is umask 0[digit1][digit2][digit3], eg., umask 0002\n");

return;

}

else{//获取每一位的值（用来得到最终的umask）

switch(cmd\_array[1][1]){

case'0':

bit1=0000;

break;

case'1':

bit1=0100;

break;

case'2':

bit1=0200;

break;

case'3':

bit1=0300;

break;

case'4':

bit1=0400;

break;

case'5':

bit1=0500;

break;

case'6':

bit1=0600;

break;

case'7':

bit1=0700;

break;

default:

printf("myshell: umask: out of range\n");

break;

}

switch(cmd\_array[1][2]){

case'0':

bit2=0000;

break;

case'1':

bit2=0010;

break;

case'2':

bit2=0020;

break;

case'3':

bit2=0030;

break;

case'4':

bit2=0040;

break;

case'5':

bit2=0050;

break;

case'6':

bit2=0060;

break;

case'7':

bit2=0070;

break;

default:

printf("myshell: umask: out of range\n");

break;

}

switch(cmd\_array[1][3]){

case'0':

bit3=0000;

break;

case'1':

bit3=0001;

break;

case'2':

bit3=0002;

break;

case'3':

bit3=0003;

break;

case'4':

bit3=0004;

break;

case'5':

bit3=0005;

break;

case'6':

bit3=0006;

break;

case'7':

bit3=0007;

break;

default:

printf("myshell: umask: out of range\n");

break;

}

bitsum=bit1+bit2+bit3;

new\_umask=bitsum;//新的umask是这三个值的和

printf("sum:%o\n",bitsum);

old\_umask=umask(new\_umask);//这个函数返回旧的umask值

}

printf("myshell: umask changed successfully\n");

printf("myshell: old value: %o\n",old\_umask);//打印旧的值

printf("myshell: new value: %o\n",new\_umask);//打印当前的新值

}

void myjobs(){

//可以使用ps命令来实现查看进程

pid\_t pid;

pid=fork();//必须fork，否则会出现myshell退出这种奇怪的bug

if(pid<0){

perror("myshell: fork");

}

else if(pid==0){//子进程

if(cmd\_cnt>1){

printf("myshell: jobs: incorrect use of jobs\n");

}

else{

execlp("ps","ps","ax",NULL);//使用ps

}

}

else{//父进程

waitpid(pid,NULL,0);

}

}

void myfg(pid\_t pid){

setpgid(pid,pid);

if (tcsetpgrp(1,getpgid(pid))== 0){

kill(pid,SIGCONT);//向对应的进程发送SIG\_CONT信号

waitpid(pid,NULL,WUNTRACED);//必须使用WUNTRACED

}

else{

printf("myshell: fg: no such job\n");

}

}

void mybg(pid\_t pid){

if(kill(pid,SIGCONT)<0){//发送SIGCONT信号

printf("myshell: bg: no such job\n");//如果有错就打印提示信息

}

else{

waitpid(pid,NULL,WUNTRACED);//和myfg()一样，必须用WUNTRACED

}

}

void mybatch(){

//这个函数用来支持命令"myshell"

//命令的格式是"myshell [filename]"

//您可以把需要执行的命令存入一个文件

//再用"myshell [filename]"一次性执行它们

FILE \*fp,\*helper;

char filename[MAX\_NAME\_LEN];

char cmdname[MAX\_LINE];

int fmark,cnt=0;

strcpy(filename,cmd\_array[1]);//获取文件名

fp=fopen(filename,"r");//以只读方式打开文件

helper=fopen(filename,"r");//helper在后面也要用到，这里再打开一遍

if(fp==NULL||helper==NULL){//如果打开失败

printf("myshell: myshell: no file named %s",filename);

}

else{//如果成功

int pos\_after\_pipe,bg,num=0;

pid\_t pid;

char\* cptr;

while((fmark=fgetc(fp))!=EOF){//获取文件中的命令个数

if(fmark=='\n'){

cnt++;

}

}

for(int i=0;i<cnt;i++){

memset(cmd\_array,0,MAX\_LINE/2+1);//必须要每次清空！

cmd\_cnt=0;//同时也要情况cmd\_cnt

num=0;//num也要清空

for(int i=0;i<MAX\_LINE&&cmdname[i]!='\0';i++){//清空cmdname

cmdname[i]='\0';

}

fgets(cmdname,MAX\_LINE,helper);//cmdname每次读取一行

cmdname[strlen(cmdname)-1]='\0';//必须减1，调整到正确位置

cptr=strtok(cmdname," ");//用空格分割命令

while(cptr!=NULL){//然后把分割好的内容存到cmd\_array

cmd\_array[num]=(char\*)malloc(sizeof(\*cptr));

strcpy(cmd\_array[num++],cptr);//需要注意的是即使直接回车num也是1，这样cmd\_cnt至少是1

cptr=strtok(NULL," ");

}

cmd\_cnt=num;//cmd\_cnt取得它的值，和num是相等的

if(is\_internal\_cmd()){

continue;

}

if(pos\_after\_pipe=is\_pipe()){

pipe(pipe\_fd);

}

if((pid=fork())==0){//子进程

int thispid=getpid();//获取子进程pid

signal(SIGINT,SIG\_DFL);//默认是终止进程

signal(SIGTSTP,SIG\_DFL);//默认是暂停进程

signal(SIGCONT,SIG\_DFL);//默认是继续这个进程

if(pos\_after\_pipe){//如果有管道

close(pipe\_fd[0]);//关闭读

dup2(pipe\_fd[1],1);//把stdout重定向到pipe\_fd[1]

}

if(bg==1){//如果命令需要在后台执行

printf("myshell: in background: the job's pid: [%d]\n",thispid);

run\_external\_cmd(0);

exit(0);

}

do\_redirection();

run\_external\_cmd(0);

break;

}

else if(pid>0){//父进程

signal(SIGINT,SIG\_IGN);//忽视信号

signal(SIGTSTP,SIG\_IGN);//忽视信号

signal(SIGCONT,SIG\_DFL);//默认是继续

if(bg==1){//如果命令要在后台执行

signal(SIGCHLD,SIG\_IGN);//忽视SIGCHLD信号

}

else{

waitpid(pid,NULL,WUNTRACED);//后台执行的关键代码，必须用WUNTRACED

}

do\_pipe(pos\_after\_pipe);//如果有pipe，在这里执行它

}

else{

perror("myshell: fork");

break;

}

}

}

}

void mydir(){

char pathname[MAX\_PATH\_LEN];//保存当前路径

DIR \*dir;//DIR struct保存关于目录的信息

struct dirent \*dp;

if(!getcwd(pathname,MAX\_PATH\_LEN)){//获取路径名

perror("myshell: getcwd");

exit(1);

}

dir=opendir(pathname);//返回指向DIR struct的指针

printf("the directory(ies) under the current path is(are):\n");

while((dp=readdir(dir))!=NULL){//列出得到的信息

printf("%s\n",dp->d\_name);

}

}

void mydir\_redirect(){

//这个函数支持了mydir()的重定向

//>>向文件末尾追加内容（如果文件存在的话，不存在就新建）

//而>是重写了整个文件内容

//下面的一些逻辑和mydir()类似

//但还要加上文件操作

int fd;//文件描述符

pid\_t pid;

char pathname[MAX\_PATH\_LEN];

char filename[MAX\_NAME\_LEN];

DIR \*dir;

struct dirent \*dp;

if(!getcwd(pathname,MAX\_PATH\_LEN)){//获取路径名

perror("myshell: getcwd");

exit(1);

}

dir=opendir(pathname);//返回指向DIR struct的指针

strcpy(filename,cmd\_array[2]);//获取文件名

if(strcmp(cmd\_array[1],">")==0){

if((fd=open(filename,O\_CREAT|O\_TRUNC|O\_WRONLY,0600))<0){//必须用O\_TRUNC

perror("myshell: open");

exit(1);

}

else{

pid=fork();//fork一个子进程执行重定向

if(pid==0){//子进程

while((dp=readdir(dir))!=NULL){//读取目录信息

dup2(fd,1);//把stdout重定向到文件

printf("%s\n",dp->d\_name);//这样一来打印内容其实去到了文件

}

exit(0);//必须exit

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//等待子进程

}

else{

printf("fork failed\n");

exit(1);

}

}

close(fd);//别忘了关闭文件

}

else{//带有>>的重定向，添加内容到文件

if((fd=open(filename,O\_CREAT|O\_APPEND|O\_WRONLY,0600))<0){//必须使用O\_APPEND

perror("myshell: open");

exit(1);

}

else{

pid=fork();//为了做重定向，fork

if(pid==0){//子进程

while((dp=readdir(dir))!=NULL){//读取目录

dup2(fd,1);//重定向stdout到文件

printf("%s\n",dp->d\_name);//这样一来print的内容就写进了文件

}

exit(0);

}

else if(pid>0){//父进程

waitpid(pid,NULL,0);//阻塞父进程，等待子进程

}

else{

printf("fork failed\n");

exit(1);

}

}

close(fd);//别忘了关闭文件

}

}

int main(){

int should\_run=1;//标记什么时候退出大循环

pid\_t pid;//fork的时候要用

int cmd\_len=0;//记录命令长度

int pos\_after\_pipe;//记录重定向符号|的位置

int bg;//后台执行的标记

welcome();//欢迎信息

while(should\_run){

printprompt();//打印命令提示符

readcommand();//读取命令，同时也取得了cmd\_cnt的值

cmd\_len=getcommandlen();

if(cmd\_len>MAX\_LINE){//如果用户输入长度超过规定的长度

printf("the length of your input is too long to be read in\n");

exit(1);

}

bg=is\_bg\_cmd();//如果bg=1，说明该命令需要在后台执行

if(is\_internal\_cmd()){//处理内部命令

continue;

}

if(pos\_after\_pipe=is\_pipe()){

pipe(pipe\_fd);

}

if((pid=fork())==0){//子进程

int thispid=getpid();//获取子进程pid

signal(SIGINT,SIG\_DFL);//默认:停止进程

signal(SIGTSTP,SIG\_DFL);//默认:终止进程

signal(SIGCONT,SIG\_DFL);//用默认的方式处理SIGCONT

if(pos\_after\_pipe){//如果有pipe

close(pipe\_fd[0]);//关闭读

dup2(pipe\_fd[1],1);//把stdout重定向到pipe\_fd[1]

}

if(bg==1){//如果需要在后台执行

printf("myshell: in background: the job's pid: [%d]\n",thispid);

run\_external\_cmd(0);//执行外部命令

return 0;

}

do\_redirection();//执行重定向（如果有的话）

run\_external\_cmd(0);

break;

}

else if(pid>0){//父进程

signal(SIGINT,SIG\_IGN);//忽视这个信号

signal(SIGTSTP,SIG\_IGN);//忽视这个信号

signal(SIGCONT,SIG\_DFL);//用默认的方式处理SIGCONT

if(bg==1){//如果需要在后台执行

signal(SIGCHLD,SIG\_IGN);//忽视SIGCHLD

}

else{

waitpid(pid,NULL,WUNTRACED);//后台执行的关键代码

}

do\_pipe(pos\_after\_pipe);//在这里处理带pipe的命令

}

else{

perror("myshell: fork");

break;

}

}

return 0;

}