**操作系统课程设计报告** 日期：19.6.6

|  |  |  |  |
| --- | --- | --- | --- |
| 实习题目：shell程序 | | | |
| 完成人 | 姓名：龙佳琪 | 班级：173040107 | 学号2017141461065 |
| 实习内容简要描述 | 使用C语言实现一个简易的shell命令窗口，功能与Linux系统下的命令行类似。每一次输入命令前出现命令行的提示符，且主要有以下功能：  ·能够执行外部程序命令，命令可以带参数  ·能够执行fg、bg、cd、history、exit等内部命令  ·使用管道和输入输出重定向。  ·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。 | | |
| 主要代码结构  （附注释） | 全局变量：        Commands记录的是当前的解析后的命令    所有的可能出错的情况用联合写在一起，最后通过switch-case进行错误信息的统一输出    获取当前的用户名、主机名、当前工作目录    （从上到下）：  寻找命令，用于判断当前读入的命令是否存在  获得用户名  获得主机名  获得当前工作目录  解析命令，即以空格为界把输入的命令分开并存入一个名为commands的二维数组中  退出  执行命令  执行含有管道符号的命令  执行含有重定向符号的命令  执行输出历史命令  执行后台命令  执行前台命令  执行查看作业命令  创建作业  通过while（1）的形式不断地等待用户输入命令，直到出错或者输入exit退出 | | |
| 结果分析（或错误原因分析） | 重定向命令：  应该注意的是，重定向符号的后边应该紧接着的就是文件名，且在一行指令中重定向符号可能多次出现  以输入重定向为例：  如果是输入重定向符’>’，那么就要以读的形式打开该符号的下一个参数（即它重定向指向的文件）为名的文件，然后开始执行指令，先将目标文件重定向为标准输入的方式打开，然后执行剩下的命令  同理可得输出的重定向实现    History命令：  这是要查看历史命令，那么只要每次将用户输入的命令都储存并以追加的方式更新，的一个不会被抹去信息的全局变量中即可，可采用循环数组的方式确保最大存储量    管道命令：  因为可能会存在不止一的管道，设计用递归实现，在指定下标中遍历寻找“|”的管道符号 | | |
| 注释：如果上述表格空间不够，可以另附表格进行说明 | | | |

设计：

通过字符串进行输入并分割后存放于一个指令集commands里边，逐一进行命令的比较，用strcmp函数，若一致便执行该命令

伪代码：

While(1){

Print$();

Input();

Split();

If(commands[0] = = exit)

退出

If(commands[0] = = x)

CallX();

Else if(xxx)

…..

调用相应函数

具体功能函数设计：

Cd命令：

由于cd命令是要改变工作目录，只要获取当前目录，并且在当前目录下对第二个参数，即commands[1]，这是将要改变的目录，在比较是否存在后调用系统所给的函数 chdir改变目录即可

History命令：

这是要查看历史命令，那么只要每次将用户输入的命令都储存并以追加的方式更新，的一个不会被抹去信息的全局变量中即可，可采用循环数组的方式确保最大存储量

管道命令：

因为可能会存在不止一的管道，设计用递归实现，在指定下标中遍历寻找“|”的管道符号，如果找到了便记录一下当前的管道符号的下标，对当前所得进行操作，再对该标号的下一个命令进行管道符号的寻找，终止条件是最左边的下标大于最右边的。

当前找到管道后的操作：创建一个子进程，且子程序执行下一个命令；其父程序得到下一个输入并递归执行后边的命令

重定向命令：

应该注意的是，重定向符号的后边应该紧接着的就是文件名，且在一行指令中重定向符号可能多次出现

以输入重定向为例：

如果是输入重定向符’>’，那么就要以读的形式打开该符号的下一个参数（即它重定向指向的文件）为名的文件，然后开始执行指令，先将目标文件重定向为标准输入的方式打开，然后执行剩下的命令

同理可得输出的重定向实现

其余的一些系统内部指令不需要自己实现，只需要判断是否存在于系统然后给出回应，若存在，那么就执行；不存在，报错

附代码:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/signal.h>

#include <sys/types.h>

#include <errno.h>

#include <pwd.h>

#include<time.h>

#define BUF\_SZ 256

#define TRUE 1

#define FALSE 0

const char\* COMMAND\_EXIT = "exit";

const char\* COMMAND\_HELP = "help";

const char\* COMMAND\_CD = "cd";

const char\* COMMAND\_IN = "<";

const char\* COMMAND\_OUT = ">";

const char\* COMMAND\_PIPE = "|";

const char\* COMMAND\_HIS = "history";

const char\* COMMAND\_JOBS = "jobs";

const char\* COMMAND\_BG = "bg";

const char\* COMMAND\_FG = "fg";

int ALL = 0;

enum jobstate{

READY,

RUNNING,

STOP

};

struct jobinfo{

int jid;

int pid;

int wait\_time;

time\_t creat\_time;

int run\_time;

enum jobstate state;

};

// 内置的状态码

enum {

RESULT\_NORMAL,

ERROR\_FORK,

ERROR\_COMMAND,

ERROR\_WRONG\_PARAMETER,

ERROR\_MISS\_PARAMETER,

ERROR\_TOO\_MANY\_PARAMETER,

ERROR\_CD,

ERROR\_SYSTEM,

ERROR\_EXIT,

/\* 重定向的错误信息 \*/

ERROR\_MANY\_IN,

ERROR\_MANY\_OUT,

ERROR\_FILE\_NOT\_EXIST,

/\* 管道的错误信息 \*/

ERROR\_PIPE,

ERROR\_PIPE\_MISS\_PARAMETER

};

int jidx=0,pidx=0;

char username[BUF\_SZ];

char hostname[BUF\_SZ];

char curPath[BUF\_SZ];

char commands[BUF\_SZ][BUF\_SZ];

char history[BUF\_SZ][BUF\_SZ];

struct jobinfo jobs[2];

int isCommandExist(const char\* command);

void getUsername();

void getHostname();

int getCurWorkDir();

int splitCommands(char command[BUF\_SZ]);

int callExit();

int callCommand(int commandNum);

int callCommandWithPipe(int left, int right);

int callCommandWithRedi(int left, int right);

int callCd(int commandNum);

void callHistory();

void callBg(int jid);

void callFg(int jid);

void callJobs(int jid);

int creatJobInfo(struct jobinfo \*jobInfo);

int main() {

/\* 获取当前工作目录、用户名、主机名 \*/

int result = getCurWorkDir();

if (ERROR\_SYSTEM == result) {

fprintf(stderr, "\e[31;1mError: System error while getting current work directory.\n\e[0m");

exit(ERROR\_SYSTEM);

}

getUsername();

getHostname();

/\* 启动myshell \*/

char argv[BUF\_SZ];

creatJobInfo(&jobs[0]);

creatJobInfo(&jobs[1]);

while (TRUE) {

printf("\e[32;1m%s@%s:%s\e[0m$ ", username, hostname,curPath); // 显示为绿色

/\* 获取用户输入的命令 \*/

fgets(argv, BUF\_SZ, stdin);

int len = strlen(argv);

if (len != BUF\_SZ) {

argv[len-1] = '\0';

}

int commandNum = splitCommands(argv);

if (commandNum != 0) { // 用户有输入指令

strcpy(history[ALL], argv);

ALL++;

if (strcmp(commands[0], COMMAND\_EXIT) == 0) { // exit命令

result = callExit();

if (ERROR\_EXIT == result) {

exit(-1);

}

}

else if (strcmp(commands[0], COMMAND\_HIS) == 0) { //history命令

callHistory();

}

else if (strcmp(commands[0], COMMAND\_BG) == 0) { //bg命令

int jid\_temp;

printf("请输入作业号：\n");

scanf("%d",&jid\_temp);

callBg(jid\_temp);

}

else if (strcmp(commands[0], COMMAND\_FG) == 0) { //fg命令

int jid\_temp;

printf("请输入作业号：\n");

scanf("%d",&jid\_temp);

callFg(jid\_temp);

}

else if (strcmp(commands[0], COMMAND\_JOBS) == 0) { //jobs命令

int jid\_temp;

printf("请输入作业号：\n");

scanf("%d",&jid\_temp);

callJobs(jid\_temp);

}

else if (strcmp(commands[0], COMMAND\_CD) == 0) { // cd命令

result = callCd(commandNum);

switch (result) {

case ERROR\_MISS\_PARAMETER:

fprintf(stderr, "\e[31;1mError: Miss parameter while using command \"%s\".\n\e[0m"

, COMMAND\_CD);

break;

case ERROR\_WRONG\_PARAMETER:

fprintf(stderr, "\e[31;1mError: No such path \"%s\".\n\e[0m", commands[1]);

break;

case ERROR\_TOO\_MANY\_PARAMETER:

fprintf(stderr, "\e[31;1mError: Too many parameters while using command \"%s\".\n\e[0m"

, COMMAND\_CD);

break;

case RESULT\_NORMAL: // cd命令正常执行，更新当前工作l目录

result = getCurWorkDir();

if (ERROR\_SYSTEM == result) {

fprintf(stderr

, "\e[31;1mError: System error while getting current work directory.\n\e[0m");

exit(ERROR\_SYSTEM);

} else {

break;

}

}

} else { // 其它命令

result = callCommand(commandNum);

switch (result) {

case ERROR\_FORK:

fprintf(stderr, "\e[31;1mError: Fork error.\n\e[0m");

exit(ERROR\_FORK);

case ERROR\_COMMAND:

fprintf(stderr, "\e[31;1mError: Command not exist in myshell.\n\e[0m");

break;

case ERROR\_MANY\_IN:

fprintf(stderr, "\e[31;1mError: Too many redirection symbol \"%s\".\n\e[0m", COMMAND\_IN);

break;

case ERROR\_MANY\_OUT:

fprintf(stderr, "\e[31;1mError: Too many redirection symbol \"%s\".\n\e[0m", COMMAND\_OUT);

break;

case ERROR\_FILE\_NOT\_EXIST:

fprintf(stderr, "\e[31;1mError: Input redirection file not exist.\n\e[0m");

break;

case ERROR\_MISS\_PARAMETER:

fprintf(stderr, "\e[31;1mError: Miss redirect file parameters.\n\e[0m");

break;

case ERROR\_PIPE:

fprintf(stderr, "\e[31;1mError: Open pipe error.\n\e[0m");

break;

case ERROR\_PIPE\_MISS\_PARAMETER:

fprintf(stderr, "\e[31;1mError: Miss pipe parameters.\n\e[0m");

break;

}

}

}

}

}

int isCommandExist(const char\* command) { // 判断指令是否存在

if (command == NULL || strlen(command) == 0) return FALSE;

int result = TRUE;

int fds[2];

if (pipe(fds) == -1) {

result = FALSE;

} else {

/\* 暂存输入输出重定向标志 \*/

int inFd = dup(STDIN\_FILENO);

int outFd = dup(STDOUT\_FILENO);

pid\_t pid = vfork();

if (pid == -1) {

result = FALSE;

} else if (pid == 0) {

/\* 将结果输出重定向到文件标识符 \*/

close(fds[0]);

dup2(fds[1], STDOUT\_FILENO);

close(fds[1]);

char tmp[BUF\_SZ];

sprintf(tmp, "command -v %s", command);

system(tmp);

exit(1);

} else {

waitpid(pid, NULL, 0);

/\* 输入重定向 \*/

close(fds[1]);

dup2(fds[0], STDIN\_FILENO);

close(fds[0]);

if (getchar() == EOF) { // 没有数据，意味着命令不存在

result = FALSE;

}

/\* 恢复输入、输出重定向 \*/

dup2(inFd, STDIN\_FILENO);

dup2(outFd, STDOUT\_FILENO);

}

}

return result;

}

void getUsername() { // 获取当前登录的用户名

struct passwd\* pwd = getpwuid(getuid());

strcpy(username, pwd->pw\_name);

}

void getHostname() { // 获取主机名

gethostname(hostname, BUF\_SZ);

}

int getCurWorkDir() { // 获取当前的工作目录

char\* result = getcwd(curPath, BUF\_SZ);

if (result == NULL)

return ERROR\_SYSTEM;

else return RESULT\_NORMAL;

}

int splitCommands(char command[BUF\_SZ]) { // 以空格分割命令， 返回分割得到的字符串个数

int num = 0;

int i, j;

int len = strlen(command);

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

if (command[i] != ' ') {

commands[num][j++] = command[i];

} else {

if (j != 0) {

commands[num][j] = '\0';

++num;

j = 0;

}

}

}

if (j != 0) {

commands[num][j] = '\0';

++num;

}

return num;

}

int callExit() { // 发送terminal信号退出进程

pid\_t pid = getpid();

if (kill(pid, SIGTERM) == -1)

return ERROR\_EXIT;

else return RESULT\_NORMAL;

}

int callCommand(int commandNum) { // 给用户使用的函数，用以执行用户输入的命令

pid\_t pid = fork();

if (pid == -1) {

return ERROR\_FORK;

} else if (pid == 0) {

/\* 获取标准输入、输出的文件标识符 \*/

int inFds = dup(STDIN\_FILENO);

int outFds = dup(STDOUT\_FILENO);

int result = callCommandWithPipe(0, commandNum);

/\* 还原标准输入、输出重定向 \*/

dup2(inFds, STDIN\_FILENO);

dup2(outFds, STDOUT\_FILENO);

exit(result);

} else {

int status;

waitpid(pid, &status, 0);

return WEXITSTATUS(status);

}

}

int callCommandWithPipe(int left, int right) { // 所要执行的指令区间[left, right)，可能含有管道

if (left >= right) return RESULT\_NORMAL;

/\* 判断是否有管道命令 \*/

int pipeIdx = -1;

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

if (strcmp(commands[i], COMMAND\_PIPE) == 0) {

pipeIdx = i;

break;

}

}

if (pipeIdx == -1) { // 不含有管道命令

return callCommandWithRedi(left, right);

} else if (pipeIdx+1 == right) { // 管道命令'|'后续没有指令，参数缺失

return ERROR\_PIPE\_MISS\_PARAMETER;

}

/\* 执行命令 \*/

int fds[2];

if (pipe(fds) == -1) {

return ERROR\_PIPE;

}

int result = RESULT\_NORMAL;

pid\_t pid = vfork();

if (pid == -1) {

result = ERROR\_FORK;

} else if (pid == 0) { // 子进程执行单个命令

close(fds[0]);

dup2(fds[1], STDOUT\_FILENO); // 将标准输出重定向到fds[1]

close(fds[1]);

result = callCommandWithRedi(left, pipeIdx);

exit(result);

} else { // 父进程递归执行后续命令

int status;

waitpid(pid, &status, 0);

int exitCode = WEXITSTATUS(status);

if (exitCode != RESULT\_NORMAL) { // 子进程的指令没有正常退出，打印错误信息

char info[4096] = {0};

char line[BUF\_SZ];

close(fds[1]);

dup2(fds[0], STDIN\_FILENO); // 将标准输入重定向到fds[0]

close(fds[0]);

while(fgets(line, BUF\_SZ, stdin) != NULL) { // 读取子进程的错误信息

strcat(info, line);

}

printf("%s", info); // 打印错误信息

result = exitCode;

} else if (pipeIdx+1 < right){

close(fds[1]);

dup2(fds[0], STDIN\_FILENO); // 将标准输入重定向到fds[0]

close(fds[0]);

result = callCommandWithPipe(pipeIdx+1, right); // 递归执行后续指令

}

}

return result;

}

int callCommandWithRedi(int left, int right) { // 所要执行的指令区间[left, right)，不含管道，可能含有重定向

if (!isCommandExist(commands[left])) { // 指令不存在

return ERROR\_COMMAND;

}

/\* 判断是否有重定向 \*/

int inNum = 0, outNum = 0;

char \*inFile = NULL, \*outFile = NULL;

int endIdx = right; // 指令在重定向前的终止下标

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

if (strcmp(commands[i], COMMAND\_IN) == 0) { // 输入重定向

++inNum;

if (i+1 < right)

inFile = commands[i+1];

else return ERROR\_MISS\_PARAMETER; // 重定向符号后缺少文件名

if (endIdx == right) endIdx = i;

} else if (strcmp(commands[i], COMMAND\_OUT) == 0) { // 输出重定向

++outNum;

if (i+1 < right)

outFile = commands[i+1];

else return ERROR\_MISS\_PARAMETER; // 重定向符号后缺少文件名

if (endIdx == right) endIdx = i;

}

}

/\* 处理重定向 \*/

if (inNum == 1) {

FILE\* fp = fopen(inFile, "r");

if (fp == NULL) // 输入重定向文件不存在

return ERROR\_FILE\_NOT\_EXIST;

fclose(fp);

}

if (inNum > 1) { // 输入重定向符超过一个

return ERROR\_MANY\_IN;

} else if (outNum > 1) { // 输出重定向符超过一个

return ERROR\_MANY\_OUT;

}

int result = RESULT\_NORMAL;

pid\_t pid = vfork();

if (pid == -1) {

result = ERROR\_FORK;

} else if (pid == 0) {

/\* 输入输出重定向 \*/

int p;

char\* temp;

if (inNum == 1){

freopen(inFile, "r", stdin);

scanf("%d",&p);

fclose(stdin);

}

if (outNum == 1)

freopen(outFile, "w", stdout);

/\* 执行命令 \*/

char\* comm[BUF\_SZ];

//sprintf(temp,"%d",p);

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

comm[i] = commands[i];

comm[endIdx] = NULL;

char \*t[]={comm[0],"100",NULL};//comm+left

execvp(comm[left], comm+left);

exit(errno); // 执行出错，返回errno

} else {

int status;

waitpid(pid, &status, 0);

int err = WEXITSTATUS(status); // 读取子进程的返回码

if (err) { // 返回码不为0，意味着子进程执行出错，用红色字体打印出错信息

printf("\e[31;1mError: %s\n\e[0m", strerror(err));

}

}

return result;

}

int callCd(int commandNum) { // 执行cd命令

int result = RESULT\_NORMAL;

if (commandNum < 2) {

result = ERROR\_MISS\_PARAMETER;

} else if (commandNum > 2) {

result = ERROR\_TOO\_MANY\_PARAMETER;

} else {

int ret = chdir(commands[1]);

if (ret) result = ERROR\_WRONG\_PARAMETER;

}

return result;

}

void callHistory()

{

printf("%d\n",ALL);

if (ALL == 0)

{

printf("\e[31;1mError: No history !\e[0m");

}

int i=0;

do

{

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

i++;

}while (i < ALL);

}

int creatJobInfo(struct jobinfo \*jobInfo)

{

jobInfo->jid=jidx;

jidx+=1;

jobInfo->pid=pidx;

pidx+=1;

jobInfo->wait\_time=0;

time(&jobInfo->creat\_time);

jobInfo->run\_time=rand()%50;//随机生成运行时间

jobInfo->state=READY;

}

void callBg(int jid)

{

jobs[jid].state=STOP;

}

void callFg(int jid)

{

jobs[jid].state=READY;

}

void callJobs(int jid)

{

printf("pid: %d\n",jobs[jid].pid);

printf("jid: %d\n",jobs[jid].jid);

printf("run time: %d\n",jobs[jid].run\_time);

printf("wait time: %d\n",jobs[jid].wait\_time);

printf("creat time: %ld\n",jobs[jid].creat\_time);

printf("job state: %d\n",jobs[jid].state);

printf("\n");

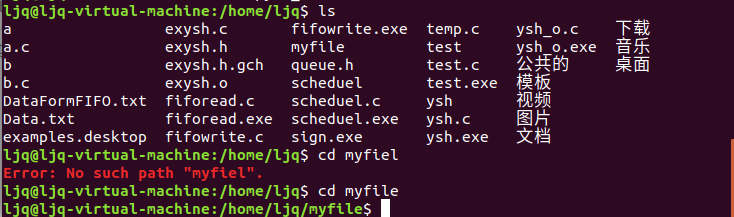
}

运行结果截屏：

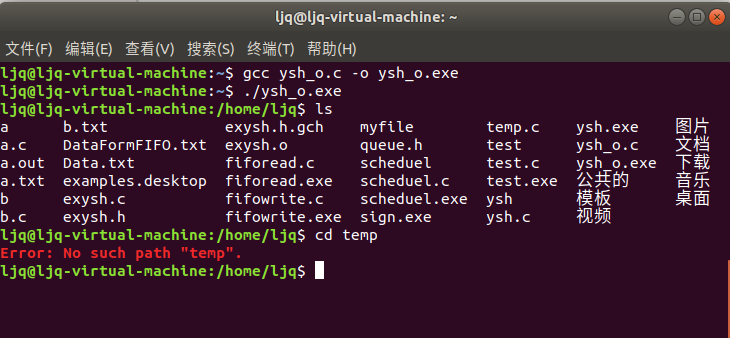
编译运行：



查看和改变目录：



如果目录不存在：



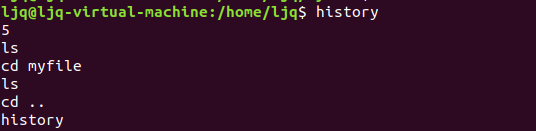
缺少目录名：



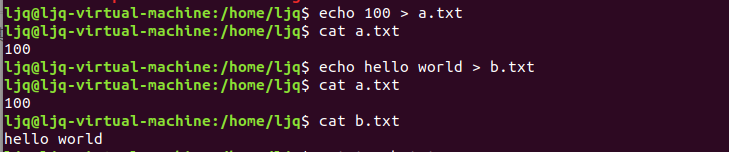
命令带参数：

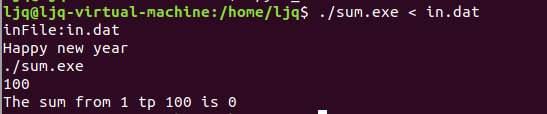


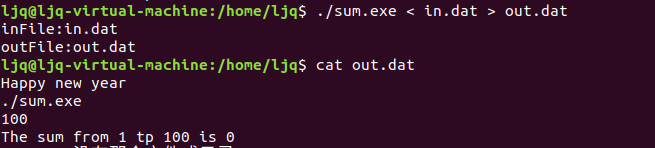
查看历史输入：



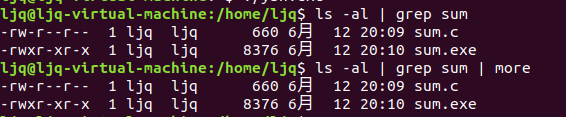
重定向：



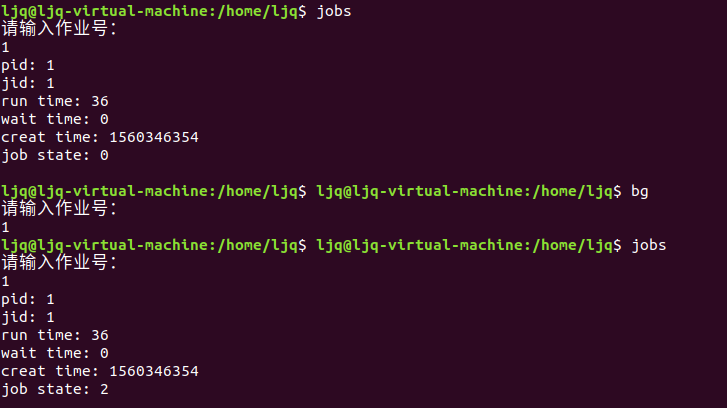




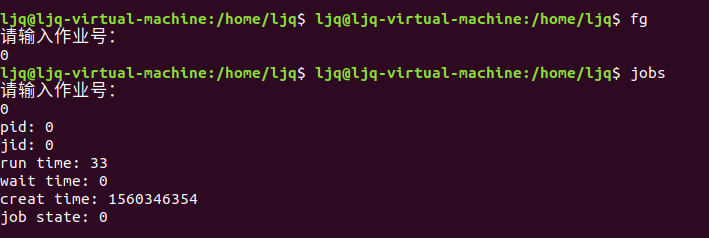
管道：



Bg和查看作业：



Fg和查看作业



退出

