## Mini Shell

#### 实验目的

本次实验旨在开发一个简单的 Mini Shell，支持基本的命令和功能，如欢迎提示、退出提示、路径切换（cd命令及其变种）、文件列表显示（ls命令）、文本输出（echo命令）、文件查看（cat命令）、当前路径显示（pwd命令）以及输出重定向（>和>>命令）。同时，需要考虑错误命令和错误路径时的友好提示。

#### 实验分析

* 全局变量的管理：为了实现 Shell 的路径功能，需要保存一个全局变量表示路径。
* 进程管理：为从输入读取命令并执行，对每个命令需要 fork 一个子进程来执行。
* 命令解析和执行：需要对用户输入的命令进行解析和执行，区分内部和外部命令。
* 用户友好的提示符：支持以当前路径和用户名为提示符，方便用户定位。

#### 实验结构

* 主函数：输出欢迎语句，获取本地shell初始地址，获取输入，进行命令识别。
* split函数：讲读取的多个命令进行分割储存，返回vector
* eval函数：运行命令，通过对关键词进行识别，区分shell内部和bash命令，分别进行处理和输出。

#### 实现的功能

* 欢迎进入：用户进入 Mini Shell 时，显示欢迎提示。
* 退出提示：当用户输入 exit 命令时，显示退出提示。
* 路径切换：

cd：切换到用户的登录目录。

cd ~：切换到用户的登录目录。

cd ../：切换到上级目录。

cd /：切换到根目录。

cd +目录：切换到指定目录。

* 文件列表显示：

ls：显示当前目录下的文件列表。

ls -l：显示详细的文件列表。

* 文本输出：

echo：输出用户输入的文本。

* 文件查看：

cat：查看指定文件的内容。

* 当前路径显示：

pwd：显示当前工作路径。

* 输出重定向：

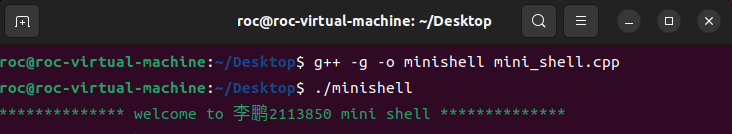
>：将命令输出写入文件，如果文件存在则覆盖。

>>：将命令输出追加到文件末尾。

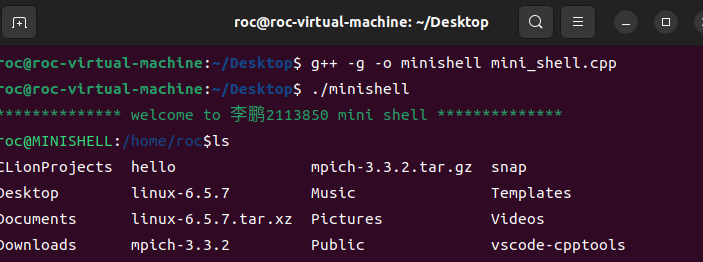
* 错误命令和错误路径提示：当用户输入错误的命令或路径时，给予友好的提示信息。

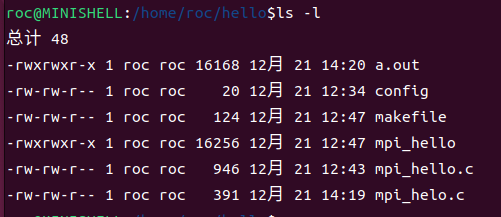
#### 实验结果

编译并启动minishell

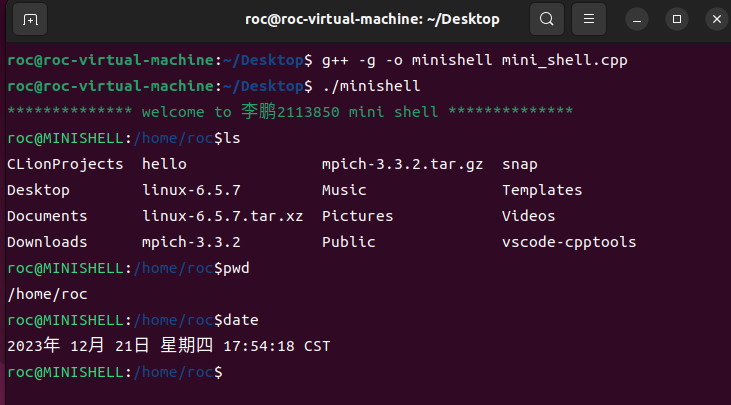


ls 命令

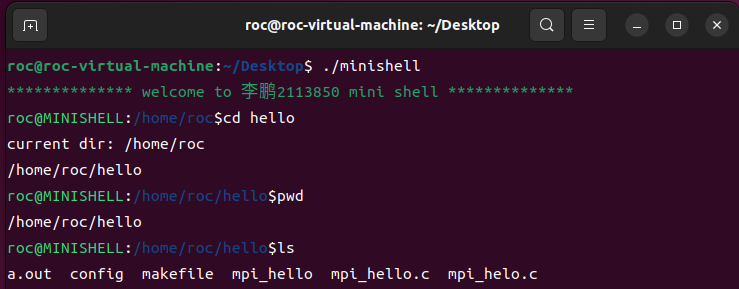


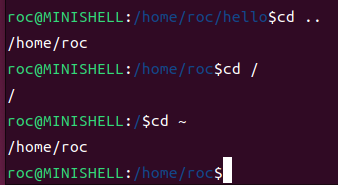


pwd date命令

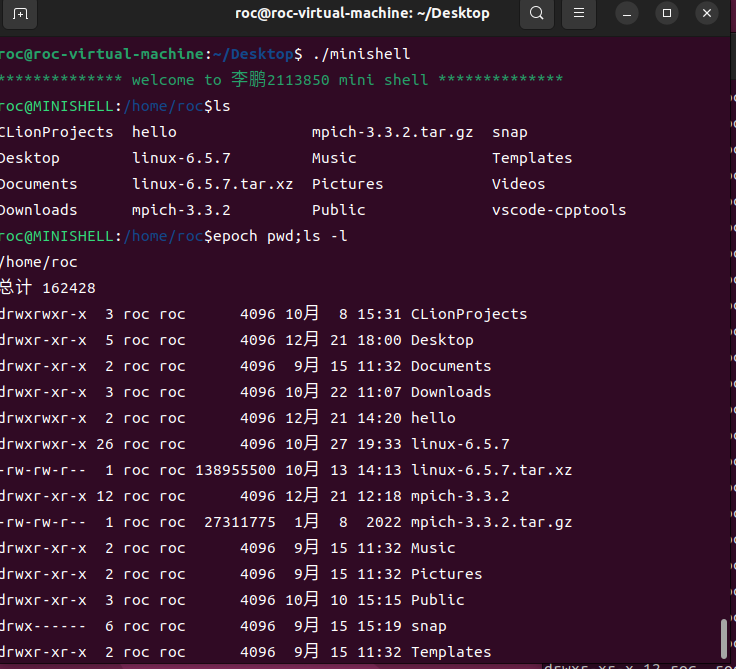


cd 命令

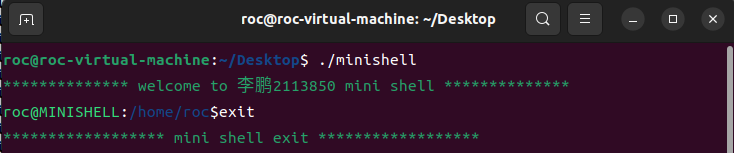




pwd； ls -l 顺序执行（以分号分隔）



exit 命令



#### 实验总结

通过这次实验，进一步了解了 Linux 操作系统的基本原理，加深了对进程、文件操作和字符串处理等方面的理解。对于平时学习过程中使用到的各种shell客户端有了更多的认识，了解了其背后的原理，同时对 C 语言编程能力也有所提升。

源代码：

#include <dirent.h>

#include <libgen.h>

#include <iostream>

#include <vector>

#include <cstring>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <cstdlib>

#include <sstream>

#include <pwd.h>

#include <fcntl.h>

using namespace std;

#define SUCCESS 0

#define ERROR -1

int reflag = 0;

char current\_dir[100];

char user\_dir[100];

vector<string> re;

int eval(vector<string> res);

vector<string> split(const string &s, const string &separator);

int eval(vector<string> res) {

if (res[0] == "cd") {

if (res.size() == 1) {

strcpy(current\_dir, user\_dir);

} else {

const char \*rest = res[1].c\_str();

if (res[1] == "/") {

opendir(rest);

strcpy(current\_dir, rest);

} else if (res[1] == "..") {

char \*parent\_dir = dirname(current\_dir);

strcpy(current\_dir, parent\_dir);

} else if (res[1] == "~") {

strcpy(current\_dir, user\_dir);

} else {

char target\_path[100];

cout << "current dir: " << current\_dir << endl;

if (strcmp(current\_dir, "/") == 0) {

snprintf(target\_path, 1024, "%s%s", current\_dir, rest);

} else {

snprintf(target\_path, 1024, "%s/%s", current\_dir, rest);

}

if (opendir(target\_path) == NULL) {

cout << "cd: " << rest << ":";

printf("\033[31m 没有那个文件或目录.\n\033[0m");

return ERROR;

}

strcpy(current\_dir, target\_path);

}

cout << current\_dir << endl;

return ERROR;

}

} else if (res[0] == "pwd") {

char buf[300];

cout << current\_dir << endl;

} else if (res[0] == "ls") {

int pid = fork(), wpid;

int status;

int count = 0;

const char \*rest = res[0].c\_str();

if (pid == 0) {

char \*env[] = {0, NULL};

for (int i = 0; i < res.size(); i++) {

if (res[i] == ">") {

reflag = 1;

count = i;

}

if (res[i] == ">>") {

reflag = 2;

count = i;

}

}

if (reflag != 0) {

char \*\*cmd\_temp = new char \*[count];

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

cmd\_temp[i] = new char[500];

memset(cmd\_temp[i], 0, sizeof(cmd\_temp[i]));

}

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

strcpy(cmd\_temp[i], res[i].c\_str());

}

cmd\_temp[count] = current\_dir;

cmd\_temp[count + 1] = NULL;

int fd = 1;

if (reflag == 1)

fd = open(res[count + 1].c\_str(), O\_CREAT | O\_WRONLY | O\_TRUNC, 0664);

else if (reflag == 2)

fd = open(res[count + 1].c\_str(), O\_CREAT | O\_WRONLY | O\_APPEND, 0664);

dup2(fd, 1);

if (execvp(rest, cmd\_temp) < 0) {

printf("\033[31m%s:command not found.\n\033[0m", res[0].c\_str());

}

} else {

char \*\*cmd\_temp = new char \*[res.size() + 1];

for (int i = 0; i < res.size(); i++) {

cmd\_temp[i] = new char[500];

memset(cmd\_temp[i], 0, sizeof(cmd\_temp[i]));

}

for (int i = 0; i < res.size(); i++) {

strcpy(cmd\_temp[i], res[i].c\_str());

}

cmd\_temp[res.size()] = current\_dir;

cmd\_temp[res.size() + 1] = NULL;

if (execvp(rest, cmd\_temp) < 0) {

printf("\033[31m%s:command not found.\n\033[0m", res[0].c\_str());

}

}

} else if (pid > 0) {

do {

wpid = waitpid(pid, &status, WUNTRACED);

} while (!WIFEXITED(status) && !WIFSIGNALED(status));

}

} else {

int pid = fork(), wpid;

int status;

const char \*rest = res[0].c\_str();

if (pid == 0) {

char \*\*cmd\_temp = new char \*[res.size()];

char \*env[] = {0, NULL};

for (int i = 0; i < res.size(); i++) {

cmd\_temp[i] = new char[500];

memset(cmd\_temp[i], 0, sizeof(cmd\_temp[i]));

}

for (int i = 0; i < res.size(); i++) {

strcpy(cmd\_temp[i], res[i].c\_str());

}

cmd\_temp[res.size()] = NULL;

if (execvp(rest, cmd\_temp) < 0) {

printf("\033[31m%s:command not found.\n\033[0m", res[0].c\_str());

}

} else if (pid > 0) {

do {

wpid = waitpid(pid, &status, WUNTRACED);

} while (!WIFEXITED(status) && !WIFSIGNALED(status));

}

}

return SUCCESS;

}

vector<string> split(const string &s, const string &separator) {

vector<string> result;

typedef string::size\_type string\_size;

string\_size i = 0;

while (i != s.size()) {

int flag = 0;

while (i != s.size() && flag == 0) {

flag = 1;

for (string\_size x = 0; x < separator.size(); ++x)

if (s[i] == separator[x]) {

++i;

flag = 0;

break;

}

}

flag = 0;

string\_size j = i;

while (j != s.size() && flag == 0) {

for (string\_size x = 0; x < separator.size(); ++x)

if (s[j] == separator[x]) {

flag = 1;

break;

}

if (flag == 0)

++j;

}

if (i != j) {

result.push\_back(s.substr(i, j - i));

i = j;

}

}

return result;

}

int main() {

string cmdstring;

printf("\033[32m\*\*\*\*\*\*\*\*\*\*\*\*\*\* welcome to 李鹏2113850 mini shell \*\*\*\*\*\*\*\*\*\*\*\*\*\* \n\033[0m");

strcpy(current\_dir, getpwuid(getuid())->pw\_dir);

strcpy(user\_dir, getpwuid(getuid())->pw\_dir);

printf("\033[92m%s@MINISHELL\033[0m:\033[34m%s\033[0m$", getlogin(), current\_dir);

while (1) {

for (int i = 0; i < re.size(); i++) {

re[i].clear();

}

getline(cin, cmdstring);

string result;

vector<string> v = split(cmdstring, ";");

for (int i = 0; i < v.size(); i++) {

re.clear();

stringstream input2(v[i]);

while (input2 >> result) {

re.push\_back(result);

}

if (result == "exit") {

printf("\033[32m\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* mini shell exit \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\033[0m");

return 0;

}

if (re.size()) {

eval(re);

}

}

printf("\033[92m%s@MINISHELL\033[0m:\033[34m%s\033[0m$", getlogin(), current\_dir);

}

return 0;

}