lab09 Mini Shell

软件工程2018级 1813075 刘茵

实验目的

* Implement a mini shell which is simlar to bash shell

实验分析

1. 为了实现shell的路径功能，需要保存一个全局变量表示路径
2. 为了从输入读取命令并执行，对每个命令需要fork一个子进程

实验结构

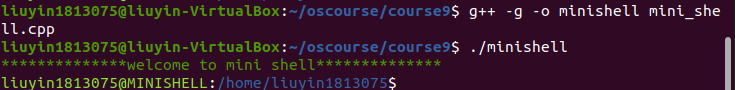
1. 主函数：输出欢迎语句，获取本地shell初始地址，获取输入，进行命令识别。
2. split函数：讲读取的多个命令进行分割储存，返回vector
3. eval函数：运行命令，通过对关键词进行识别，区分shell内部和bash命令，分别进行处理和输出。

实现功能

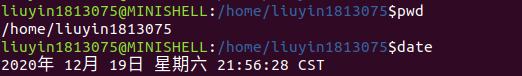
* 欢迎进入
* 退出提示
* cd
* cd ~
* cd ../
* cd /
* cd +目录
* ls 多项功能
* echo
* cat
* pwd
* 重定向 >
* 重定向 >>
* 错误命令提示
* 错误路径提示

**操作截图：**

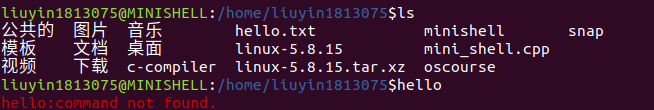
1. 程序启动，输出欢迎提示，设定所有地址启动minishell都预先返回home/liuyin1813075



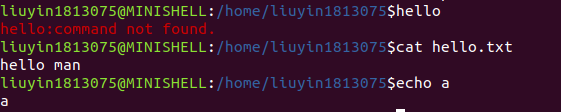
1. pwd date功能



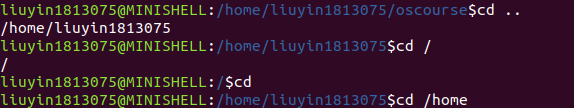
1. ls命令，无效操作命令的提示



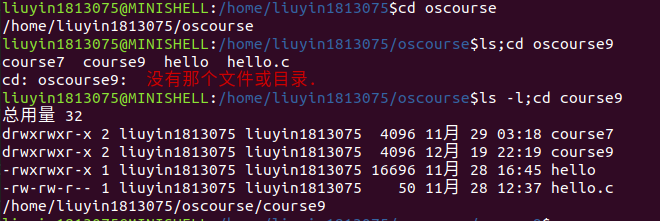
1. cat命令查看文件并打印，echo函数

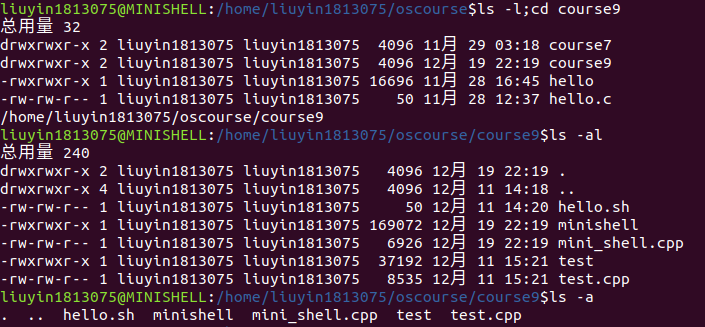


1. cd多种命令，ls 多命令

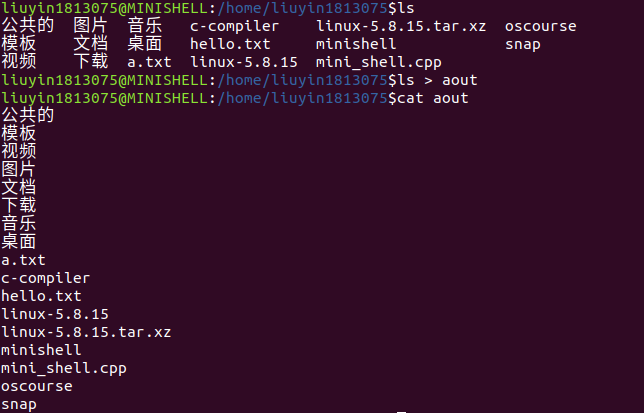


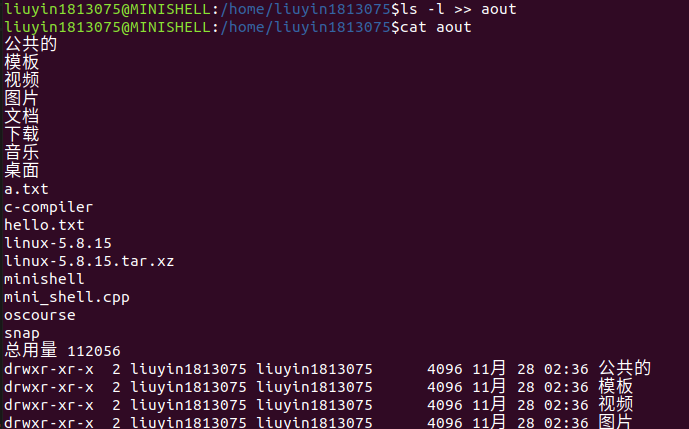






1. 重定向 覆写和增加





附：c++代码：

#include <iostream>

#include <cstdio>

#include <string.h>

#include <unistd.h>

#include <pwd.h>

#include <libgen.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <vector>

#include <sstream>

#include <dirent.h>

#include <cstring>

#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*);

int eval(vector<string> *res*)

{

    if (*res*[0] == "cd")

    {

*// 当前系统目录*

*// char target\_path[100];*

*// getcwd(target\_path, 100);*

*// cout<<"now path"<<target\_path<<endl;*

        if (*res*.size() == 1)

        {

            strcpy(current\_dir, user\_dir);

        }

        else

        {

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

*// cout << "rest:" << rest << endl;*

*// printf("\033[31mno such directory\n\033[0m");*

*// printf("could open\n");*

            if (*res*[1] == "/")

            {

                opendir(rest);

                strcpy(current\_dir, rest);

*// cout<<"current / is"<<current\_dir<<endl;*

            }

            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 dit:" << 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();

*// printf(rest); //print ok*

        if (pid == 0)

        {

            char \*env[] = {0, NULL};

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

            {

*// cout<<"res"<<res[i]<<endl;*

                if (*res*[i] == ">")

                {

*// cout<<"check <"<<endl;*

                    reflag = 1;

                    count = i;

                }

                if (*res*[i] == ">>")

                {

                    reflag = 2;

                    count = i;

                }

            }

            if (reflag != 0)

            {

*// cout<<"reflag!=0"<<endl;*

                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++)

                {

*// cout << "now path" << res[i].c\_str() << endl;*

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

                }

                cmd\_temp[count] = current\_dir;

                cmd\_temp[count + 1] = NULL;

*// 标准输出重定向，将原本要写入标准输出1的数据写入新文件(fd)中*

                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

            {

*// cout<<"reflag00"<<endl;*

                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++)

                {

*// cout << "now path" << res[i].c\_str() << endl;*

                    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();

*// printf(rest); //print ok*

        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 &*seperator*)

{

    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 < *seperator*.size(); ++x)

                if (*s*[i] == *seperator*[x])

                {

                    ++i;

                    flag = 0;

                    break;

                }

        }

        flag = 0;

        string\_size j = i;

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

        {

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

                if (*s*[j] == *seperator*[x])

                {

                    flag = 1;

                    break;

                }

            if (flag == 0)

                ++j;

        }

        if (i != j)

        {

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

            i = j;

        }

    }

*// 打印split命令 √*

*// for (int i = 0; i < result.size(); ++i)*

*// {*

*//     cout << i << ":" << result[i] << " " << endl;*

*// }*

    return result;

}

int main()

{

    string cmdstring;

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

*//test*

*// printf("hello!\n");*

    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);*//input string with ' '*

        string result;

        vector<string> v = split(cmdstring, ";");*//slipt command*

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

        {

            re.clear();

            stringstream input2(v[i]);*//string stream  initialize 不按照空格划分*

            while (input2 >> result)

            {

                re.push\_back(result);

            }

            if (result == "exit")

            {

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

                return 0;

            }

            if (re.size())

            {

*// int a = re.size();*

*// cout << "size:" << a << endl;*

                eval(re);

            }

        }

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

    }

    return 0;

}