《操作系统》课第09次实验报告

学院:	软件学院
姓名:	张怡桢
学号:	2013747
邮箱:	2662765987@qq.com
时间:	11/17/2022

1. 开篇感言

“你长大后想成为什么人？”

“什么意思？长大后我就不能成为我自己了吗？”

-- 《阿甘正传》

2. 实验题目

在 Linux 平台上，采用 C 语言编写一个 Mini Shell 命令解释环境（即类似 Bash Shell 环境）。该环境可以循环接受用户（从标准输入中）输入的（外部和内部） 命令以及若干参数，然后能对上述命令进行解析和执行，最后将用户输入的命令 的执行结果显示在标准输出上。即：

```
bash-2.03$ ps_03
**************welcome to mini shell**************
MINI SHELL#pwd /home/unixmng/oscourses/ps_prog
MINI SHELL#exit
****************** mini shell exit******************
bash-2.03$
```

3. 实验要求

1. 支持用户输入一行命令及其多个参数，并解析执行，并输出结果；
2. 支持 cd 命令，若无参数则回到当前用户的登录目录（见下面提示）；
3. 支持以“当前路径”和“用户名”为提示符；支持对命令行中空格的自动忽略处理；

4. 支持对命令行中 tab 键的自动忽略处理；
5. 支持一行中以“；”（为标志）分隔的多个命令及多个参数的顺序执行，即如下：
    a. MINI SHELL#pwd; ls –l;date
    b. 说明：上述三个命令须在本 Mini Shell 下依次顺序执行，最后由 Mini Shell 再次循环接受用户的新命令。

4. 原理方法

4.1 相关函数

1. 相关头文件：

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <pwd.h>

2. 相关参考函数：
    a. fork 、execvp、wait ：创建进程
    b. strcmp、strcpy、strncpy ：字符串相关操作
    c. fopen、fclose、fscanf、fprintf、fgets、fputs：文件 stdio 操作
    d. sprintf ：任意类型转换为字符串
    e. atoi ：字符串转换为整数类型 int
    f. getlogin ：获取当前用户名
    g. getcwd ：获取当前路径
    h. chdir ：改变当前路径
    i. getenv ：获取环境变量
    j. 获取当前用户的登录目录

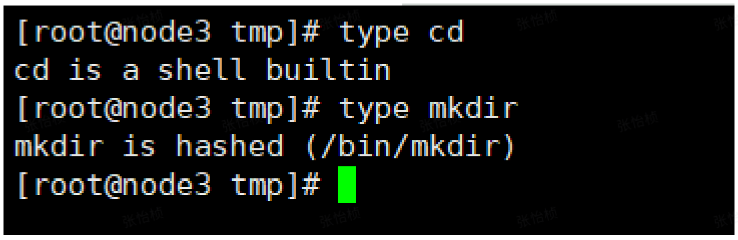
```
//////////////////////get user default dir
int getuserdir(char *aoUserDir)
{
    char *LoginId;
    struct passwd *pwdinfo;
    if (aoUserDir == NULL)
        return -9;
    if ((LoginId = getlogin()) == NULL)
    {
        perror("getlogin");

        aoUserDir[0] = '\0';

        return -8;
    }
    if ((pwdinfo = getpwnam(LoginId)) == NULL)
    {

        perror("getpwnam");

        return -7;
    }
    strcpy(aoUserDir, pwdinfo->pw_dir);
}
```

4.2 shell内置命令和外部命令的区别

内部命令实际上是shell程序的一部分，其中包含的是一些比较简单的linux系统命令，这些命令由shell程序识别并在shell程序内部完成运行，通常在linux系统加载运行时shell就被加载并驻留在系统内存中。内部命令是写在bashy源码里面的，其执行速度比外部命令快，因为解析内部命令shell不需要创建子进程。比如：exit，history，cd，echo等。

外部命令是linux系统中的实用程序部分，因为实用程序的功能通常都比较强大，所以其包含的程序量也会很大，在系统加载时并不随系统一起被加载到内存中，而是在需要时才将其调用内存。通常外部命令的实体并不包含在shell中，但是其命令执行过程是由shell程序控制的。shell程序管理外部命令执行的路径查找、加载存放，并控制命令的执行。外部命令是在bash之外额外安装的，通常放在/bin，/usr/bin，/sbin，/usr/sbin......等等。可通过“echo $PATH”命令查看外部命令的存储路径，比如：ls、vi等。

用type命令可以分辨内部命令与外部命令：



```
[root@node3 tmp]# type cd
cd is a shell builtin
[root@node3 tmp]# type mkdir
mkdir is hashed (/bin/mkdir)
[root@node3 tmp]# 
```

内部命令和外部命令最大的区别之处就是性能。内部命令由于构建在shell中而不必创建多余的进程，要比外部命令执行快得多。因此和执行更大的脚本道理一样，执行包含很多外部命令的脚本会损害脚本的性能。

5. 代码分析

5.1 全局定义

全局变量：

current_dir：当前所在的系统路径

user_dir：当前所在的系统路径

cmdline：从终端读入的字符串

separator：终端读入多条命令的规定分隔符“；”

commands：解析cmdline后得到的string数组

command：commands的其中一个元素

child_commands:解析单条命令command的参数

builtins[]:内部命令的结构体数组，用于处理内部命令

全局函数：

void init()：初始化函数

int execute_command()：执行命令函数

void read_command()：读取命令函数

void parse_command()：解析命令函数

int builtin(void) ：内部命令

void do_cd()：内部命令的cd函数

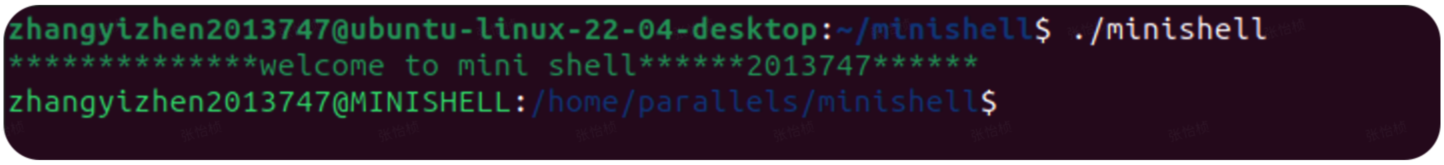
void do_ls()：内部命令的ls函数

```
//全局变量定义

int re_flag = 0;
char current_dir[100];
char user_dir[100];

string cmdline; //command input
string separator=";";
string command;


vector<string> commands; // 所有命令
vector<string> child_commands; //仅一个命令

//结构体定义
typedef void (CMD_HANDLER)(void);
typedef struct builtin_cmd
{
    char* name;
    CMD_HANDLER *handler;
} BUILTIN_CMD;

//全局函数声明
void init();
int execute_command();
void read_command();
void parse_command();
int builtin(void);
void do_cd();
void do_ls();

//内部命令解析
BUILTIN_CMD builtins[] = {
        {"cd", do_cd},
        {"ls", do_ls},
        {NULL, NULL}

};
```

5.2 获取当前文件的路径以及系统用户名

```
strcpy(current_dir, getcwd(NULL, 0));
strcpy(user_dir, getcwd(NULL, 0));
printf("\033[92m%s@MINISHELL\033[0m:\033[34m%s\033[0m$", getlogin(), current_dir
```

得到进入minishell后可以获得当前路径以及文件名的提示



```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ ./minishell
*************welcome to mini shell******2013747******
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

5.3 read_command()

读入终端的命令,对终端的命令使用“; ”进行划分，对多条命令进行划分;

```
void read_command()
{
    //处理cmdline
    getline(cin, cmdline); // input string with ' '
    typedef string::size_type string_size;
    string_size i = 0;
    while (i != cmdline.size())
    {
        int flag = 0;
        while (i != cmdline.size() && flag == 0)
        {
            flag = 1;
            for (string_size x = 0; x < separator.size(); ++x)
                if (cmdline[i] == separator[x])
                {
                    ++i;
                    flag = 0;
                    break;
                }
        }
        flag = 0;
        string_size j = i;
        while (j != cmdline.size() && flag == 0)
        {
            for (string_size x = 0; x < separator.size(); ++x)
                if (cmdline[j] == separator[x])
                {
                    flag = 1;
                    break;
                }
            if (flag == 0)
                ++j;
        }
        if (i != j)
        {
            commands.push_back(cmdline.substr(i, j - i));
            i = j;
        }
    }
}
```

5.4 parse_command()

处理命令，对单条命令进行参数解析

```
void parse_command()
{
    //处理cmds
    for (int i = 0; i < commands.size(); i++)
    {
        child_commands.clear();
        command.clear();
        stringstream input2(commands[i]); // string stream initialize 不按照空格戈
        while (input2 >> command)
        {
            child_commands.push_back(command);
        }
        if (command == "exit")
        {
            printf("\033[32m****************** mini shell exit*******2013747****
            exit(0);
        }
        if (child_commands.size())
        {
            execute_command();
        }
    }
}
```

5.5 execute_command

这一步分成shell内部命令与shell的外部命令

5.5.1 内部命令

实现cd

```
void do_cd(){

        // 当前系统目录
        // char target_path[100];
        // getcwd(target_path, 100);
        // cout<<"now path"<<target_path<<endl;
        if (child_commands.size() == 1)
        {
            strcpy(current_dir, user_dir);
        }
        else
        {
            const char *rest = child_commands[1].c_str();
            if (child_commands[1] == "/")
            {
                opendir(rest);
                strcpy(current_dir, rest);
                chdir(rest);

            }
            else if (child_commands[1] == "..")
            {
                char *parent_dir = dirname(current_dir);
                strcpy(current_dir, parent_dir);
                chdir(parent_dir);
            }
            else if (child_commands[1] == "~")
            {
                strcpy(current_dir, user_dir);
                chdir(user_dir);
            }
            else
            {
                char target_path[1024];
                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");
                }
                strcpy(current_dir, target_path);
                chdir(current_dir);
            }
            cout << current_dir << endl;
        }
    }
```

实现ls以及ls的文件重定向

```
void do_ls(){


        pid_t pid;
        pid= fork();
        int status;
        int count = 0;
        const char *rest = child_commands[0].c_str();
        if (pid == 0)
        {
            for (int i = 0; i < child_commands.size(); i++)
            {

                if (child_commands[i] == ">")
                {
                    re_flag = 1;
                    count = i;
                }
                if (child_commands[i] == ">>")
                {
                    re_flag = 2;
                    count = i;
                }
            }
            if (re_flag != 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], child_commands[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 (re_flag == 1)
                    fd = open(child_commands[count + 1].c_str(), O_CREAT | O_WRO
                else if (re_flag == 2)
                    fd = open(child_commands[count + 1].c_str(), O_CREAT | O_WRO
                dup2(fd, 1);
                if (execvp(rest, cmd_temp) < 0)
                {
                    printf("\033[31m%s:command not found.\n\033[0m", child_comma
                }
            }
            else
            {
                char **cmd_temp = new char *[child_commands.size() + 1];
                for (int i = 0; i < child_commands.size(); i++)
                {
                    cmd_temp[i] = new char[500];
                    memset(cmd_temp[i], 0, sizeof(*cmd_temp[i]));
                }
                for (int i = 0; i < child_commands.size(); i++)
                {
                    strcpy(cmd_temp[i], child_commands[i].c_str());
                }
                cmd_temp[child_commands.size()] = current_dir;
                cmd_temp[child_commands.size() + 1] = NULL;
                if (execvp(rest, cmd_temp) < 0)
                {
                    printf("\033[31m%s:command not found.\n\033[0m", child_comma
                }
            }
        }

        else if (pid > 0)
        {
            do
            {
                waitpid(pid, &status, WUNTRACED);
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        }


}
```

5.5.2 外部命令

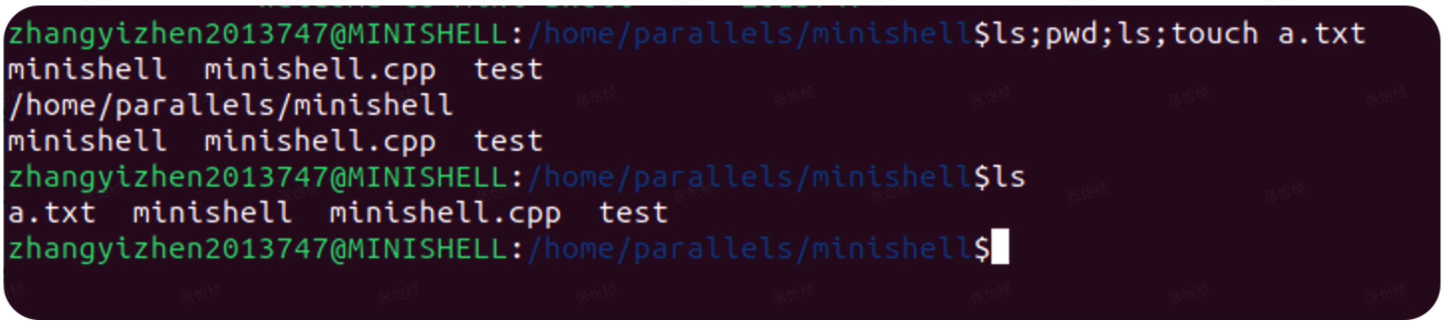
创建子进程进行外部命令的执行

```
pid_t pid;
pid = fork();
int status;
const char *rest = child_commands[0].c_str();
if (pid == 0)
{
    //子线程
    char **cmd_temp = new char *[child_commands.size()];
    for (int i = 0; i < child_commands.size(); i++)
    {
        cmd_temp[i] = new char[500];
        memset(cmd_temp[i], 0, sizeof(*cmd_temp[i]));
```

```
  }
  for (int i = 0; i < child_commands.size(); i++)
  {
    strcpy(cmd_temp[i], child_commands[i].c_str());
  }
  cmd_temp[child_commands.size()] = NULL;
  if (execvp(rest, cmd_temp) < 0)
  {
    printf("\033[31m%s:command not found.\n\033[0m", child_commands[0].c_str
  }

}
else if (pid > 0)
{
  do
  {
    waitpid(pid, &status, WUNTRACED);
  } while (!WIFEXITED(status) && !WIFSIGNALED(status));
}
```

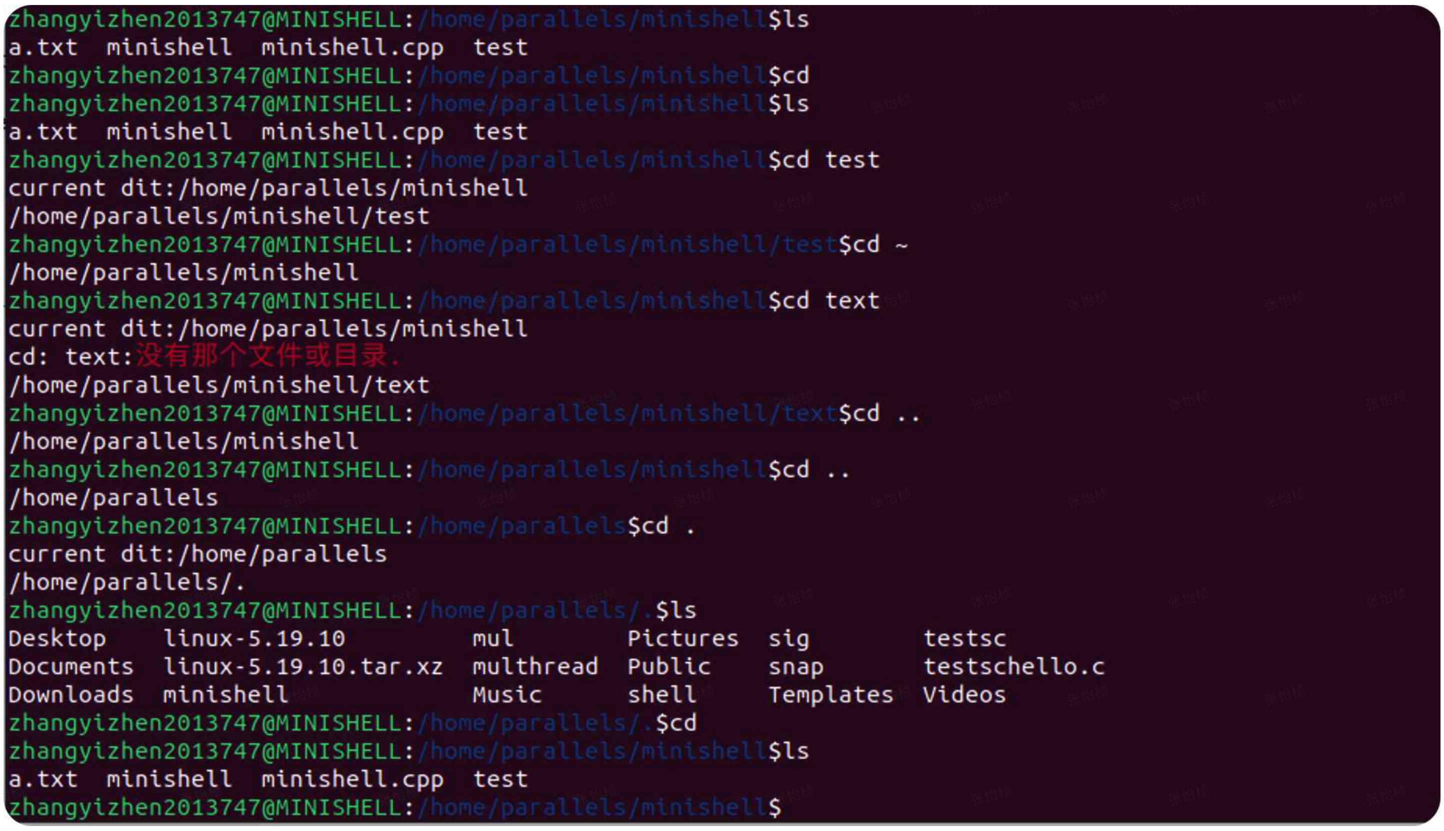
6. 实现功能

1. 支持用户输入一行命令及其多个参数，并解析执行，并输出结果；



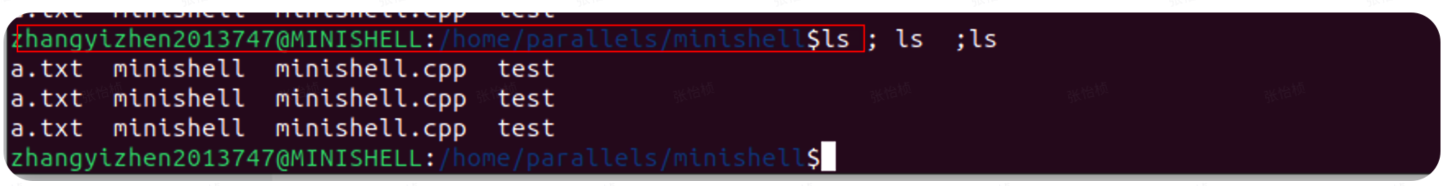
```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls;pwd;ls;touch a.txt
minishell  minishell.cpp  test
/home/parallels/minishell
minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

2. 支持 cd 命令，若无参数则回到当前用户的登录目录（见下面提示）；



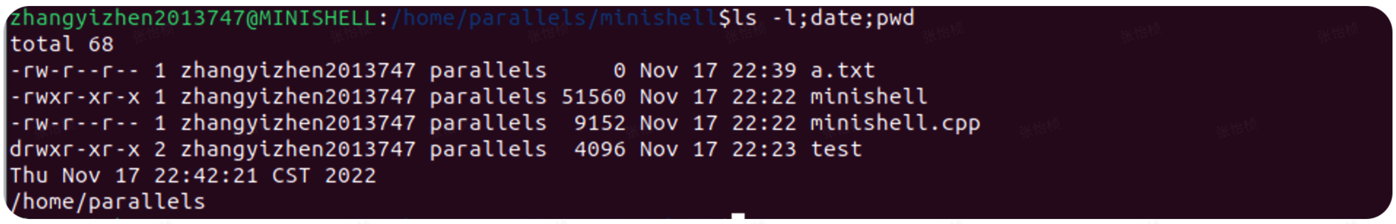
```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cd
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cd test
current dit:/home/parallels/minishell
/home/parallels/minishell/test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell/test$cd ~
/home/parallels/minishell
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cd text
current dit:/home/parallels/minishell
cd: text:没有那个文件或目录.
/home/parallels/minishell/text
zhangyizhen2013747@MINISHELL:/home/parallels/minishell/text$cd ..
/home/parallels/minishell
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cd ..
/home/parallels
zhangyizhen2013747@MINISHELL:/home/parallels$cd .
current dit:/home/parallels
/home/parallels/.
zhangyizhen2013747@MINISHELL:/home/parallels/.$ls
Desktop    linux-5.19.10         mul        Pictures  sig        testsc
Documents  linux-5.19.10.tar.xz  multhread  Public    snap       testschello.c
Downloads  minishell             Music      shell     Templates  Videos
zhangyizhen2013747@MINISHELL:/home/parallels/.$cd
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

3. 支持以“当前路径”和“用户名”为提示符；支持对命令行中空格的自动忽略处理；
4. 支持对命令行中 tab 键的自动忽略处理；



```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls ; ls  ;ls
a.txt  minishell  minishell.cpp  test
a.txt  minishell  minishell.cpp  test
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

5. 支持一行中以“；”（为标志）分隔的多个命令及多个参数的顺序执行，即如下：
   a. MINI SHELL#pwd; ls –l;date



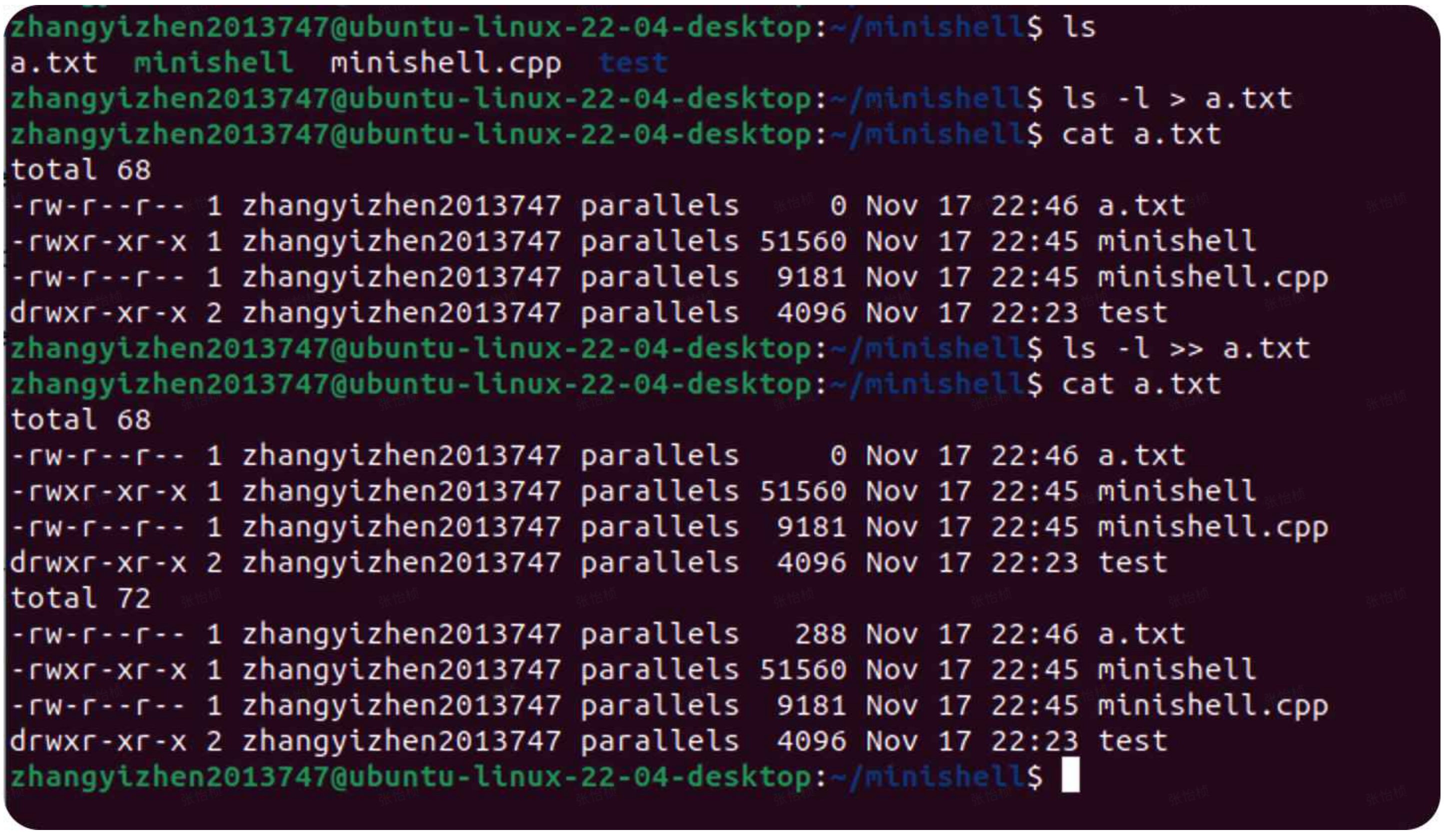
```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls -l;date;pwd
total 68
-rw-r--r-- 1 zhangyizhen2013747 parallels     0 Nov 17 22:39 a.txt
-rwxr-xr-x 1 zhangyizhen2013747 parallels 51560 Nov 17 22:22 minishell
-rw-r--r-- 1 zhangyizhen2013747 parallels  9152 Nov 17 22:22 minishell.cpp
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:23 test
Thu Nov 17 22:42:21 CST 2022
/home/parallels
```

   b. 说明：上述三个命令须在本 Mini Shell 下依次顺序执行，最后由 Mini Shell 再次循环接受用户的新命令。

附加功能：

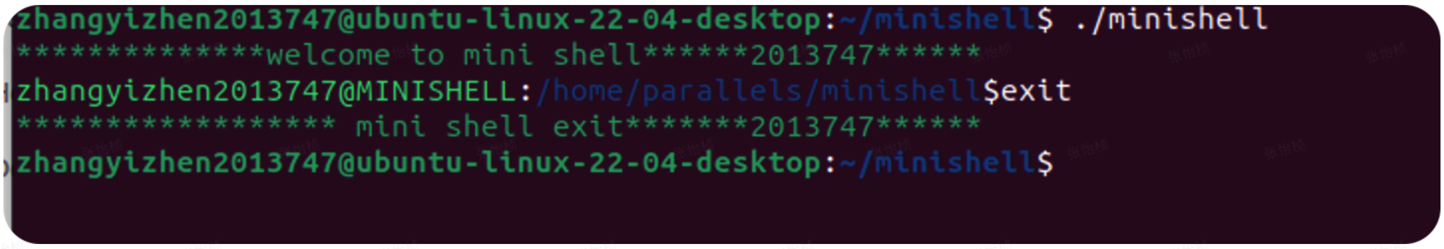
ls -l 文件重定向

cat查看文件



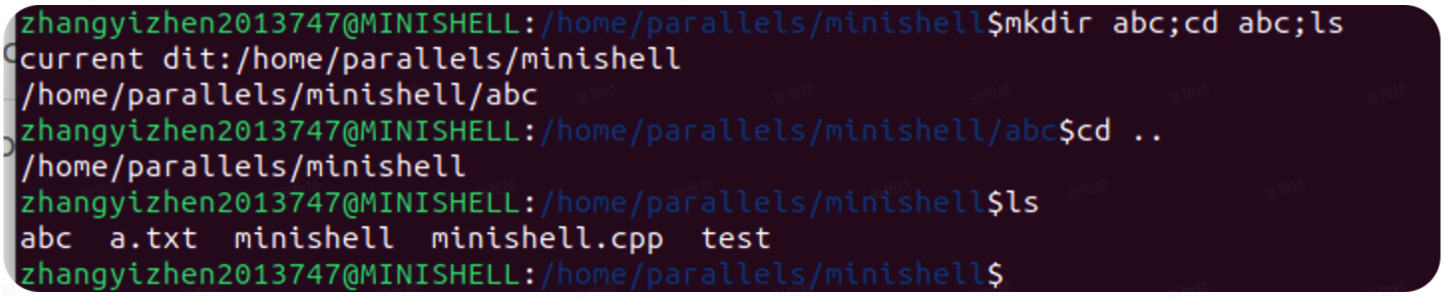
```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ ls
a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ ls -l > a.txt
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ cat a.txt
total 68
-rw-r--r-- 1 zhangyizhen2013747 parallels     0 Nov 17 22:46 a.txt
-rwxr-xr-x 1 zhangyizhen2013747 parallels 51560 Nov 17 22:45 minishell
-rw-r--r-- 1 zhangyizhen2013747 parallels  9181 Nov 17 22:45 minishell.cpp
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:23 test
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ ls -l >> a.txt
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ cat a.txt
total 68
-rw-r--r-- 1 zhangyizhen2013747 parallels     0 Nov 17 22:46 a.txt
-rwxr-xr-x 1 zhangyizhen2013747 parallels 51560 Nov 17 22:45 minishell
-rw-r--r-- 1 zhangyizhen2013747 parallels  9181 Nov 17 22:45 minishell.cpp
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:23 test
total 72
-rw-r--r-- 1 zhangyizhen2013747 parallels   288 Nov 17 22:46 a.txt
-rwxr-xr-x 1 zhangyizhen2013747 parallels 51560 Nov 17 22:45 minishell
-rw-r--r-- 1 zhangyizhen2013747 parallels  9181 Nov 17 22:45 minishell.cpp
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:23 test
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ 
```

exit()



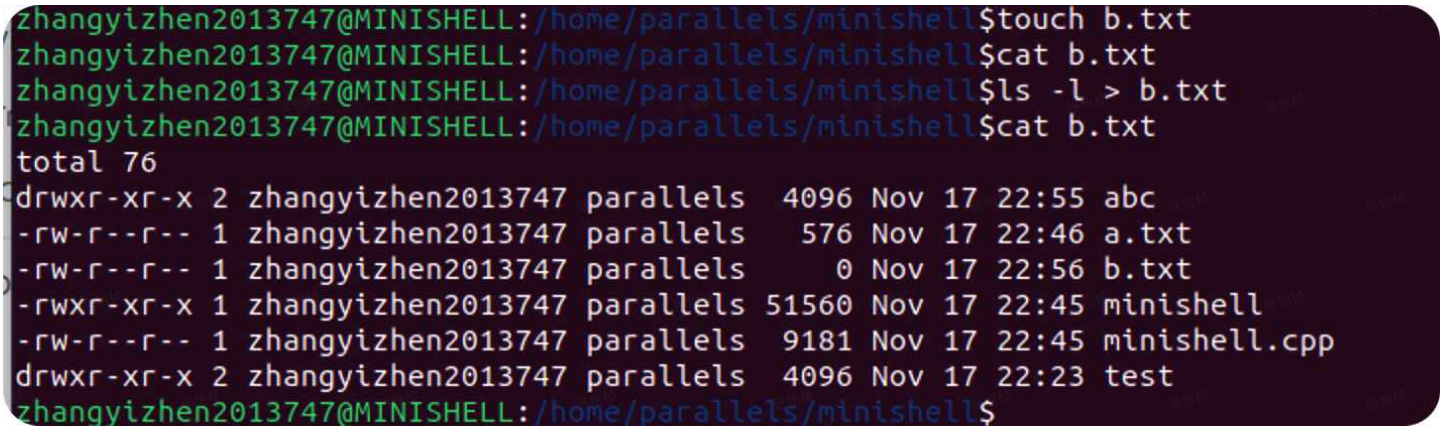
```
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$ ./minishell
*************welcome to mini shell******2013747******
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$exit
***************** mini shell exit*******2013747******
zhangyizhen2013747@ubuntu-linux-22-04-desktop:~/minishell$
```

Mkdir 文件夹



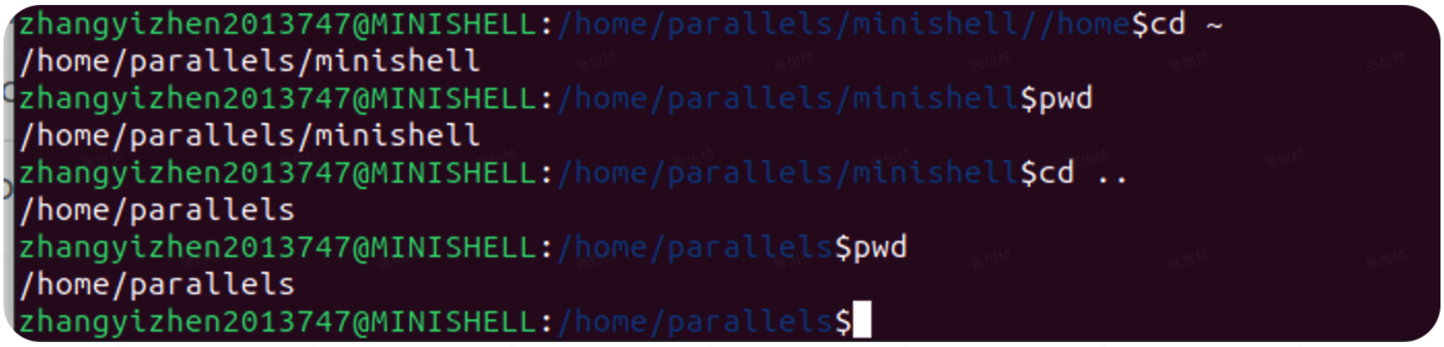
```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$mkdir abc;cd abc;ls
current dit:/home/parallels/minishell
/home/parallels/minishell/abc
zhangyizhen2013747@MINISHELL:/home/parallels/minishell/abc$cd ..
/home/parallels/minishell
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
abc  a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

Touch 文件



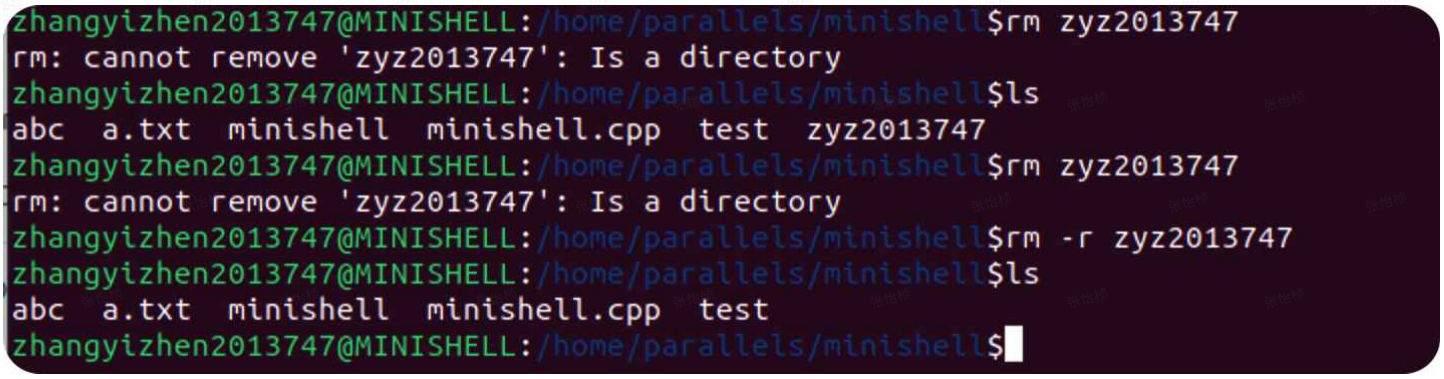
```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$touch b.txt
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cat b.txt
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls -l > b.txt
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cat b.txt
total 76
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:55 abc
-rw-r--r-- 1 zhangyizhen2013747 parallels   576 Nov 17 22:46 a.txt
-rw-r--r-- 1 zhangyizhen2013747 parallels     0 Nov 17 22:56 b.txt
-rwxr-xr-x 1 zhangyizhen2013747 parallels 51560 Nov 17 22:45 minishell
-rw-r--r-- 1 zhangyizhen2013747 parallels  9181 Nov 17 22:45 minishell.cpp
drwxr-xr-x 2 zhangyizhen2013747 parallels  4096 Nov 17 22:23 test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

pwd



```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell//home$cd ~
/home/parallels/minishell
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$pwd
/home/parallels/minishell
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$cd ..
/home/parallels
zhangyizhen2013747@MINISHELL:/home/parallels$pwd
/home/parallels
zhangyizhen2013747@MINISHELL:/home/parallels$
```

rm



```
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$rm zyz2013747
rm: cannot remove 'zyz2013747': Is a directory
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
abc  a.txt  minishell  minishell.cpp  test  zyz2013747
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$rm zyz2013747
rm: cannot remove 'zyz2013747': Is a directory
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$rm -r zyz2013747
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$ls
abc  a.txt  minishell  minishell.cpp  test
zhangyizhen2013747@MINISHELL:/home/parallels/minishell$
```

7. 参考资料

老师的github实验文档

8. 附件

```cpp
#include <iostream>
#include <cstdio>
#include <string.h>
```

```
#include <unistd.h>
#include <libgen.h>
#include <sys/wait.h>
#include <vector>
#include <sstream>
#include <dirent.h>
#include <fcntl.h>
using namespace std;

//全局常量定义
#define SUCCESS 0

//全局变量定义

int re_flag = 0;
char current_dir[100];
char user_dir[100];

string cmdline; //command input
string separator=";";
string command;


vector<string> commands; // 所有命令
vector<string> child_commands; //仅一个命令

typedef void (CMD_HANDLER)(void);
typedef struct builtin_cmd
{
    char* name;
    CMD_HANDLER *handler;
} BUILTIN_CMD;

//全局函数声明
void init();
int execute_command();
void read_command();
void parse_command();
int builtin(void);
void do_cd();
void do_ls();

//内部命令解析
BUILTIN_CMD builtins[] = {
        {(char*)"cd", do_cd},
        {(char*)"ls", do_ls},
        {NULL, NULL}
```

```

};



int main()
{

    printf("\033[32m*************welcome to mini shell******2013747****** \n\03
    strcpy(current_dir, getcwd(NULL, 0));
    strcpy(user_dir, getcwd(NULL, 0));
//    printf("PATH : %s\n", getenv("PATH"));
    while (1)
    {
        printf("\033[92m%s@MINISHELL\033[0m:\033[34m%s\033[0m$", getlogin(), cur
        init();
        read_command();
        parse_command();

    }
    return 0;
}

void init(){
    for (int i = 0; i < child_commands.size(); i++)
    {
        child_commands[i].clear();
    }
    for (int i = 0; i < commands.size(); i++)
    {
        commands[i].clear();
    }
    cmdline.clear();
    command.clear();
    child_commands.clear();
    commands.clear();
}

void read_command()
{
    //处理cmdline
    getline(cin, cmdline); // input string with ' '
    typedef string::size_type string_size;
    string_size i = 0;
    while (i != cmdline.size())
    {
        int flag = 0;
        while (i != cmdline.size() && flag == 0)
        {
            flag = 1;
            for (string_size x = 0; x < separator.size(); ++x)
                if (cmdline[i] == separator[x])
                {
                    ++i;
                    flag = 0;
                    break;
                }
        }
        flag = 0;
        string_size j = i;
        while (j != cmdline.size() && flag == 0)
        {
            for (string_size x = 0; x < separator.size(); ++x)
                if (cmdline[j] == separator[x])
                {
                    flag = 1;
                    break;
                }
            if (flag == 0)
                ++j;
        }
        if (i != j)
        {
            commands.push_back(cmdline.substr(i, j - i));
            i = j;
        }
    }
}

void parse_command()
{
    //处理cmds
    for (int i = 0; i < commands.size(); i++)
    {
        child_commands.clear();
        command.clear();
      du  stringstream input2(commands[i]); // string stream initialize 不按照空ᡀ
        while (input2 >> command)
        {
            child_commands.push_back(command);
        }
        if (command == "exit")
        {
            printf("\033[32m***************** mini shell exit*******2013747****
            exit(0);
        }
        if (child_commands.size())
        {
            execute_command();
        }
    }
}

//内部命令解析
//执行返回1，没有表示0
int builtin(void){
    int i = 0;
    int found = 0;
    while(builtins[i].name != NULL){
        if(builtins[i].name == child_commands[0]) {
            builtins[i].handler();
            found=1;
            break;
        }
        i++;
    }
    return found;
}

int execute_command()
{

    if(builtin())
        return SUCCESS;
    else
    {
        pid_t pid;
        pid = fork();
        int status;
        const char *rest = child_commands[0].c_str();
        if (pid == 0)
        {
            //子线程
            char **cmd_temp = new char *[child_commands.size()];
            for (int i = 0; i < child_commands.size(); i++)
            {
                cmd_temp[i] = new char[500];
                memset(cmd_temp[i], 0, sizeof(*cmd_temp[i]));
            }
            for (int i = 0; i < child_commands.size(); i++)
            {
                strcpy(cmd_temp[i], child_commands[i].c_str());
            }
            cmd_temp[child_commands.size()] = NULL;
            if (execvp(rest, cmd_temp) < 0)
            {
                printf("\033[31m%s:command not found.\n\033[0m", child_commands[

        }
        else if (pid > 0)
        {
            do
            {
                waitpid(pid, &status, WUNTRACED);
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        }
    }
    return SUCCESS;
}

void do_cd(){

        // 当前系统目录
        // char target_path[100];
        // getcwd(target_path, 100);
        // cout<<"now path"<<target_path<<endl;
        if (child_commands.size() == 1)
        {
            strcpy(current_dir, user_dir);
            chdir(user_dir);
        }
        else
        {
            const char *rest = child_commands[1].c_str();
            if (child_commands[1] == "/")
            {
                opendir(rest);
                strcpy(current_dir, rest);
                chdir(rest);

            }
            else if (child_commands[1] == "..")
            {
                char *parent_dir = dirname(current_dir);
                strcpy(current_dir, parent_dir);
                chdir(parent_dir);
            }
            else if (child_commands[1] == "~")
            {
                strcpy(current_dir, user_dir);
                chdir(user_dir);
            }
            else
            {
                char target_path[1024];
                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");
                }
                strcpy(current_dir, target_path);
                chdir(current_dir);
            }
            cout << current_dir << endl;
        }
    }

void do_ls(){


        pid_t pid;
        pid= fork();
        int status;
        int count = 0;
        const char *rest = child_commands[0].c_str();
        if (pid == 0)
        {
            for (int i = 0; i < child_commands.size(); i++)
            {

                if (child_commands[i] == ">")
                {
                    re_flag = 1;
                    count = i;
                }
                if (child_commands[i] == ">>")
                {
                    re_flag = 2;
                    count = i;
                }
            }
            if (re_flag != 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], child_commands[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 (re_flag == 1)
                    fd = open(child_commands[count + 1].c_str(), O_CREAT | O_WRO
                else if (re_flag == 2)
                    fd = open(child_commands[count + 1].c_str(), O_CREAT | O_WRO
                dup2(fd, 1);
                if (execvp(rest, cmd_temp) < 0)
                {
                    printf("\033[31m%s:command not found.\n\033[0m", child_comma
                }
            }
            else
            {
                char **cmd_temp = new char *[child_commands.size() + 1];
                for (int i = 0; i < child_commands.size(); i++)
                {
                    cmd_temp[i] = new char[500];
                    memset(cmd_temp[i], 0, sizeof(*cmd_temp[i]));
                }
                for (int i = 0; i < child_commands.size(); i++)
                {
                    strcpy(cmd_temp[i], child_commands[i].c_str());
                }
                cmd_temp[child_commands.size()] = current_dir;
                cmd_temp[child_commands.size() + 1] = NULL;
                if (execvp(rest, cmd_temp) < 0)
                {
                    printf("\033[31m%s:command not found.\n\033[0m", child_comma
                }
            }
        }

        else if (pid > 0)
        {
            do
            {
                waitpid(pid, &status, WUNTRACED);
            } while (!WIFEXITED(status) && !WIFSIGNALED(status));
        }


}
```