《操作系统》课第九次实验报告

学院:	软件学院
姓名:	杨万里
学号:	2013774
邮箱:	2013774@mail.nankai.edu.cn
时间:	2022/11/13

0. 开篇感言

在中国，我们有谚语“莫看江面平如镜，要看水底万丈深”；

在国外，名人说“尘世间的事物，真相与表象总是有很大的区别。只有一小部分人能透过现象看到本质，绝大多数人还停留在表象上。”

shell 命令解释环境如此的便利、快捷，但是看似简单的命令行之下是严谨而复杂的解释系统。

本次实验通过实现 Mini Shell 程序，让我感受到其中美妙的逻辑。

1. 实验题目

C 语言实现 Mini Shell 命令解释环境

2. 实验目标

（1） 在 Linux 平台完成实验

（2） 编写 C 程序

（3） 实现 Mini Shell 命令解释环境

（4） 该环境循环接收用户输入的命令以及若干参数，对命令解析执行，并输出执行结果

具体要求：

- 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；
- 支持 cd 命令，若无参数则回到当前用户的登录目录；
- 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；
- 支持对命令行中 tab 键的自动忽略处理；
- 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，命令须在 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。

3. 原理方法

（实验源代码附在实验报告结尾处）

（1） 提前设定内部命令表，包括 cd、pwd、help、exit、echo、ls

（2） 获取当前系统的用户名称，为后续提示信息做准备

```
//用户名
struct passwd *pwd;
pwd = getpwuid(getuid());
char* user_name = pwd->pw_name;
```

（3） 建立基本的 Mini Shell 程序框架：程序启动，开始 shell_loop 函数，该函数内部是一个循环，循环结束的标识 flag。循环的每一次，读入一行用户

命令，并对该指令解析执行，执行返回参数 flag 表示 shell 程序是否继续。

（用户名只需要获取一次，所在路径需要实时获取）

```
void shell_loop(){
    //首先需要获取当前路径和用户名
    //用户名
    struct passwd *pwd;
    pwd = getpwuid(getuid());
    char* user_name = pwd->pw_name;
    //路径
    char* path = malloc(sizeof(char) * BUFSIZE);
    char* line = malloc(sizeof(char) * BUFSIZE);  //接
    int flag = 1;  //循环是否终止的标识
    while(flag == 1){
        memset(line,'\0',(sizeof(char) * BUFSIZE));
        getcwd(path, BUFSIZE);
        printf("%s:%s$ ", user_name, path);
        line = shell_readline();
        //读入命令之后，需要执行
        flag = execute_line(line);
    }
    free(line);
}
```

(4) 编写读入用户输入的函数 shell_readline，主要就是逐个读入字符，遇到换行符结束读入，并且把换行符写为'\0'

(5) 执行命令的函数 execute_line 包括两个步骤：

1）划分输入的 line，包括识别其中有几条命令（对于“；”的处理），并且把每个命令的单词分开。

```
int execute_line(char* line){
    //先识别有几个封号
    char** cmds = malloc(sizeof(char*) * 8);
    for (int i = 0; i < 8; i++)
    {
        cmds[i] = malloc(sizeof(char) * 128);
    }
    int i = 0, j = 0, k=0;
    while(line[i] != '\0'){
        if(line[i] == ';'){
            cmds[j][k] = '\0';
            i++;
            j++;
            k = 0;
        }else{
            if(line[i] == '\t'){
                line[i] = ' ';
            }
            cmds[j][k] = line[i];
            i++;
            k++;
        }
    }
    cmds[j][k] = '\0';
    int cmd_num = j + 1;
    int flag;
    for (int i = 0; i < cmd_num; i++)
```

上图把指令分开，顺序执行多条指令或一条指令。顺便把输入当中的 tab 换成了空格“ ”，以便后续忽略的处理。

```
for (int i = 0; i < cmd_num; i++)
{
    char** tokens = split_line(cmds[i]);
    flag = execute(tokens);
    if(flag == 0){
        break;
    }
}
```

对于单条指令，先划分为一系列 tokens，再进行命令执行。

划分函数依据空格“ ”进行划分，因此实现了忽略空格的要求，由于此前已经把 tab 换成了空格，因此也实现了 tab 的忽略。具体内容如下图所示。

```
//输入的命令需要切分
char** split_line(char* line){
    int i = 0;
    char** tokens = malloc(sizeof(char*) * TOKENNUM);
    char* token;
    //分配失败
    if(!tokens){
        printf("allocation failed!\n");
        exit(1);
    }
    token = strtok(line, " ");
    while (token != NULL)
    {
        tokens[i] = token;
        i++;
        //理论上允许用户输入无限个单词，所以当输入单词超过预定的64个之
        //但是本次实验就假设输入单词个数在64之内
        token = strtok(NULL, " ");  //继续用之前的line
    }
    tokens[i] = NULL;  //终结符号
    return tokens;
}
```

2）得到 tokens 序列之后，就可以开始解析执行了

```
int execute(char** tokens){
    if(tokens[0] == NULL){
        //没有命令
        return 1;
    }
    //尝试逐个匹配内部命令
    for(int i=0;i<inner_commands_num;i++){
        if(strcmp(tokens[0], inner_commands[i]) == 0){
            //执行对应的内部命令
            return (*funcs[i])(tokens);
        }
    }
    //如果没有匹配的内部命令，则作为外部命令处理
    return outter_commands(tokens);
}
```

首先逐个匹配内部命令，如果匹配到，则执行相应的函数

如果都没有匹配到，则作为外部命令处理，调用外部命令处理函数

（6） 逐个实现内部命令对应的函数：

cd 命令：注意 cd 无后续路径的情况处理，需要回到登录目录下

```
int func_cd(char** tokens){
    //正常来说，cd只有两个串
    char* buffer = malloc(sizeof(char) * BUFSIZE);
    if(tokens[1] == NULL){
        //一个串返回登录目录
        getuserdir(buffer);
        tokens[1] = buffer;
    }
    //系统调用chdir，执行成功返回0
    if(chdir(tokens[1]) != 0){
        perror("myshell");
    }
    free(buffer);
    return 1;
}
```

pwd 命令：获取当前路径

```
int func_pwd(char** tokens){
    char* buffer = malloc(sizeof(char) * BUFSIZE);  //接收路径
    getcwd(buffer, BUFSIZE);
    printf("current path is : %s\n", buffer);
    free(buffer);
    return 1;
}
```

help 命令：获取内部命令集合

```
//help用于调用内部命令的集合
int func_help(char** tokens){
    printf("--------------- YWL's Mini Shell -----------
    printf("These shell commands are defined internally.
    for (int i = 0; i < inner_commands_num; i++)
    {
        printf("%s\n", inner_commands[i]);
    }
    return 1;
}
```

exit 命令：只需要把返回值置零即可

```
int func_exit(char** tokens){
    printf("--------------- Goodbye! ---------------\n");
    return 0;
}
```

echo 命令：将用户输入输出

```
int func_echo(char** tokens){
    int i = 1;
    while(tokens[i] != NULL){
        printf("%s ", tokens[i]);
        i++;
    }
    printf("\n");
    return 1;
}
```

ls 命令：列出当前路径下的文件及目录

```
int func_ls(char** tokens){
    //首先获得当前的路径，用getcwd
    char* path = malloc(sizeof(char) * BUFSIZE);  //接收路径
    getcwd(path, BUFSIZE);
    DIR* dir;
    struct dirent * ptr;
    if((dir=opendir(path)) == NULL){
      perror("Open dir error");
      exit(1);
    }
    while((ptr=readdir(dir)) != NULL){
        if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"..")
            continue;
        printf("%s\n",ptr->d_name);
    }
    closedir(dir);
    free(path);
    return 1;
}
```

（7） 实现外部命令处理函数：

```
int outter_commands(char** tokens){
    int pid = fork();
    int status;
    if(pid < 0){
        fprintf(stderr, "Fork Failed\n");
    }
    else if (pid == 0){
        if (execvp(tokens[0], tokens) == -1){
            perror("myshell ");
            exit(1);
        }
    }
    else {
        while(1){
            pid = wait(&status);
            if(pid == -1){
                break;
            }
            else{
                WEXITSTATUS (status);
            }
        }
    }
    return 1;
}
```

创建子进程，传入用户输入的参数列表，执行目标程序，shell 程序主进程等待子进程执行

以上则完成了整个 Mini Shell 程序的编写。

4. 具体步骤

（1） 按照上一小节的原理方法编写程序，编译生成可执行文件，执行程序

（2） 开始验证 Mini Shell 功能：

支持内部命令（cd、pwd、ls、echo、help、exit）和外部命令

1）进入 Mini Shell 程序，每次输入前提示用户名和所在路径

```
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab9_code$ ./shell
************ Welcome to YWL's Mini Shell! ************
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$
```

2）cd 命令的验证

包括无目标路径则进入登录目录

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$ cd ..
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ cd lab9_code
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$ cd
wanliyang2013774:/home/wanliyang2013774$
```

3）ls 命令的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ ls
lab7_code
lab6_code
suiyi
CopyDirMulProcess
test.c
lab9_code
OsLab2
lab8_code
```

4）pwd 命令的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ pwd
current path is : /home/wanliyang2013774/OSWorkSpace
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$
```

5）echo 命令的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ echo hello this is a test
hello this is a test
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$
```

6）help 命令的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ help
-------------- YWL's Mini Shell --------------
These shell commands are defined internally.  Type `help' to see this list.
cd
pwd
help
exit
echo
ls
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$
```

7）exit 命令的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$ exit
-------------- Goodbye! --------------
************ YWL's Mini Shell Exit! ************
wanliyang2013774@wanliyang2013774-virtual-machine:~/OSWorkSpace/lab9_code$
```

8）外部命令的验证

使用测试程序 test.c

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$ ./test this is a test program
这是测试程序
这是运行程序的路径: ./test
这是第1个参数: this
这是第2个参数: is
这是第3个参数: a
这是第4个参数: test
这是第5个参数: program
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$
```

9）验证对空格和 Tab 的忽略

空格：

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$  pwd
current path is : /home/wanliyang2013774/OSWorkSpace/lab9_code
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$
```

Tab：

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$          pwd
current path is : /home/wanliyang2013774/OSWorkSpace/lab9_code
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$
```

10）对“;”的验证

```
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace/lab9_code$ pwd;ls;cd ..
current path is : /home/wanliyang2013774/OSWorkSpace/lab9_code
test.c
test
shell
myshell.c
wanliyang2013774:/home/wanliyang2013774/OSWorkSpace$
```

5. 总结心得

（1） shell 解释环境可以说是很多初学者对于 Linux 系统的初印象了，刚开始接触 Linux 系统，觉得很多操作都需要在 shell 环境中进行，感到非常不适

应。但是熟练掌握之后，感到 shell 环境非常便利和好用。本次实验通过自己编写简单的 shell 程序，更为深入地理解了其背后运作的原理与奥妙。

（2） 由于操作系统封装了很多函数，诸如 getcwd、chdir 等，所以实现内部命令并不困难，并不需要自己造轮子。

（3） 其实之前一直不理解，为什么程序接收的命令行参数的第一个参数是程序路径。在本次实验编写外部命令处理函数的时候体会到了这种设定的便利。在这种设定下，传入参数时，只需要直接把命令行包括可执行程序在内的所有参数一同传入即可，而不需要去掉可执行程序路径。

（4） 再次加深我对多进程的理解和掌握。

6. 参考资料

实验指导书《os_lab_minishell.pdf》

7. 实验源代码

Mini Shell 源程序:

```
1.  #include <stdio.h>
2.  #include <string.h>
3.  #include <unistd.h>
4.  #include <pwd.h>
5.  #include <stdlib.h>
6.  #include <sys/wait.h>
7.  #include <dirent.h>
8.
9.  #define BUFSIZE 1024  //输入缓冲区大小
10. #define TOKENNUM 64  //默认最多的输入单词数量
11.
12. //内部命令表
13. char* inner_commands[] = {"cd", "pwd", "help", "exit", "echo", "ls"};

14. int inner_commands_num = 6;
```

```
15.
16. int getuserdir(char *aoUserDir)
17. {
18.     char *LoginId;
19.     struct passwd *pwdinfo;
20.     if (aoUserDir == NULL)
21.         return -9;
22.     if ((LoginId = getlogin ()) == NULL) {
23.         perror ("getlogin");
24.         aoUserDir[0] = '\0';
25.         return -8;
26.     }
27.     if ((pwdinfo = getpwnam (LoginId)) == NULL) {
28.         perror ("getpwnam");
29.         return -7;
30.     }
31.     strcpy (aoUserDir, pwdinfo->pw_dir);
32. }
33.
34. int func_cd(char** tokens){
35.     //正常来说，cd 只有两个串
36.     char* buffer = malloc(sizeof(char) * BUFSIZE);
37.     if(tokens[1] == NULL){
38.         //一个串返回登录目录
39.         getuserdir(buffer);
40.         tokens[1] = buffer;
41.     }
42.     //系统调用 chdir，执行成功返回 0
43.     if(chdir(tokens[1]) != 0){
44.         perror("myshell");
45.     }
46.     free(buffer);
47.     return 1;
48. }
49.
50. int func_pwd(char** tokens){
51.     char* buffer = malloc(sizeof(char) * BUFSIZE);  //接收路径
52.     getcwd(buffer, BUFSIZE);
53.     printf("current path is : %s\n", buffer);
54.     free(buffer);
55.     return 1;
56. }
57.
58. //help 用于调用内部命令的集合
```

```
59. int func_help(char** tokens){
60.     printf("--------------- YWL's Mini Shell ---------------\n");
61.     printf("These shell commands are defined internally.  Type `help'
    to see this list.\n");
62.     for (int i = 0; i < inner_commands_num; i++)
63.     {
64.         printf("%s\n", inner_commands[i]);
65.     }
66.     return 1;
67. }
68. 
69. int func_exit(char** tokens){
70.     printf("--------------- Goodbye! ---------------\n");
71.     return 0;
72. }
73. 
74. int func_echo(char** tokens){
75.     int i = 1;
76.     while(tokens[i] != NULL){
77.         printf("%s ", tokens[i]);
78.         i++;
79.     }
80.     printf("\n");
81.     return 1;
82. }
83. 
84. int func_ls(char** tokens){
85.     //首先获得当前的路径，用 getcwd
86.     char* path = malloc(sizeof(char) * BUFSIZE);  //接收路径
87.     getcwd(path, BUFSIZE);
88.     DIR* dir;
89.     struct dirent * ptr;
90.     if((dir=opendir(path)) == NULL){
91.       perror("Open dir error");
92.       exit(1);
93.     }
94.     while((ptr=readdir(dir)) != NULL){
95.         if(strcmp(ptr->d_name,".")==0 || strcmp(ptr->d_name,"..")==0)
    ///current dir OR parrent dir
96.             continue;
97.         printf("%s\n",ptr->d_name);
98.     }
99.     closedir(dir);
100.       free(path);
```

```
101.        return 1;
102.    }
103.
104.    //函数指针数组
105.    int (*funcs[])(char**) = {&func_cd, &func_pwd, &func_help, &func_e
    xit, &func_echo, &func_ls};
106.
107.    //读入整行命令
108.    char* shell_readline(){
109.        int i = 0;
110.        char* buffer = malloc(sizeof(char) * BUFSIZE);
111.        int c;  //接收字符 ACSII 码值
112.        if(!buffer){
113.            printf("allocation failed!\n");
114.            exit(1);
115.        }
116.        while(1){
117.            c = getchar();
118.            if(c == EOF || c == '\n'){
119.                buffer[i] = '\0';
120.                return buffer;  //遇到错误和换行都结束
121.            }
122.            else{
123.                buffer[i] = c;
124.                i++;
125.            }
126.        }
127.    }
128.
129.    //输入的命令需要切分
130.    char** split_line(char* line){
131.        int i = 0;
132.        char** tokens = malloc(sizeof(char*) * TOKENNUM);
133.        char* token;
134.        //分配失败
135.        if(!tokens){
136.            printf("allocation failed!\n");
137.            exit(1);
138.        }
139.        token = strtok(line, " ");
140.        while (token != NULL)
141.        {
142.            tokens[i] = token;
143.            i++;
```

```
144.            //理论上允许用户输入无限个单词，所以当输入单词超过预定的 64 个之后，应该重新分配
145.            //但是本次实验就假设输入单词个数在 64 之内
146.            token = strtok(NULL, " ");  //继续用之前的 line
147.        }
148.        tokens[i] = NULL;  //终结符号
149.        return tokens;
150.    }
151.
152.    int outter_commands(char** tokens){
153.        int pid = fork();
154.        int status;
155.        if(pid < 0){
156.            fprintf(stderr, "Fork Failed\n");
157.        }
158.        else if (pid == 0){
159.            if (execvp(tokens[0], tokens) == -1){
160.                perror("myshell ");
161.                exit(1);
162.            }
163.        }
164.        else {
165.            while(1){
166.                pid = wait(&status);
167.                if(pid == -1){
168.                    break;
169.                }
170.                else{
171.                    WEXITSTATUS (status);
172.                }
173.            }
174.        }
175.        return 1;
176.    }
177.
178.    int execute(char** tokens){
179.        if(tokens[0] == NULL){
180.            //没有命令
181.            return 1;
182.        }
183.        //尝试逐个匹配内部命令
184.        for(int i=0;i<inner_commands_num;i++){
185.            if(strcmp(tokens[0], inner_commands[i]) == 0){
186.                //执行对应的内部命令
```

```
187.                return (*funcs[i])(tokens);
188.            }
189.        }
190.        //如果没有匹配的内部命令，则作为外部命令处理
191.        return outter_commands(tokens);
192.    }
193.
194.    int execute_line(char* line){
195.        //先识别有几个封号
196.        char** cmds = malloc(sizeof(char*) * 8);
197.        for (int i = 0; i < 8; i++)
198.        {
199.            cmds[i] = malloc(sizeof(char) * 128);
200.        }
201.        int i = 0, j = 0, k=0;
202.        while(line[i] != '\0'){
203.            if(line[i] == ';'){
204.                cmds[j][k] = '\0';
205.                i++;
206.                j++;
207.                k = 0;
208.            }else{
209.                if(line[i] == '\t'){
210.                    line[i] = ' ';
211.                }
212.                cmds[j][k] = line[i];
213.                i++;
214.                k++;
215.            }
216.        }
217.        cmds[j][k] = '\0';
218.        int cmd_num = j + 1;
219.        int flag;
220.        for (int i = 0; i < cmd_num; i++)
221.        {
222.            char** tokens = split_line(cmds[i]);
223.            flag = execute(tokens);
224.            if(flag == 0){
225.                break;
226.            }
227.        }
228.        for (int i = 0; i < 8; i++)
229.        {
230.            free(cmds[i]);
```

```
231.        }
232.        free(cmds);
233.        return flag;
234.    }
235.
236.    void shell_loop(){
237.        //首先需要获取当前路径和用户名
238.        //用户名
239.        struct passwd *pwd;
240.        pwd = getpwuid(getuid());
241.        char* user_name = pwd->pw_name;
242.        //路径
243.        char* path = malloc(sizeof(char) * BUFSIZE);
244.        char* line = malloc(sizeof(char) * BUFSIZE);  //接收的字符串长度
    在 1024 以内
245.        int flag = 1;  //循环是否终止的标识
246.        while(flag == 1){
247.            memset(line,'\0',(sizeof(char) * BUFSIZE));
248.            getcwd(path, BUFSIZE);
249.            printf("%s:%s$ ", user_name, path);
250.            line = shell_readline();
251.            //读入命令之后，需要执行
252.            flag = execute_line(line);
253.        }
254.        free(line);
255.    }
256.
257.    int main(){
258.        printf("************ Welcome to YWL's Mini Shell! ************
    \n");
259.        shell_loop();  //开启 shell 循环
260.        printf("************ YWL's Mini Shell Exit! ************\n");

261.        return 0;
262.    }
```

测试程序 test.c:

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.
4.  int main(int argc, char* argv[]){
5.      printf("这是测试程序\n");
6.      printf("这是运行程序的路径: %s\n", argv[0]);
7.      for (int i = 1; i < argc; i++)
```

```
8.      {
9.          printf("这是第%d 个参数: %s\n", i, argv[i]);
10.     }
11.     return 0;
12. }
```