操作系统第十二次作业

实验目的

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 BashShell 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。

实验代码及解释

具体代码如下：

```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pwd.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <grp.h>
#include <time.h>
#define BUFSIZE 1024 //输入缓冲区大小
#define TOKENNUM 64 //默认最多的输入单词数量

char* commands[] = {"cd", "pwd", "exit", "echo", "ls"};
int num = 5;

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) {
```

```
31          perror ("getpwnam");
32          return -7;
33      }
34      strcpy (aoUserDir, pwdinfo->pw_dir);
35  }
36
37  int func_cd(char** tokens){
38      char* buffer = malloc(sizeof(char) * BUFSIZE);
39      if(tokens[1] == NULL){
40          getuserdir(buffer);
41          tokens[1] = buffer;
42      }
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  int func_exit(char** tokens){
59      return 0;
60  }
61
62  int func_echo(char** tokens){
63      int i = 1;
64      while(tokens[i] != NULL){
65          printf("%s ", tokens[i]);
66          i++;
67      }
68      printf("\n");
69      return 1;
70  }
71
72  void print_mode(mode_t mode) {
73      printf((S_ISDIR(mode)) ? "d" : "-");
74      printf((mode & S_IRUSR) ? "r" : "-");
75      printf((mode & S_IWUSR) ? "w" : "-");
76      printf((mode & S_IXUSR) ? "x" : "-");
77      printf((mode & S_IRGRP) ? "r" : "-");
```

```
78      printf((mode & S_IWGRP) ? "w" : "-");
79      printf((mode & S_IXGRP) ? "x" : "-");
80      printf((mode & S_IROTH) ? "r" : "-");
81      printf((mode & S_IWOTH) ? "w" : "-");
82      printf((mode & S_IXOTH) ? "x" : "-");
83  }
84
85  int func_ls(char** tokens) {
86      char* path = malloc(sizeof(char) * BUFSIZE);
87      getcwd(path, BUFSIZE);
88      DIR* dir;
89      struct dirent *ptr;
90      struct stat statbuf;
91
92      if ((dir = opendir(path)) == NULL) {
93          perror("Open dir error");
94          exit(1);
95      }
96      int l =0;
97      if(tokens[1] &&strcmp(tokens[1], "-l") == 0)
98      l=1;
99
100     while ((ptr = readdir(dir)) != NULL) {
101         if (strcmp(ptr->d_name, ".") == 0 || strcmp(ptr->d_name, "..") == 0) {
102             continue;
103         }
104
105         char fullpath[BUFSIZE];
106         snprintf(fullpath, sizeof(fullpath), "%s/%s", path, ptr->d_name);
107
108         if (l) {
109             if (stat(fullpath, &statbuf) == -1) {
110                 perror("Failed to get file status");
111                 continue;
112             }
113
114             print_mode(statbuf.st_mode);
115             printf(" %ld", statbuf.st_nlink);
116             printf(" %s", getpwuid(statbuf.st_uid)->pw_name);
117             printf(" %s", getgrgid(statbuf.st_gid)->gr_name);
118             printf(" %ld", statbuf.st_size);
119
120             char time_str[BUFSIZE];
121             strftime(time_str, sizeof(time_str), "%b %d %H:%M", localtime(&statb
122             printf(" %s", time_str);
123         }
124
```

```
125         printf(" %s\n", ptr->d_name);
126     }
127 
128     closedir(dir);
129     free(path);
130     return 1;
131 }
132 char** split_line(char* line){
133     int i = 0;
134     char** tokens = malloc(sizeof(char*) * TOKENNUM);
135     char* token;
136     if(!tokens){
137         printf("allocation failed!\n");
138         exit(1);
139     }
140     token = strtok(line, " ");
141     while (token != NULL){
142         tokens[i] = token;
143         i++;
144         token = strtok(NULL, " ");
145     }
146     tokens[i] = NULL;
147     return tokens;
148 }
149 
150 int outter_commands(char** tokens){
151     int pid = fork();
152     int status;
153     if(pid < 0){
154         fprintf(stderr, "Fork Failed\n");
155     }
156     else if (pid == 0){
157         if (execvp(tokens[0], tokens) == -1){
158             perror("myshell ");
159             exit(1);
160         }
161     }
162     else {
163         while(1){
164             pid = wait(&status);
165             if(pid == -1){
166                 break;
167             }
168             else{
169                 WEXITSTATUS (status);
170             }
171         }
```

```
172     }
173     return 1;
174 }
175 int execute_func(int i,char**tokens){
176 switch(i)
177 {
178 case 0:return func_cd(tokens);
179 case 1:return func_pwd(tokens);
180 case 2:return func_exit(tokens);
181 case 3:return func_echo(tokens);
182 case 4:return func_ls(tokens);
183 }
184 }
185 int execute(char** tokens){
186     if(tokens[0] == NULL){
187         return 1;
188     }
189     for(int i=0;i<num;i++){
190         if(strcmp(tokens[0], commands[i]) == 0){
191              return execute_func(i,tokens);
192 
193         }
194     }
195     return outter_commands(tokens);
196 }
197 
198 int execute_line(char* line){
199     char** cmds = malloc(sizeof(char*) * 8);
200     for (int i = 0; i < 8; i++){
201         cmds[i] = malloc(sizeof(char) * 128);
202     }
203     int i = 0, j = 0, k=0;
204     while(line[i] != '\0'){
205         if(line[i] == ';'){
206             cmds[j][k] = '\0';
207             i++;
208             j++;
209             k = 0;
210         }else{
211             if(line[i] == '\t'){
212                 line[i] = ' ';
213             }
214             cmds[j][k] = line[i];
215             i++;
216             k++;
217         }
218     }
```

```
219     cmds[j][k] = '\0';
220     int cmd_num = j + 1;
221     int flag;
222     for (int i = 0; i < cmd_num; i++){
223         char** tokens = split_line(cmds[i]);
224 
225         flag = execute(tokens);
226         if(flag == 0){
227             break;
228         }
229     }
230     for (int i = 0; i < 8; i++){
231         free(cmds[i]);
232     }
233     free(cmds);
234     return flag;
235 }
236 
237 void shell(){
238     struct passwd *pwdinfo;
239     pwdinfo = getpwuid(getuid());
240     char* user_name = pwdinfo->pw_name;
241     //路径
242     char* path = malloc(sizeof(char) * BUFSIZE);
243     char* cmdline = malloc(sizeof(char) * BUFSIZE);
244     int flag = 1;
245     while(1){
246         memset(cmdline,'\0',(sizeof(char) * BUFSIZE));
247         getcwd(path, BUFSIZE);
248         printf("%s@%s$ ", user_name, path);
249         int i = 0;
250         char* buffer = malloc(sizeof(char) * BUFSIZE);
251         int c;
252         while (1) {
253             c = getchar();
254             if (c == EOF || c == '\n') {
255                 buffer[i] = '\0';
256                 break;
257             }
258             else {
259                 buffer[i] = c;
260                 i++;
261             }
262         }
263         cmdline = buffer;
264         //printf(cmdline);
265         flag = execute_line(cmdline);
```

```
266         if (flag == 0)break;
267 
268     }
269     free(cmdline);
270 }
271 
272 int main(){
273     printf("************ welcome to mini shell! ************\n");
274     shell(); //开启 shell 循环
275     printf("************ mini shell Exit! ************\n");
276     return 0;
277 }
278 
```

代码解析：

内部命令有"cd", "pwd", "exit", "echo", "ls"。

进入程序后首先执行的是 `shell()` 函数，开始输入的循环。

在 `shell()` 函数中，先使用 `getpwuid` 函数获取当前用户的用户信息，并将该信息存储在 `pwdinfo` 中，然后获取用户的名称，接着开始循环输入。读入命令行输入的每一个字母。结束后将命令行读入的命令传入execute_line函数开始执行。

`execute_line` 函数则是将命令分割好之后再执行 `execute` 函数，对于每一个命令，按顺序执行相对应的函数。如果是内部命令，则执行 `execute_func` 函数，否则执行外部命令函数。

execute_func函数则是选择要执行哪些函数的函数，这里使用了switch结构来选择要执行的函数。

接下来对执行的函数解析：

func_cd(char** tokens)

通过 `chdir` 函数改变当前工作目录到 `tokens[1]` 指定的路径。如果 `tokens[1]` 没有指定路径，则返回到用户的登录目录。

func_pwd(char** tokens)

这个则打印当前的工作目录，使用 `getcwd` 函数来获取当前的路径，然后输出。

func_exit(char** tokens)

这个函数是直接退出的函数。

func_echo(char** tokens)

这个函数则是读取token[0]后面所有的字符并打印。

func_ls(char** tokens)

这里区分了ls -l命令和ls命令。如果有-l命令则输出详细的详细信息。如果没有则只输出文件的名称。执行ls -l命令时，需要调用 `print_mode` 函数来确定文件的类型和权限并打印。同时遍历整个目录。

outter_commands(char** tokens)

这是执行外部命令的函数，当内部命令查找不到时则执行这个函数，比如执行date。

实验结果

```
************ welcome to mini shell! ************
cgn2112966@/home/cgn2112966/lab11$ cd ../../
cgn2112966@/home$ cd /2112966
myshell: No such file or directory
cgn2112966@/home$ cd ./cgn2112966
cgn2112966@/home/cgn2112966$ cd
cgn2112966@/home/cgn2112966$ cd ../../../
cgn2112966@/$ cd
cgn2112966@/home/cgn2112966$
```

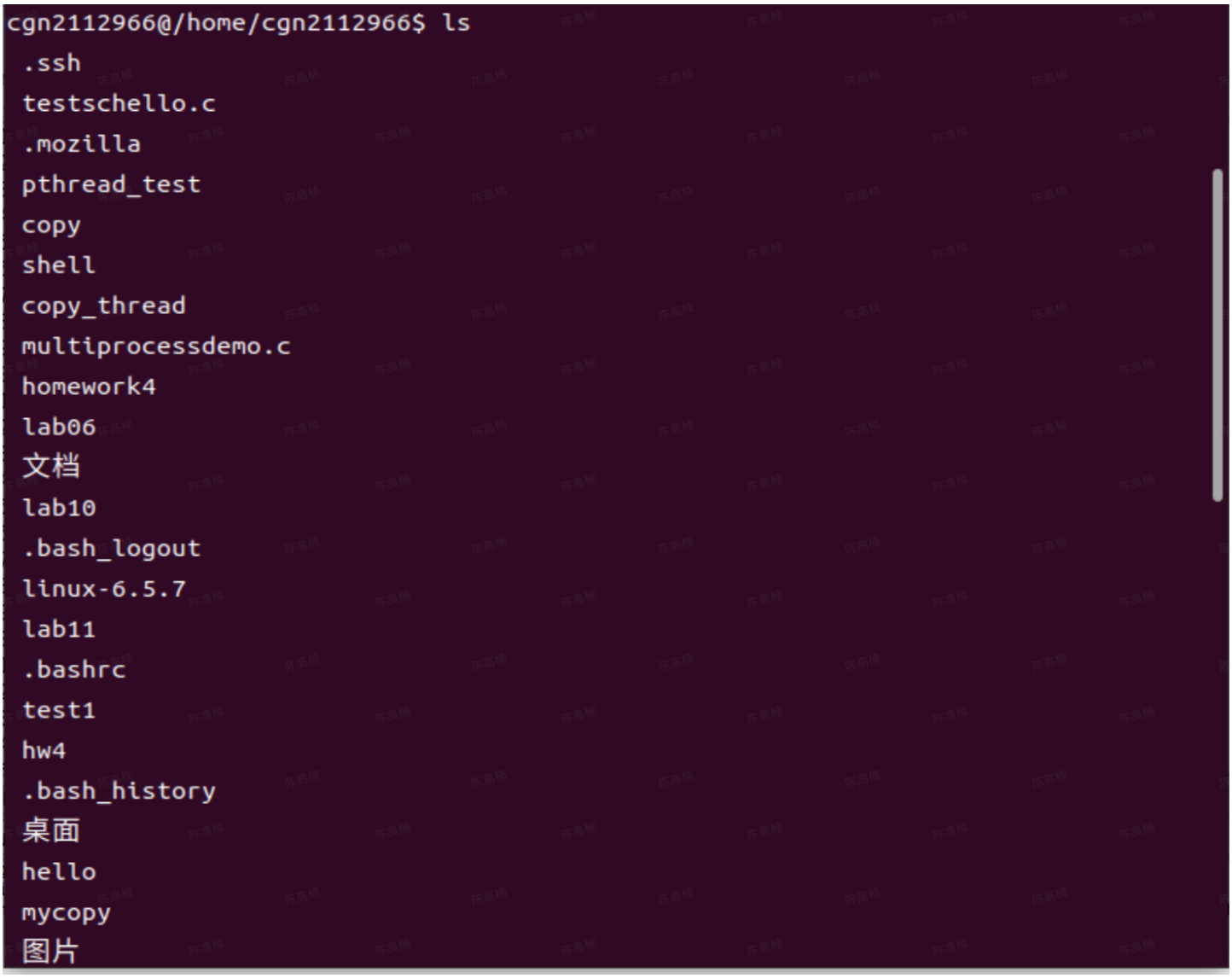
cd命令可正常运行

```
cgn2112966@/home/cgn2112966$ echo hello world
hello world
```

echo命令可正常运行

```
cgn2112966@/home/cgn2112966$ pwd
current path is : /home/cgn2112966
```

pwd命令可正常运行


cgn2112966@/home/cgn2112966$ ls
.ssh
testschello.c
.mozilla
pthread_test
copy
shell
copy_thread
multiprocessdemo.c
homework4
lab06
文档
lab10
.bash_logout
linux-6.5.7
lab11
.bashrc
test1
hw4
.bash_history
桌面
hello
mycopy
图片

ls命令可正常运行

```
cgn2112966@/home/cgn2112966$ ls -l
drwx------ 2 cgn2112966 cgn2112966 4096 Oct 12 17:00 .ssh
-rw-r--r-- 1 cgn2112966 cgn2112966 236 Nov 16 12:32 testschello.c
drwx------ 3 cgn2112966 cgn2112966 4096 Nov 10 14:21 .mozilla
drwxr-xr-x 4 cgn2112966 cgn2112966 4096 Nov 18 21:54 pthread_test
-rwxrwxr-x 1 cgn2112966 cgn2112966 16816 Nov 10 14:44 copy
drwxr-xr-x 2 cgn2112966 cgn2112966 4096 Dec 15 16:46 shell
-rwxrwxr-x 1 cgn2112966 cgn2112966 17616 Nov 18 22:23 copy_thread
-rw-rw-r-- 1 cgn2112966 cgn2112966 1124 Nov 10 14:20 multiprocessdemo.c
drwxr-xr-x 2 cgn2112966 cgn2112966 4096 Oct 20 15:15 homework4
drwxr-xr-x 5 cgn2112966 cgn2112966 4096 Nov 10 14:20 lab06
drwxr-xr-x 2 cgn2112966 cgn2112966 4096 Sep 19 01:09 文档
drwxr-xr-x 4 cgn2112966 cgn2112966 4096 Dec 07 17:26 lab10
-rw-r--r-- 1 cgn2112966 cgn2112966 220 Sep 19 01:04 .bash_logout
drwxrwxr-x 26 cgn2112966 cgn2112966 4096 Dec 12 12:41 linux-6.5.7
drwxr-xr-x 4 cgn2112966 cgn2112966 4096 Dec 20 21:00 lab11
-rw-r--r-- 1 cgn2112966 cgn2112966 3771 Sep 19 01:04 .bashrc
drwxr-xr-x 4 cgn2112966 cgn2112966 4096 Oct 26 19:30 test1
drwxr-xr-x 4 cgn2112966 cgn2112966 4096 Oct 26 15:44 hw4
-rw------- 1 cgn2112966 cgn2112966 9765 Dec 20 20:56 .bash_history
drwxr-xr-x 2 cgn2112966 cgn2112966 4096 Sep 19 01:09 桌面
```

ls -l命令可正常运行

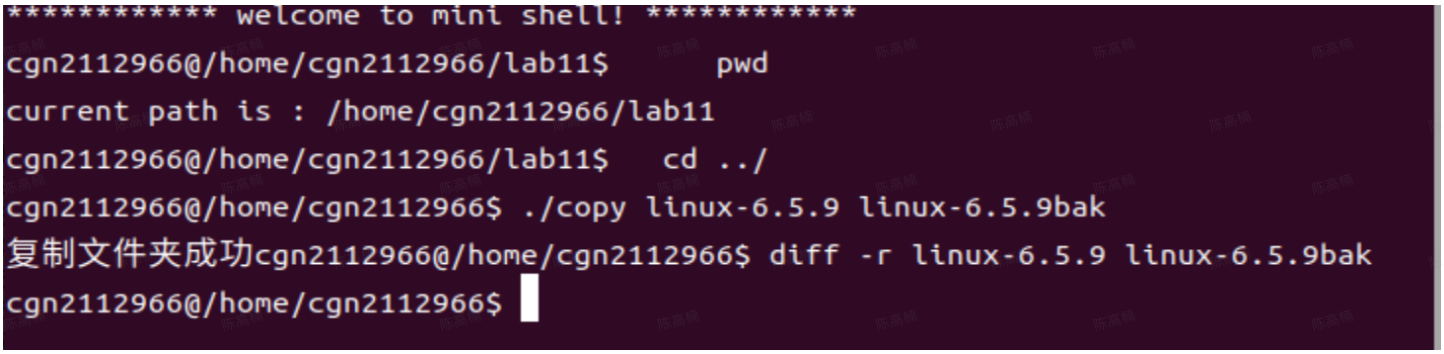
```
cgn2112966@/home/cgn2112966/lab11$ pwd;ls -l;date
current path is : /home/cgn2112966/lab11
drwxr-xr-x 3 cgn2112966 cgn2112966 4096 Dec 19 18:10 obj
drwxr-xr-x 3 cgn2112966 cgn2112966 4096 Dec 19 18:10 bin
-rw-rw-r-- 1 cgn2112966 cgn2112966 222 Dec 21 15:09 lab11.depend
-rw-r--r-- 1 cgn2112966 cgn2112966 6602 Dec 21 15:12 main.c
-rw-rw-r-- 1 cgn2112966 cgn2112966 354 Dec 20 21:00 lab11.layout
-rw-rw-r-- 1 cgn2112966 cgn2112966 1001 Dec 19 16:58 lab11.cbp
2023年 12月 21日 星期四 15:13:02 CST
```

采用分号来隔离命令，运行pwd；ls -l；date；均正常运行

```
cgn2112966@/home/cgn2112966/lab11$ exit
************ mini shell Exit! ************
```

exit命令可正常运行

忽略空格和tab 键成功



执行外部命令成功

实验心得体会

本次实验我通过编写程序实现了一个minishell，并实现了循环输入命令并执行，将结果显示在输出上。实现之后非常有成就感。这次作业锻炼了我的代码能力，令我受益匪浅。