操作系统实验二 Shell 的实现

71121117 马骁宇

2023.11.21

一、实验内容

实现具有管道、重定向功能的 shell，能够执行一些简单的基本命令，如进程执行、列目录等。

二、实验目的

通过实验，让学生了解 Shell 实现机制。

三、实验流程与结果展示

（一）管道的实现——myShell.c 代码设计

（1）命令执行和管道处理

该部分主要负责处理命令的执行，包括管道操作和输入/输出重定向。pipel 函数用于处理管道，而 redirect 函数用于处理输入/输出重定向。

```
68  struct HELP_DOC * help_doc;
69
70  int pipel(char *cmd) {…
98
99  int redirect (char *cmd) {…
207
208 int is_back(char *order) {
```

（2）后台任务管理

处理后台任务的管理，包括检测命令是否需要在后台运行、处理子进程的退出和停止信号。

```
53  int is_back(char *order);
54  void handle_sigchld(int s);
55  void handle_sigstop(int s);
56  // back process
57  typedef struct BACK_JOBS {
58      pid_t pid ;
59      char * cmd;
60      int status;
61  } BACK_JOB ;
```

（3）命令解析和执行

包含一些辅助函数，用于解析和执行命令，如去除空格、检查命令是否存在、执行 cd 命令、显示帮助文档以及打印当前工作目录。

```
42  char * trim (const char *str);
43  char * if_exist (char *order);
44  void do_help(char *);
45  void do_pwd();
46  void do_cd(char *argv[]);
47
```

（4）信号处理

处理信号，主要是中断信号（SIGINT），用于处理用户键入 Ctrl+C 的情况。

```
47
48  void handle_sigint (int s);
49
```

（5）帮助文档和初始化

用于初始化帮助文档结构，并提供显示帮助文档的函数。

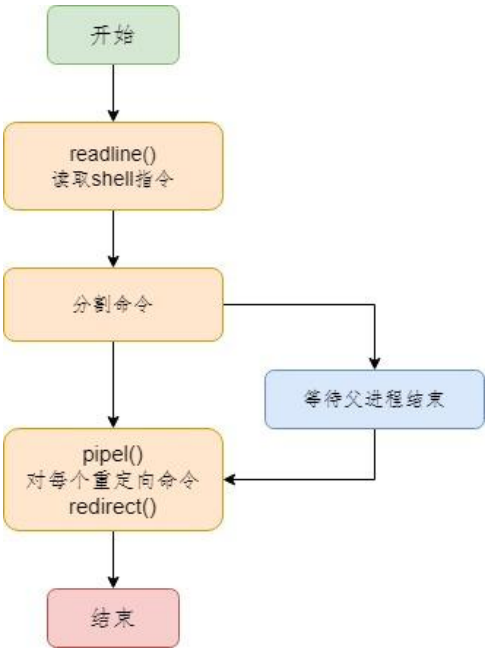
```
37
38  struct HELP_DOC {
39      char * usage[lengthOfBUILTIN_COMMANDS];
40      char * info [lengthOfBUILTIN_COMMANDS];
41  } ;
42
43  void initWithHelpDoc (struct HELP_DOC*);
```

（6）主函数和自动补全

包含主函数，其中包括程序的入口点，以及使用 readline 库实现的命令自动补全功能。

```
409 > char * command_generator (const char *text, int state) {···
427
428 > char ** command_completion (const char *text, int start, int end) {···
436
437 v void initialize_readline () {
438       rl_attempted_completion_function = (rl_completion_func_t *)command_completion;
439   }
440
441 > int main (void) {···
```

（二）算法流程图



（三）myShell.c 的编译执行

gcc 编译后执行即可，其中 readline 库和 termcap 库需要手动 wget 并安装。

```
[root@localhost exp2]# ls
myShell.c
[root@localhost exp2]# gcc myShell.c -o myShell -lreadline -ltermcap
myShell.c:238: 警告： 与 'do_help' 类型冲突
myShell.c:50: 警告： 'do_help' 的上一个声明在此
[root@localhost exp2]# ls
myShell  myShell.c
[root@localhost exp2]# ./myShell
------------*Welcome*------------
root:/home/seu/exp2$
```

（三）测试 myShell

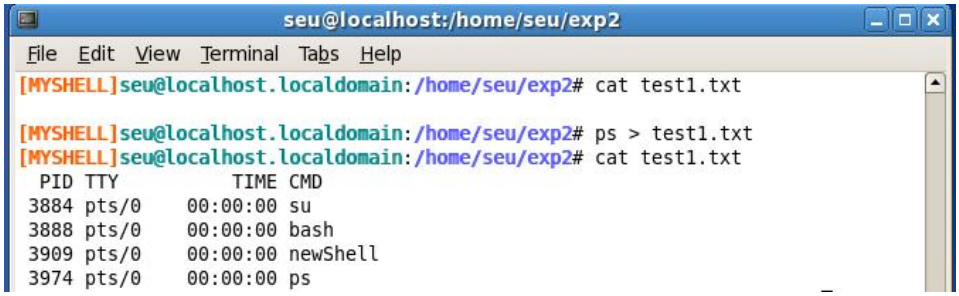
（1）执行并进入 myShell

```
[root@localhost exp2]# ./myShell
------------*Welcome*------------
root:/home/seu/exp2$
```

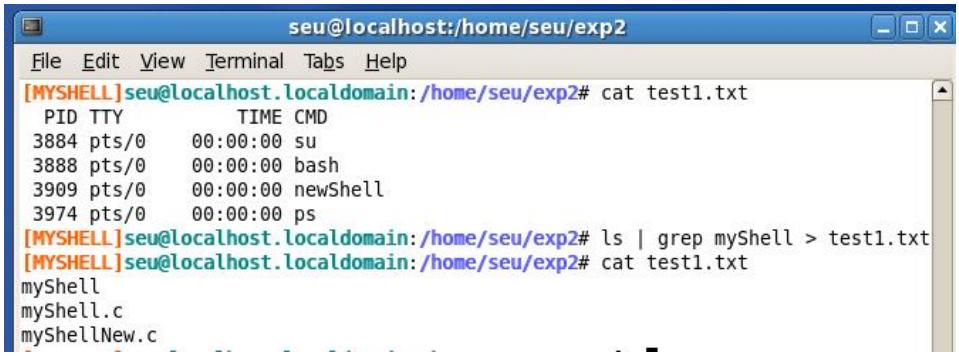
（2）测试部分 shell 指令

（注：a 和 b 的测试后来补的，截图格式上稍微有些差异）

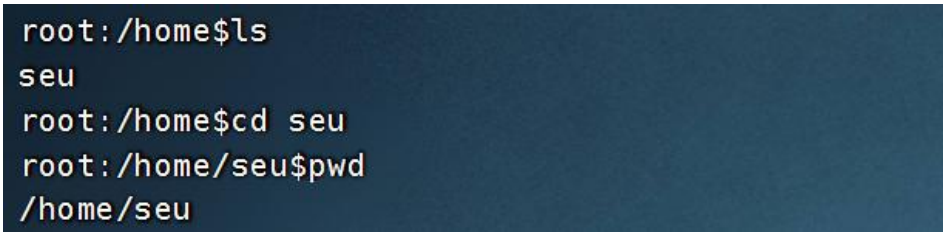
a. 重定向：ps > test1.txt



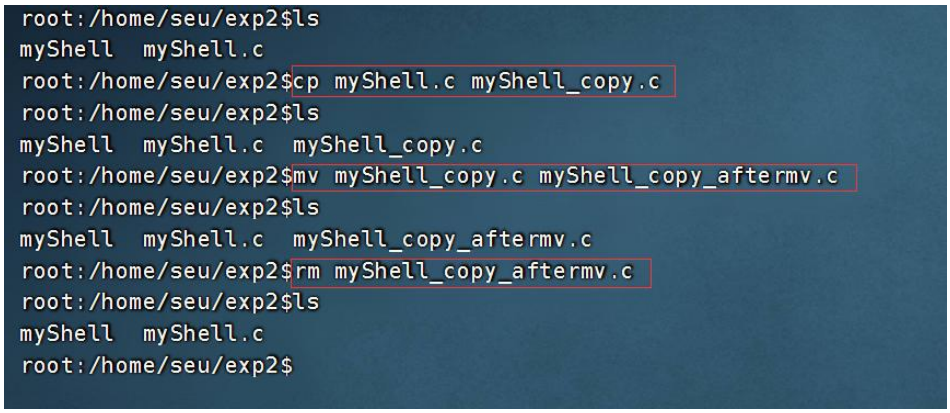
b. 重定向与管道：ls | grep myShell > test1.txt



c. cd / pwd / ls



d. cp / mv / rm



e. vi / cat

```
root:/home/seu/exp2$cat myShell.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <grp.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <readline/readline.h>
```

（下图为执行 vi 后部分界面）

```
static int back_jobs_ptr = 0;
char *st[] = { "DONE" , "RUNNING" , "STOPPED" };
int currentpid; // current running pid of shell
int isCtrlz;
enum BUILTIN_COMMANDS { NO_SUCH_BUILTIN =0, EXIT, CD, HISTORY, DO_HIS_CMD , PWD, KILL , HELP , JOBS };
//for tab completion
char *commands[] = { "cd" , "cp" , "chmod" ,
-- INSERT --
```

f. help / ps

```
root:/home/seu$help cd
cd: cd [dir]

        The default DIR is the value of the HOME shell variable.
root:/home/seu$ps
  PID TTY          TIME CMD
 4007 pts/0    00:00:00 su
 4087 pts/0    00:00:00 bash
 9176 pts/0    00:00:00 ps
27172 pts/0    00:00:00 myShell
root:/home/seu$
```

g. exit

```
27172 pts/0    00:00:00 myShell
root:/home/seu$exit
-------------*Goodbye*-------------
[root@localhost exp2]#
```

四、实验体会

本次实验涉及对 Shell 功能的简单实现，其中包括管道和重定向。由于对 Linux 和系统调用等不够熟悉，我难以独立从零实现，因此在网络上查阅并修改代码以实现解决方案。首先，实现对运行中进程任务（jobs）的存储需要采用链表的方式，同时父子进程之间的联系需要通过信号进行处理。接下来，我着手实现了管道和重定向，在标准输入输出和用户定义的输入输出口之间进行了转接，并实现了各个功能。

最后在编译 myshell.c 文件时，在阅读实验手册时我了解了 GCC 动态链接库的使用技巧。成功编译并实现了 Shell 的功能。

五、源代码与注释

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <grp.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <signal.h>
#include <readline/readline.h>
#include <readline/history.h>

#define STD_IN 0
#define STD_OUT 1
#define MAXORD 20
#define MAXPARA 8
#define MAX_BACK_JOBS_NUM 20
#define SIGSTCP 20
#define lengthOfBUILTIN_COMMANDS 10
```

```
static int back_jobs_ptr = 0;
char *st[] = { "DONE" , "RUNNING" , "STOPPED" };
int currentpid; // current running pid of shell
int isCtrlz;
enum BUILTIN_COMMANDS { NO_SUCH_BUILTIN =0, EXIT, CD, HISTORY, DO_HIS_CMD , PWD, KILL , HELP , JOBS };
//for tab completion
char *commands[] = { "cd" , "cp" , "chmod" ,
                     "exit" ,
                     "mv" , "man" ,
                     "rm" , "rmdir" ,
                     "vi" , "bg" ,"fg" ,"grep" ,"ls" ,"cat" ,
                     "history" , "help" ,
                     "jobs" , "kill",
                     "ps" , "pwd" };
```

```
struct HELP_DOC {
    char * usage[lengthOfBUILTIN_COMMANDS];
```

```
    char * info [lengthOfBUILTIN_COMMANDS];
} ;
char * trim (const char *str);
int is_back(char *order);
int pipel(char *cmd);
char * if_exist (char *order);
void handle_sigchld(int s);
void handle_sigint (int s);
void handle_sigstop(int s);
void initWithHelpDoc (struct HELP_DOC*);
void do_help(char *);
void do_pwd();
```

```
// back process
typedef struct BACK_JOBS {
    pid_t pid ;
    char * cmd;
    int status;
} BACK_JOB ;
```

```
typedef struct Node {
    int id;
    char cmd[100];
    struct Node * next;
} NODE ;
```

```
struct BACK_JOBS *back_jobs[MAX_BACK_JOBS_NUM];
static NODE *head;
struct HELP_DOC * help_doc;
```

```
int pipel(char *cmd) {
    char * trim (const char *str);
    int redirect (char *cmd);
```

```
    int fd[2], status, pid;
    char *order, *other;
    order = trim( strtok(cmd, "|" ));
    other = trim( strtok (NULL, "" ));
    if (!other)
        redirect(order);
    else {
        pipe(&fd[0]);
        if ((pid = fork ()) == 0) {
            close(fd[0]);
            close(STD_OUT);
            dup(fd[1]);
```

```
                close(fd[1]);
                redirect(order);
            } else {
                close(fd[1]);
                close(STD_IN);
                dup(fd[0]);
                close(fd[0]);
                waitpid (pid, &status, 0);
                pipel(other);
            }
        }
        return 1;
    }
```

```
    int redirect (char *cmd) {
        char * trim (const char *str);
        void do_cd(char *argv[]);

        char *order = trim(cmd), *order_path, *real_order;
        char *infile, *outfile, *arg[MAXPARA];
        int i, type = 2, fd_out, fd_in;

        for (i = 0; i < strlen(cmd); ++i) {
            if (cmd[i] == '<')
                ++type;
            if (cmd[i] == '>')
                type = type * 2;
        }
        if (type == 3 || type == 6)
            real_order = trim( strtok (cmd, "<" ));
        else if (type == 4 || type == 5)
            real_order = trim( strtok (cmd, ">" ));
        else if (type == 2)
            real_order = trim(cmd);
        else {
            fprintf (stderr, "#error: bad redirection form\n" );
            return -1;
        }
```

```
        arg[0] = trim( strtok (real_order, " " ));
        for (i = 1; (arg[i] = trim( strtok (NULL, " " ))) != NULL; ++i);
        if (strcmp (arg[0], "history" ) == 0) {
            while (head->next != NULL) {
                printf ("id:%d %s\n" , head->id , head->cmd);
                head = head->next;
            }
            exit(1);
```

```
        return 1;
    }
    if (strcmp (arg[0], "jobs" ) == 0) {
        int i = 1;
        for (; i < MAX_BACK_JOBS_NUM; ++i) {
            if (back_jobs[i] != NULL)
                printf ("[%d] %d %s\t\t\t\t%s\n" , i, back_jobs[i]-> pid ,
                    st[back_jobs[i]-> status], back_jobs[i]-> cmd);
        }
        exit(1);
        return 1;
    }
    if (strcmp (arg[0], "help" ) == 0) {
        do_help(arg[1]);
        exit(1);
        return 1;
    }
    if (strcmp (arg[0], "pwd" ) == 0) {
        do_pwd();
        exit(1);
        return 1;
    }
    if ((order_path = if_exist(arg[0])) == NULL) {
        fprintf (stderr, "#error: this command doesn't exist\n" );
        exit(1);
        return -1;
    }
    switch (type) {
        case 2:
            break;
        case 3:
            infile = trim( strtok (NULL, "" ));
            break;
        case 4:

            outfile = trim( strtok(NULL, "" ));
            break;
        case 5:

            outfile = trim( strtok(NULL, "<" ));
            infile = trim( strtok (NULL, "" ));
            break;
        case 6:

            infile = trim( strtok (NULL, ">" ));
            outfile = trim( strtok(NULL, "" ));
            break;
```

```
            default :
                return -1;
        }
```

```
        if (type == 4 || type == 5 || type == 6) {
            if ((fd_out = creat(outfile, 0755)) == -1) {
                fprintf (stderr, "#error: redirect standard out error\n" );
                return -1;
            }
            close(STD_OUT);
            dup(fd_out);
            close(fd_out);
        }

        if (type == 3 || type == 5 || type == 6) {
            if ((fd_in = open(infile, O_RDONLY, S_IRUSR | S_IWUSR)) == -1) {
                fprintf (stderr, "#error: can't open inputfile '%s'\n" , infile);
                return -1;
            }
            close(STD_IN);
            dup(fd_in);
            close(fd_in);
        }
        execv(order_path, arg);
```

```
        exit(0);
        return 1;
    }
```

```
    int is_back(char *order) {
        int len = strlen(order);
        if (order[len - 1] == '&' ) {
            order[len] = '\0';
            return 1;
        } else
            return 0;
    }
```

```
    void do_cd(char *argv[]) {
        if (argv[1] != NULL) {
            if (chdir (argv[1]) < 0) {
                switch (errno) {
                    case ENOENT:
                        fprintf (stderr, "#error: directory can't be found\n" );
                        break ;
                    case ENOTDIR:
```

```
                    fprintf (stderr, "#error: this is not a directory name\n" );
                    break ;
                case EACCES:
                    fprintf (stderr, "#error: you have no right to access\n" );
                    break ;
                default :
                    fprintf (stderr, "#error: unknown error\n" );
            }
        }
    }
}
```

```
do_help(char *argv){
    int i;
    if(argv == NULL){
        i = HELP;
    }
    else if(strcmp(argv, "cd" )==0){
        i = CD;
    } else if(strcmp(argv, "exit" )==0){
        i = EXIT;
    } else if(strcmp(argv, "history" )==0){
        i = HISTORY;
    } else if(strcmp(argv, "pwd" )==0){
        i = PWD;
    } else if(strcmp(argv, "help" )==0){
        i = HELP;
    } else if(strcmp(argv, "jobs" )==0){
        i = JOBS;
    }
    printf ("%s\n" ,help_doc-> usage[i]);
    printf ("%s\n" ,help_doc-> info [i]);
    return ;
}
```

```
char * if_exist(char *order) {
    char *all_path, *p, *path, *buffer;
    int len;
    all_path = getenv("PATH" );
    buffer = trim(all_path);
    len = strlen(all_path) + strlen(order);
    if ((path = ( char *) malloc(len * ( sizeof(char) ))) == 0) {
        fprintf (stderr, "#error: can't malloc enough space for buffer\n" );
        return NULL;
    }
    p = strtok (buffer, ":" );
    while (p) {
```

```
            strcat(strcat(strcpy(path, p), "/" ), order);
            if (access(path, F_OK) == 0) {
                return path;
            }
            p = strtok (NULL, ":" );
        }
        strcpy(path, order);
        if (access(path, F_OK) == 0)
            return path;
        return NULL;
    }
```

```
    char * trim (const char *str) {
        int i, j, k;
        char *order;
        if (str == NULL)
            return NULL;
        for (i = 0; i < strlen(str); ++i)
            if (str[i] != ' ' && str[i] != ' ')
                break ;
            for (j = strlen(str) - 1; j > -1; --j)
                if (str[j] != ' ' && str[j] != ' ')
                    break ;
            if (i <= j) {
                if ((order = ( char *) malloc((j - i + 2) * ( sizeof(char )))) == 0) {
                    fprintf (stderr, "#error: can't malloc enough space\n" );
                    return NULL;
                }
                for (k = 0; k < j - i + 1; ++k)
                    order[k] = str[k + i];
                order[k] = '\0';
                return order;
            } else
        return NULL;
    }
```

```
    void handle_sigchld( int s) {
    /* execute non-blocking waitpid, loop because we may only receive
    * a single signal if multiple processes exit around the same time.
    */
        // printf("recieve %d pid %d.\n",s,currentpid);
```

```
        int i = 1;
        if(isCtrlz == 0) {
            for (; i < MAX_BACK_JOBS_NUM; ++i) {
                if (back_jobs[i] == NULL)
                    continue;
```

```
                if (back_jobs[i]-> pid == currentpid) {
                    back_jobs[i]-> status = 0;
                    break;
                }
            }
        } else{
            isCtrlz = 0;
        }
        pid_t pid;
        while ((pid = waitpid (0, NULL, WNOHANG)) > 0) {
            int i = 1;
            for (; i < MAX_BACK_JOBS_NUM; ++i) {
                if (back_jobs[i] == NULL)
                    continue;
                if (back_jobs[i]-> pid == pid) {
                    back_jobs[i]-> status = 0;
                    break ;
                }
            }
        }
    }
```

```
    void handle_sigint (int s) {
        //printf("A sigint receive.");
        return ;
    }
```

```
    void handle_sigstcp(int s) {
        int i = 1;
        isCtrlz = 1; //告诉另一个信号处理函数，这是一个 ctrlz
        int flag = 0; //如果等于 0，则代表这个程序没有被后台过。
        printf ("\n" );
        // kill(currentpid,SIGSTCP);
        for (; i < MAX_BACK_JOBS_NUM; ++i) {
            if (back_jobs[i] == NULL)
                continue;
            if (back_jobs[i]-> pid == currentpid) {
                back_jobs[i]-> status = 2;
                flag = 1;
                break ;
            }
        }
```

```
        if(flag == 0){
            ++back_jobs_ptr;
            back_jobs[back_jobs_ptr] = ( struct BACK_JOBS *) malloc(sizeof(struct BACK_JOBS *));
            back_jobs[back_jobs_ptr]-> pid = currentpid;
```

```
            back_jobs[back_jobs_ptr]-> cmd = (char *) malloc(100);
            strcpy(back_jobs[back_jobs_ptr]-> cmd, "This is a stop process." );
            back_jobs[back_jobs_ptr]-> status = 2;
            printf ("[%d] %d %s\t\t\t\t%s\n" , back_jobs_ptr, back_jobs[back_jobs_ptr]-> pid ,
                    st[back_jobs[back_jobs_ptr]-> status], back_jobs[back_jobs_ptr]-> cmd);
        }
        return ;
    }

    void initWithHelpDoc (struct HELP_DOC *help_doc) {
        help_doc-> usage[ EXIT] = "exit: exit" ;
        help_doc-> info [EXIT ] = "Exit the shell." ;

        help_doc-> usage[CD] = "cd: cd [dir]" ;
        help_doc-> info [CD] = "\n\tThe default DIR is the value of the HOME shell variable." ;

        help_doc-> usage[HISTORY] = "history: history [-c] [-s num]" ;
        help_doc-> info [HISTORY ] = "\n\tentry with a `*'.\n\t\n\t -s num\tsize of the history buffer to num" ;

        help_doc-> usage[PWD] = "pwd: pwd" ;
        help_doc-> info [PWD] = "Print the name of the current working directory." ;

        help_doc-> usage[HELP ] = "help: help [pattern ...]" ;
        help_doc-> info [HELP ] = "\n\t PATTERN Pattern specifiying a help topic" ;

        help_doc-> usage[JOBS] = "jobs: jobs" ;
        help_doc-> info [JOBS] = "\n\tLists the active jobs. JOBSPEC restricts output to that job." ;
    }

    void do_pwd() {
        char dirname[100];
        if(getcwd(dirname, 99) == NULL) {
            fprintf (stderr, "getcwd error\n" );
        }
        else {
            printf ("%s \n" ,dirname);
        }
    }

    char * command_generator (const char *text, int state) {
        char *name;
        static int list_index, len;

        if (!state) {
            list_index = 0;
```

```
            len = strlen (text);
        }

        while (name = commands[list_index]) {
            ++list_index;

            if (strncmp(name, text, len) == 0)
                return (strdup (name));
        }

        return ((char *)NULL);
    }

    char ** command_completion (const char *text, int start, int end) {
        char **matches = NULL;

        if (start == 0)
            matches = rl_completion_matches (text, command_generator);

        return (matches);
    }

    void initialize_readline () {
        rl_attempted_completion_function = (rl_completion_func_t *)command_completion;
    }

    int main (void) {
        signal(SIGCHLD, handle_sigchld);
        signal(SIGINT, handle_sigint);
        signal(SIGSTCP, handle_sigstcp);

        char all_order[100], *order[MAXORD];
        int i, pid, status, number = 1, back, historyid = 1;
        char buf[80],prompt[100];
        char tmp[80];
        memset(tmp, 0, sizeof(tmp));
        char *username, *arg[MAXPARA];
        struct group *data;

        head = (NODE *) malloc( sizeof(NODE) );
        strcat (head->cmd, "intial" );
        data = getgrgid(getgid());
        username = data->gr_name;
        strcat (tmp, "/home/" );
```

```
        strcat (tmp, username);
```

```
        //help 文档
        help_doc = ( struct HELP_DOC *) malloc(20* sizeof(struct HELP_DOC *));
        initialize_readline();
        initWithHelpDoc(help_doc);
        printf ("--------------*Welcome*--------------\n" );
        while (1) {
            getcwd(buf, sizeof(buf));
            if (strcmp(tmp, buf) == 0) {
                memset(buf, 0, sizeof(buf));
                buf[0] = '~';
                buf[1] = '\0';
            }
            sprintf (prompt, "%s:%s$" , username, buf);
            // printf("%s",prompt);
            // strcat(prompt,username);
            // strcat(prompt,":");
            // strcat(prompt,buf);
            // strcat(prompt,"$");
            // printf("%s",prompt);
```

```
            //fgets(all_order,sizeof(all_order),stdin);
            // gets(all_order);
            char *t;
            t = readline (prompt);
            sprintf (all_order, "%s" , t);
            if (all_order == NULL || trim(all_order) == NULL)
                continue;
            //储存历史命令
            NODE *next;
            next = (NODE *) malloc( sizeof(NODE) );
            next-> id = historyid++;
            strcpy(next-> cmd, trim(all_order));
            next-> next = head;
            head = next;
            add_history(all_order);
```

```
            back = is_back(trim(all_order));
            if (strcmp(trim(all_order), "exit" ) == 0) {
                int i = 1;
                int e = 1;
                for (; i < MAX_BACK_JOBS_NUM; ++i) {
                    if (back_jobs[i] != NULL) {
                        if (back_jobs[i]-> status != 0) {
                            e = 0;
                            break;
```

```
                }
            }
        }
        if (e == 0) {
            fprintf (stdout, "Some jobs are undone, please stop them first.\n" );
            fprintf (stdout, "Type \"jobs\" to see them.\n" );
            continue;
        } else {
            printf ("--------------*Goodbye*--------------\n" );
            exit(-1);
        }
    }

    if (trim(all_order)[0] == '!' &&trim(all_order)[1]== '-') {
        int i;
        sscanf(trim(all_order), "!-%d" , &i);
        if(i < 0)
            fprintf (stderr, "!# # must be a negative number." );
        NODE * p;
        p = head;
        int j,flag = 0;
        for(j = 0; j < i; ++j){
            p = p-> next;
            if(p == NULL){
                fprintf (stderr, "!# # must be a less than history number." );
                flag=1;
            }
        }
        if(flag == 1) continue;
        strcpy(all_order,p-> cmd);
        --historyid;
        head=head->next;
    } else if(trim(all_order)[0] == '!'){
        int i;
        sscanf(trim(all_order), "!%d" , &i);
        if(i < 0)
            fprintf (stderr, "!# # must be a negative number." );
        i = historyid-i - 1;
        NODE * p;
        p = head;
        int j, flag = 0;
        for(j = 0; j < i; ++j){
            p = p->next;
            if( p== NULL){
                fprintf (stderr, "!# # must be a less than history number." );
                flag = 1;
            }
```

```
        }
        if(flag == 1)
            continue;
        strcpy(all_order,p-> cmd);
        --historyid;
        head = head->next;
    }

    if(trim(all_order)[0]== 'f' &&trim(all_order)[1] == 'g'){
        int i;
        sscanf(trim(all_order), "fg%d" , &i);
        if(back_jobs[i]->status == 0){
            continue;
        }
        currentpid=back_jobs[i]-> pid ;
        kill (back_jobs[i]-> pid ,SIGCONT);
        back_jobs[i]-> status = 1;
        waitpid (back_jobs[i]-> pid, &status, WUNTRACED);
        continue;
    }
```

```
    if(trim(all_order)[0]== 'b' && trim(all_order)[1]== 'g'){
        int i;
        sscanf(trim(all_order), "bg%d" , &i);
        if(back_jobs[i]->status == 0){
            printf ("It is already done.\n" );
            continue;
        }
        currentpid = 0;
        // printf("%d",back_jobs[i]->pid);
        kill (back_jobs[i]-> pid ,SIGCONT);
        back_jobs[i]->status = 1;
        continue;
    }
```

```
    arg[0] = trim( strtok (trim(all_order), " " ));
    for (i = 1; (arg[i] = trim( strtok (NULL, " " ))) != NULL; ++i);
    if (strcmp(arg[0], "cd" ) == 0) {
        do_cd(arg);
        continue;
    }
    order[0] = trim( strtok (all_order, "&" ));
    for (i = 1; (order[i] = trim( strtok (NULL, "&" ))) != NULL; ++i)
        ++number;
    for (i = 0; i < number - 1; ++i) {
        if (fork () == 0) {
            pipel(order[i]);
```

```
                }
            }
            //不是后台程序运行方案
            if ((pid = fork ()) == 0) {
                pipel(order[i]);
            } else if (back == 0){
                currentpid = pid;
                waitpid (pid, &status, WUNTRACED);
            } else {
                ++back_jobs_ptr;
                back_jobs[back_jobs_ptr] = ( struct BACK_JOBS *) malloc(sizeof(struct BACK_JOBS *));
                back_jobs[back_jobs_ptr]->pid = pid;
                back_jobs[back_jobs_ptr]->cmd = (char *) malloc(100);
                strcpy(back_jobs[back_jobs_ptr]-> cmd, all_order);
                back_jobs[back_jobs_ptr]->status = 1;
                printf ("[%d] %d %s\t\t\t\t%s\n" , back_jobs_ptr,
                        back_jobs[back_jobs_ptr]->pid ,
                        st[back_jobs[back_jobs_ptr]->status],
                        back_jobs[back_jobs_ptr]->cmd);
            }

            number = 1;
        }

        return 1;
    }
```