Shell 的实现

Author：57119108 吴桐
Date：2021.7.30

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

1、学习并理解 linux 中 shell 的知识；
2、重点学会编程实现管道和重定向的功能；
3、实现自己的 shell

三、设计思路和流程图

1、对输入的解析

本次实验的第一个难题，就是对输入的命令进行解析。最简单的想法就是按照空格来划分，之后按照“<”、“>”、“|”等符号进行分类处理。

2、简单命令的执行

使用函数 execvp 可以实现简单的命令，函数原型为 int execvp(const char *file ,char * const argv []);，execvp()会从 PATH 环境变量所指的目录中查找符合参数 file 的文件名，找到后便执行该文件，然后将第二个参数 argv 传给该欲执行的文件。为了不造成阻塞，这里启用了一个新线程实现它；父进程需等待子进程，以回收分配给它的资源。

3、输入输出重定向的实现

实现重定向的主要函数是 freopen，可以将预定义的标准流文件定向到自定义的文件中。只要能够在命令解析时准确提取自定义文件，重定向功能就不难实现。

4、管道功能的实现

命令之间通过“|”符号来分隔，使用 pipe 函数来建立管道。通过 strtok_r 函数来分隔命令，利用 pipe 函数生成的的读取端和写入端，第一条命令的输出作为第二条命令的输入，从而实现管道的功能。

四、源程序及注释

首先介绍一些代码中出现的库函数，有助于对代码的理解。

ssize_t getline(char **lineptr, size_t *n, FILE *stream);

按行读入字符串，头文件为 stdio.h。其中，lineptr 指向存放该行字符的指针，如果是 NULL，则有系统帮助 malloc；n 为读取字符串的最大长度，如果是由系统 malloc 的指针，则为 0；stream 为文件描述符。如果成功则返回读取的字节数，否则返回-1。

fflush()

刷新缓冲区。fflush(stdin)刷新标准输入缓冲区，把输入缓冲区内的内容概丢弃；fflush(stdout)刷新标准输出缓冲区，把输出缓冲区内的内容强制打印到标准输出设备上。

int execvp(const char* file, const char* argv[]);

执行文件。第一个参数 file 是要运行的文件，会在环境变量 PATH 中查找 file，并执行；第二个参数 argv[]是一个参数列表，argv 列表最后一个必须是 NULL。失败会返回－1， 成功无返回值。

char *strtok_r(char *str, const char *delim, char **saveptr);

分割字符串。str 为要分解的字符串，delim 为分隔符字符串。char **saveptr 参数是一个指向 char *的指针变量，用来在 strtok_r 内部保存切分时的上下文，以应对连续调用分解相同源字符串。函数返回分割后的第一段字符串。

第一次调用 strtok_r 时，str 参数必须指向待提取的字符串，saveptr 参数的值可以忽略。连续调用时，str 赋值为 NULL，saveptr 为上次调用后返回的值，不要修改。strtok_r 实际上就是将 strtok 内部隐式保存的 this 指针，以参数的形式与函数外部进行交互。调用者在连续切分相同源字符串时，除了将 str 参数赋值为 NULL，还要传递上次切分时保存下的 saveptr。

FILE *freopen(const char *path, const char *mode, FILE *stream);

把预定义的标准流文件定向到由 path 指定的文件中。其中，参数 path 为文件名，用于存储输入输出的自定义文件名；mode 为文件打开的模式，常见的用“w+”等； stream 为一个文件，通常使用标准流文件。

int dup(int oldfd);

dup 用来复制参数 oldfd 所指的文件描述符，头文件为 unistd.h。复制成功返回最小的尚未被使用过的文件描述符，若有错误则返回-1。

int dup2(int oldfd, int newfd);

dup2 与 dup 区别是 dup2 可以用参数 newfd 指定新文件描述符的数值。若参数 newfd 已经被程序使用，则系统就会将 newfd 所指的文件关闭；若 newfd 等于 oldfd，则返回 newfd, 而不关闭 newfd 所指的文件。dup2 所复制的文件描述符与原来的文件描述符共享各种文件状态。共享所有的锁定，读写位置和各项权限或 flags 等。若 dup2 调用成功则返回新的文件描述符，出错则返回-1。

在 shell 的重定向功能中，就是通过调用 dup 或 dup2 函数对标准输入和标准输出的操作来实现的。

具体代码如下，并附有详细注释。

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <sys/stat.h>
#include<signal.h>
#include <fcntl.h>

//0:stdin 1:stdout 2:stderr
//execute the basic order
void execute(char** argv)
{
	pid_t pid;
	int status;
	//fork child process
	if ((pid = fork()) < 0) {
		printf("**Error:Fork failed.\n");
		exit(1);
	}
	else if (pid == 0) {	//child process
		// invalid command && not 'cd'
		if (execvp(argv[0], argv) < 0 && strcmp(argv[0], "cd"))
			printf("**Error:Invalid command.\n");
		exit(0);
	}
	else {	//father wait for child process
		while (wait(&status) != pid);
	}
}

//split the command
void readIn(char* comm, char** argv)
{
	int count = 0;
	memset(argv, 0, sizeof(char*) * (64));
	char* lefts = NULL;
	while (1) {
		char* p = strtok_r(comm, " ", &lefts);
		if (p == NULL) {break;}
		argv[count] = p;	//argv is an array, it stores each value of your input divided by " "
		comm = lefts;
		count++;
	}
```

```
        if (strcmp(argv[0], "exit") == 0) exit(0);
        else if (strcmp(argv[0], "cd") == 0) {
                int ch = chdir(argv[1]);
        }
}

//remove extra spaces
char* removeSpace(char* st) {
        int i = 0;
        int j = 0;
        char* ptr = malloc(sizeof(char*) * strlen(st));
        for (i = 0; st[i] != '\0'; i++)
                if (st[i] != ' ')
                {
                        ptr[j] = st[i];
                        j++;
                }
        ptr[j] = '\0';
        st = ptr;
        return st;
}

//output exludes the first element
//">"
void    execute_file(char** argv, char* output) {
        pid_t pid;
        int status, flag;
        char* file = NULL;
        if ((pid = fork()) < 0) {
                printf("**Error:Fork failed.\n");
                exit(1);
        }
        else if (pid == 0) {    //child process
                if (strstr(output, ">") > 0) {    //returns a pointer to first occurence after ">" or null
                  //more than one ">"
                        char* p = strtok_r(output, ">", &file);
                        //output includes words between two ">"
                        file = removeSpace(file);
                        // get words after (second) ">"
                        flag = 1;
                        int old_stdout = dup(1);
                        FILE* fp1 = freopen(output, "w+", stdout);    // new output
                        execute_file(argv, file);    //deal with commands after ">"
                        fclose(stdout);    // close new output
```

```
                FILE* fp2 = fdopen(old_stdout, "w");
                *stdout = *fp2;    // return to the old output
                exit(0);
            }
            if (strstr(output, "<") > 0) {
                char* p = strtok_r(output, "<", &file);
                file = removeSpace(file);
                // get words after "<"
                flag = 1;
                int fd = open(file, O_RDONLY);
                if (fd < 0) {
                    printf("**Error:No such file or directory.");
                    exit(0);
                }
            }
            if (strstr(output, "|") > 0)
            {
                char* p = strtok_r(output, "|", &file);
                file = removeSpace(file);
                // get words after "|"
                flag = 1;
                char* args[64];
                readIn(file, args);
                execute(args);
            }
            int old_stdout = dup(1);    //stdout
            FILE* fp1 = freopen(output, "w+", stdout);    //new output
            if (execvp(argv[0], argv) < 0)
                printf("**Error:Already in execution.");
            fclose(stdout);    // close new output
            FILE* fp2 = fdopen(old_stdout, "w");
            *stdout = *fp2;    //return to the old output
            exit(0);
        }
        else { while (wait(&status) != pid); }
}

//output exludes the first element
//"<"
void execute_input(char** argv, char* output)
{
        pid_t pid;
        int fd;
        char* file;
```

```
int flag = 0;
int status;
if ((pid = fork()) < 0) {
        printf("**Error:Fork failed\n");
        exit(1);
}
else if (pid == 0) {    //child process
        if (strstr(output, "<") > 0) {
                char* p = strtok_r(output, "<", &file);
                file = removeSpace(file);
                // get words after (second) “<”
                flag = 1;
                fd = open(output, O_RDONLY);
                if (fd < 0) {
                        printf("**Error:No such file or directory.");
                        exit(0);
                }
                output = file;
        }
        if (strstr(output, ">") > 0) {
                char* p = strtok_r(output, ">", &file);
                file = removeSpace(file);
                // get words after “>”
                flag = 1;
                int old_stdout = dup(1);
                FILE* fp1 = freopen(file, "w+", stdout);    //new output
                execute_input(argv, output);    //deal with commands after “>”
                fclose(stdout);
                FILE* fp2 = fdopen(old_stdout, "w");
                *stdout = *fp2;    //return to the old output
                exit(0);
        }
        if (strstr(output, "|") > 0) {
                char* p = strtok_r(output, "|", &file);
                file = removeSpace(file);
                // get words after “|”
                flag = 1;
                char* args[64];
                readIn(file, args);
                int pfds[2];
                pid_t pid, pid2;
                int status, status2;
                pipe(pfds);    //create pipe
                //pfds[0]:read    pfds[1]:write
```

```
        int fl = 0;
        if ((pid = fork()) < 0 || (pid2 = fork()) < 0) {
                printf("**Error:Fork failed.\n");
                exit(1);
        }
        if (pid == 0 && pid2 != 0) {
                close(1);    //stdout
                dup(pfds[1]);    //pfds[1]:write
                close(pfds[0]);
                close(pfds[1]);
                fd = open(output, O_RDONLY);    //output includes words between "<" and "|"
                close(0);    //stdin
                dup(fd);
                if (execvp(argv[0], argv) < 0) {
                        close(pfds[0]);
                        close(pfds[1]);
                        printf("**Eerror:Already in execution.");
                        fl = 1;
                        exit(0);
                }
                close(fd);
                exit(0);
        }
        else if (pid2 == 0 && pid != 0 && fl != 1) {
                close(0);
                dup(pfds[0]);
                close(pfds[1]);
                close(pfds[0]);
                if (execvp(args[0], args) < 0)
                {
                        close(pfds[0]);
                        close(pfds[1]);
                        printf("**Eerror:Already in execution.");
                        exit(0);
                }
        }
        else {    //father process
                close(pfds[0]);
                close(pfds[1]);
                while (wait(&status) != pid);
                while (wait(&status2) != pid2);
        }
        exit(0);
}
```

```
            fd = open(output, O_RDONLY);
            close(0);
            dup(fd);
            if (execvp(argv[0], argv) < 0) printf("**Eerror:Already in execution.");
            close(fd);
            exit(0);
        }
        else {while (wait(&status) != pid);}
}

void execute_pipe(char** argv, char* output) {
        int pfds[2], pf[2], flag;
        int status, status2, old_stdout;
        char* file;
        pid_t pid, pid2, pid3;
        pipe(pfds);    //create pipe
        //pfds[0]:read    pfds[1]:write
        int blah = 0;
        char* args[64];
        char* argp[64];
        int fl = 0;
        if ((pid = fork()) < 0 || (pid2 = fork()) < 0) {
            printf("**Error:fork failed\n");
            exit(1);
        }
        if (pid == 0 && pid2 != 0) {
            close(1);
            dup(pfds[1]);
            close(pfds[0]);
            close(pfds[1]);
            if (execvp(argv[0], argv) < 0)    //run the command
            {
                close(pfds[0]);
                close(pfds[1]);
                printf("**Error: Already in execution.");
                fl = 1;
                kill(pid2, SIGUSR1);
                exit(0);
            }
        }
        else if (pid2 == 0 && pid != 0) {
            if (fl == 1) { exit(0); }
            if (strstr(output, "<") > 0) {
                char* p = strtok_r(output, "<", &file);
```

```
            file = removeSpace(file);
            // get the first word after <
            flag = 1;
            readIn(output, args);    //divide output to the array args
            execute_input(args, file);
            close(pfds[0]);
            close(pfds[1]);
            exit(0);
        }
        if (strstr(output, ">") > 0) {
            char* p = strtok_r(output, ">", &file);
            file = removeSpace(file);
            // get words after “>”
            flag = 1;
            readIn(output, args);    //divide output to the array args
            blah = 1;
        }
        else { readIn(output, args); }
        close(0);
        dup(pfds[0]);
        close(pfds[1]);
        close(pfds[0]);
        if (blah == 1) {
            old_stdout = dup(1);
            FILE* fp1 = freopen(file, "w+", stdout);
        }
        if (execvp(args[0], args) < 0) {
            fflush(stdout);
            printf("**Error: PID %d Already in execution.\n", pid);
            fflush(stdout);
            kill(pid, SIGUSR1);
            close(pfds[0]);
            close(pfds[1]);
        }
        fflush(stdout);
        if (blah == 1) {
            fclose(stdout);
            FILE* fp2 = fdopen(old_stdout, "w");
            *stdout = *fp2;    //return to the old output
        }
    }
    else {
        close(pfds[0]);
        close(pfds[1]);
```

```
            while (wait(&status) != pid);
            while (wait(&status2) != pid2);
        }
}

void execute_pipe2(char** argv, char** args, char** argp) {
        int status;
        int i;
        int pipes[4];
        pipe(pipes);
        pipe(pipes + 2);
        if (fork() == 0) {
                dup2(pipes[1], 1);    //oldfd, newfd
                for (i = 0; i < 4; i++) close(pipes[i]);
                if (execvp(argv[0], argv) < 0) {
                        fflush(stdout);
                        printf("**Error: Already in execution.");
                        fflush(stdout);
                        for (i = 0; i < 4; i++) close(pipes[i]);
                        exit(1);
                }
        }
        else {
                if (fork() == 0) {
                        dup2(pipes[0], 0);
                        dup2(pipes[3], 1);
                        for (i = 0; i < 4; i++) close(pipes[i]);
                        if (execvp(args[0], args) < 0) {
                                fflush(stdout);
                                printf("**Error: Already in execution.");
                                fflush(stdout);
                                for (i = 0; i < 4; i++) close(pipes[i]);
                                exit(1);
                        }
                }
                else {
                        if (fork() == 0) {
                                dup2(pipes[2], 0);
                                for (i = 0; i < 4; i++) close(pipes[i]);
                                if (execvp(argp[0], argp) < 0) {
                                        fflush(stdout);
                                        printf("**Error: Already in execution.");
                                        fflush(stdout);
                                        for (i = 0; i < 4; i++) close(pipes[i]);
```

```
                    exit(1);
                }
            }
        }
    }
    for (i = 0; i < 4; i++) close(pipes[i]);
    for (i = 0; i < 3; i++) wait(&status);
}

int main()
{
    char line[1024];
    char* argv[64];
    char* args[64];
    char* left;
    size_t size = 0;
    char ch;
    int count = 0;
    char* file;

    while (1) {
        count = 0;
        int flag = 0;
        char* comm = NULL;
        char* dire[] = { "pwd" };

        fflush(stdout);    //clear stdout

        execute(dire);    //show the path
        printf("COCOT-SHELL~#:");

        int len = getline(&comm, &size, stdin);

        if (*comm == '\n') continue;    //only input Enter

        comm[len - 1] = '\0';
        char* file = NULL;
        int i = 0;
        char* temp = (char*)malloc(150);
        strcpy(temp, comm);
        readIn(temp, argv);    //divide comm to the array argv

        if (strcmp(comm, "exit") == 0) exit(0);
```

```
//classify -first round
for (i = 0; comm[i] != '\0'; i++) {
	if (comm[i] == '>') {
		char* p = strtok_r(comm, ">", &file);
		file = removeSpace(file);	//exclude first element
		flag = 1;
		break;
	}
	else if (comm[i] == '<') {
		char* p = strtok_r(comm, "<", &file);
		file = removeSpace(file);	//exclude first element
		flag = 2;
		break;
	}
	else if (comm[i] == '|') {
		char* p = strtok_r(comm, "|", &left);
		flag = 3;
		break;
	}
}

if (flag == 1) {
	readIn(comm, argv);	//divide comm to the array argv
	execute_file(argv, file);
}
else if (flag == 2) {
	readIn(comm, argv);	//divide comm to the array argv
	execute_input(argv, file);
}
else if (flag == 3) {
	char* argp[64];
	char* file;
	//left is the second part (after "|")
	if (strstr(left, "|") > 0) {	//more than one "|"
		char* p = strtok_r(left, "|", &file);
		readIn(comm, argv);	//first part
		readIn(left, args);	//second part (after "|")
		readIn(file, argp);	//left part
		execute_pipe2(argv, args, argp);
	}
	else {
		readIn(comm, argv);
		execute_pipe(argv, left);
	}
```

```
        }
        else {
            readIn(comm, argv);
            execute(argv);
        }
    }

}
```

五、实验测试结果

```
[seu@localhost Desktop]$ gcc -o myshell myshell.c
[seu@localhost Desktop]$ ./myshell
/home/seu/Desktop
COCOT-SHELL~#:ls
a.txt     commfile~           myshell    test2     test3     test.c
b.txt     linux-2.6.21        myshell.c  test2.c   test3.c   test.c~
commfile  linux-2.6.21.tar.gz  test       test2.c~  test3.c~
/home/seu/Desktop
COCOT-SHELL~#:ls -l
total 54260
-rw-rw-r--  1 seu  seu        24 2021-07-29 01:35 a.txt
-rw-rw-r--  1 seu  seu        24 2021-07-29 01:36 b.txt
-rw-rw-r--  1 seu  seu        64 2021-07-22 18:03 commfile
-rw-rw-r--  1 seu  seu        65 2021-07-22 18:02 commfile~
drwxr-xr-x 20 seu  seu      4096 2021-07-22 04:52 linux-2.6.21
-rwxrw-rw-  1 seu  seu  55328580 2014-06-03 06:41 linux-2.6.21.tar.gz
-rwxrwxr-x  1 seu  seu     12883 2021-07-29 01:37 myshell
-rwxrw-rw-  1 seu  seu     13916 2021-07-29 01:37 myshell.c
-rwxrwxr-x  1 seu  seu      4805 2021-07-22 18:03 test
-rwxr-xr-x  1 root root     4838 2021-07-22 18:05 test2
-rw-rw-r--  1 seu  seu       181 2021-07-22 07:45 test2.c
-rw-rw-r--  1 seu  seu       181 2021-07-22 07:44 test2.c~
-rwxr-xr-x  1 root root     4838 2021-07-22 18:05 test3
-rw-rw-r--  1 seu  seu       218 2021-07-22 07:47 test3.c
-rw-rw-r--  1 seu  seu       218 2021-07-22 07:46 test3.c~
-rw-rw-r--  1 seu  seu       156 2021-07-20 06:11 test.c
-rw-rw-r--  1 seu  seu       156 2021-07-20 06:09 test.c~
/home/seu/Desktop
COCOT-SHELL~#:ls -l|wc
     18     138    1010
/home/seu/Desktop
COCOT-SHELL~#:ls -l|wc >a.txt
/home/seu/Desktop
COCOT-SHELL~#:cat a.txt
     18     138    1010
/home/seu/Desktop
COCOT-SHELL~#:cat < a.txt > b.txt
/home/seu/Desktop
COCOT-SHELL~#:cat b.txt
     18     138    1010
```

图 1

```
/home/seu/Desktop
COCOT-SHELL~#:ps aux > a.txt
/home/seu/Desktop
COCOT-SHELL~#:cat a.txt
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2136   628 ?        Ss   00:56   0:01 init [5]

root         2  0.0  0.0      0     0 ?        S    00:56   0:00 [migration/0]
root         3  0.0  0.0      0     0 ?        SN   00:56   0:00 [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S    00:56   0:00 [watchdog/0]
root         5  0.0  0.0      0     0 ?        S    00:56   0:00 [migration/1]
root         6  0.0  0.0      0     0 ?        SN   00:56   0:00 [ksoftirqd/1]
root         7  0.0  0.0      0     0 ?        S    00:56   0:00 [watchdog/1]
root         8  0.0  0.0      0     0 ?        S<   00:56   0:00 [events/0]
root         9  0.0  0.0      0     0 ?        S<   00:56   0:00 [events/1]
root        10  0.0  0.0      0     0 ?        S<   00:56   0:00 [khelper]
root        11  0.0  0.0      0     0 ?        S<   00:56   0:00 [kthread]
root        51  0.0  0.0      0     0 ?        S<   00:56   0:00 [kblockd/0]
```

图 2

六、实验总结

本次实验主要是在 linux 环境下编程，最大的困难在于 lunix 环境下 c 语言库函数的运用。我搜索了大量的资料，参考了前辈的代码，尽自己可能理解代码的逻辑和实现细节。通过这种方式，可以在短时间内迅速学习难度较大的知识，毕竟实践才是进步的最大动力。