

2022-Fall-Computer Network-PJ1-Code Document

2022-Fall-Computer Network-PJ1-Code Document

- 一、组员及分工
- 二、基本情况说明
 - 1、配置情况
 - 2、FTP连接的建立流程
- 三、核心功能说明
 - 1、get
 - 2、put
 - 3、delete
 - 4、ls
 - 5、cd
 - 6、mkdir
 - 7、pwd
 - 8、quit
- 四、其他功能设计
 - 1、登录
 - 2、注册
 - 3、权限管理
 - 4、cd命令格式补充
 - 5、颜色显示
- 五、异常处理方案
 - 1、用户输入非法
 - 2、用户操作非法

一、组员及分工

姓名	学号	分工	贡献度
买巫予骛	20300290032	实现FTP系统的注册、登录功能，完成代码文档并录制视频	25%
陆一杨	20300200013	实现ls、cd、delete、mkdir等指令	25%
彭一博	19307130188	实现quit、pwd等指令、实现FTP系统的权限管理	25%
徐靖文	20302010021	实现put、get指令	25%

二、基本情况说明

1、配置情况

a.本次FTP开发于Linux环境，使用VS Code进行开发。除了实现PJ文档要求的基本功能外，还实现了用户登录、注册、权限分级管理等功能，并对系统的鲁棒性进行了设计。

b.程序运行说明：

1. 在client_linux目录下执行：`make` 命令进行编译、`make clean` 命令删除可运行文件、`sudo ./client localhost` 命令运行客户端程序。
2. 在server_linux目录下执行：`make` 命令进行编译、`make clean` 命令删除可运行文件、`sudo ./server` 命令运行客户端程序。

c.登录名和密码在 `FTPInC` 目录下的隐藏文件 `.passwd` 中。每一条记录包含用户名、密码、权限。

2、FTP连接的建立流程

- 1) 运行起服务器端程序后，首先建立一个端口号为 `LISTEN_PORT` 的监听端口，用于接收客户端程序的建立连接的申请。而运行客户端程序后，客户端就会根据localhostIP地址、主动连接这个端口号为 `LISTEN_PORT` 的监听端口，发出连接申请。
- 2) 一旦收到客户端建立连接的请求，服务端程序就fork一个子程序来对该客户端进行服务，建立起控制连接端口。
- 3) 此时，若客户端发出合法命令，客户端与服务端就新建一个数据连接端口进行数据传输，数据传输完成后就关闭这个数据连接端口。
- 4) 若客户端发出quit指令，则客户端与服务端释放控制连接端口，客户端终止客户程序、服务端同时终止fork出的子程序。

三、核心功能说明

1、get

- **格式：** `get [remote_filename]`
- **功能：** Copy file with the name `[remote_filename]` from remote directory to local directory.
- **权限：**
- **实现思路：** 首先，客户端发送get命令、并指出想要获取文件的名称到服务端。服务端接收到指令后，先判断当前文件路径是否有效，有效的话则进行文件传送、同时客户端进行接收。传送文件操作结束后，再将操作结果向客户端进行反馈、完成get操作。
- **实现细节：**

1) 首先，客户端发送get命令、与文件路径。

2) 服务器端判断用户是否打算获取子目录中的文件（为了安全考虑，系统只允许获取当前目录下的文件）。因此，调用`file_name_valid`函数来进行判断，若文件路径中包含反斜杠 `/`，那么客户端则直接返回 `PATH_FAIL`，操作结束；反之则继续执行。

```

if (!file_name_valid(file_name, sizeof(file_name))) {
    send_num(sock_fd, PATH_FAIL);
    return 0;
}

```

3) 进行发送模式判断。FTP系统提供了两种文件传输的方式：**ASCII**和**binary**。对于文本文件如 .txt、.c 等，使用**ASCII**格式进行传递，而对于可执行文件如 .o、图片文件 .jpg 等，则使用**binary**格式进行传递。在本系统的实现中，我们通过文件的后缀名来判断文件类型，进而选择文件的传递方式。

```

FILE *file = NULL;
if (strcmp(extern_name, ".txt") == 0 || strcmp(extern_name, ".c") == 0 || strcmp(extern_name, ".cpp") == 0)
    file = fopen(filepath, "r"); //ascii方式打开
else
    file = fopen(filepath, "rb"); // 二进制方式打开

```

由于本系统运行在Linux系统之上，所以不需要对文件文本的格式进行转化。

4) 首先判断文件是否存在，若不存在，则返回 FILE_UNVAIL，若文件存在，则通过open(filepath, O_RDWR)函数获取文件的信息，先计算出文件发送需要的传递次数，发送给客户端。

```

if (file == NULL) {
    send_num(sock_fd, FILE_UNVAIL); //file打开失败
    return 0;
}
else
    send_num(sock_fd, FILE_VAIL); //file已打开
struct stat s;
int fd = open(filepath, O_RDWR);
if (fstat(fd, &s) < 0) { //fd文件状态打开错误
    perror("fstat error");
    send_num(sock_fd, -1);
    return 0;
}
else
    send_num(sock_fd, s.st_size / MAX_SIZE + 1); //返回传输总次数
close(fd);

```

5) 正式传递文件开始。依次按照 MAX_SIZE 长度发送文件信息到客户端，直到文件发送结束。最后关闭file文件流。

```

char buf[MAX_SIZE];
int send_time = 0;
int size;
while (!feof(file)) {
    bzero(buf, sizeof(buf));
    size = fread(buf, 1, MAX_SIZE, file); //读取MAX_SIZE大小的文件内容
    send(sock_fd, buf, size, 0); //发送到客户端
}
fclose(file);

```

6) 至此，文件发送结束，现在在客户端进行文件接收操作。首先对服务端返回的错误代码进行输出判断，并且提前获取文件传输次数。

```

int code = get_return_code(sock_fd);
if (code == FILE_UNVAIL) printf("获取文件失败\n");
else if (code == PATH_FAIL) printf("文件名不能包含路径(斜杠)\n");
int get_num = get_return_code(sock_fd);
if (get_num < 0) printf("获取文件传输次数失败\n");

```

7) 用写模式打开文件流file后，若接收到的文件传输次数n，则循环进行n次接收文件，并写入文件流file。循环结束后，关闭文件流，文件接收操作完成。

```

FILE *file = fopen(filepath, "w");
int size;
char data[MAX_SIZE];
for (int i = 0; i < get_num; i++){
    bzero(data, sizeof(data));
    size = recv(work_fd, data, sizeof(data), 0);
    fwrite(data, 1, size, file);
}
fclose(file);

```

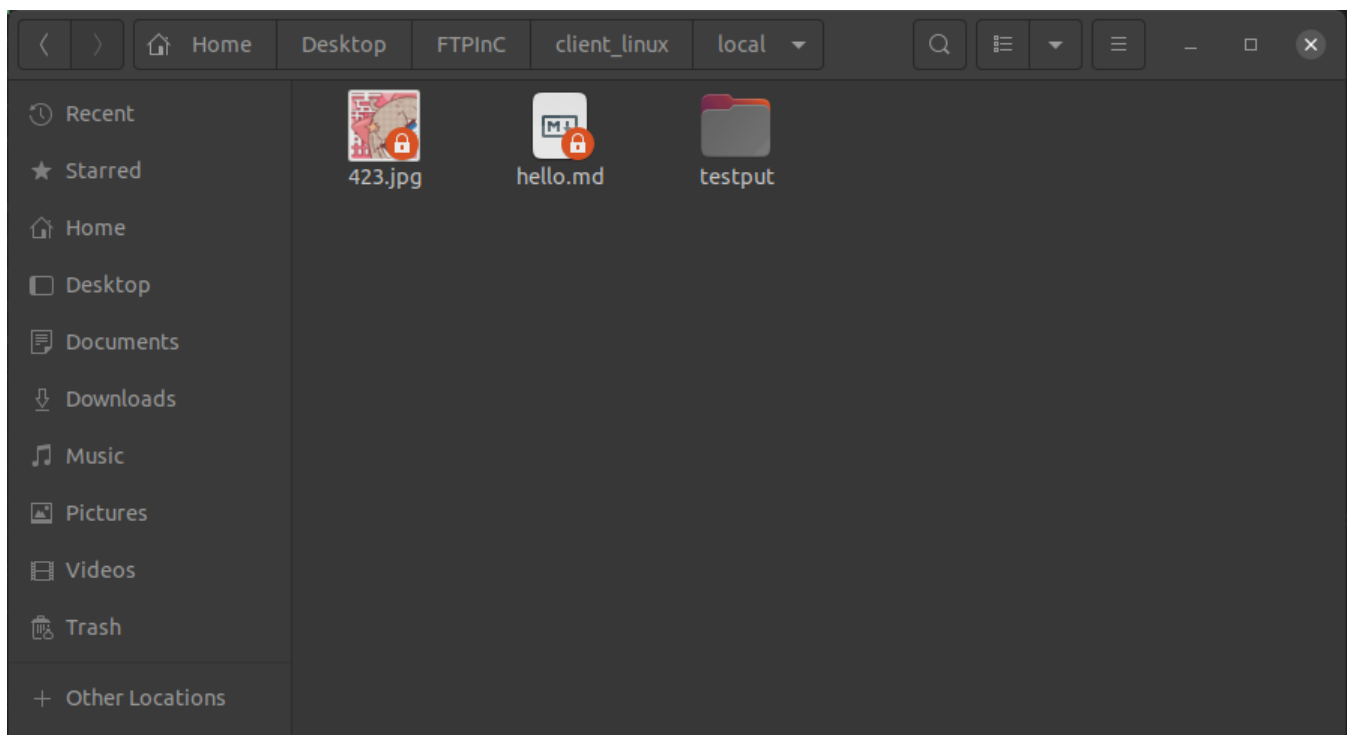
- 实现效果：

```

client> get test1/.readme
File name is not allowed to include '/'
client> get hhh.txt
file doesn't exist!
client> cd ..
change to .. succeed
client> ls
test1.txt
.
423.jpg
test.c
..
testdir2
hello.md
test.txt
testdir
test.jpg
client> get 423.jpg
File successfully received!
client> get hello.md
File successfully received!
client>

```

传输的文件存储在了local文件夹中

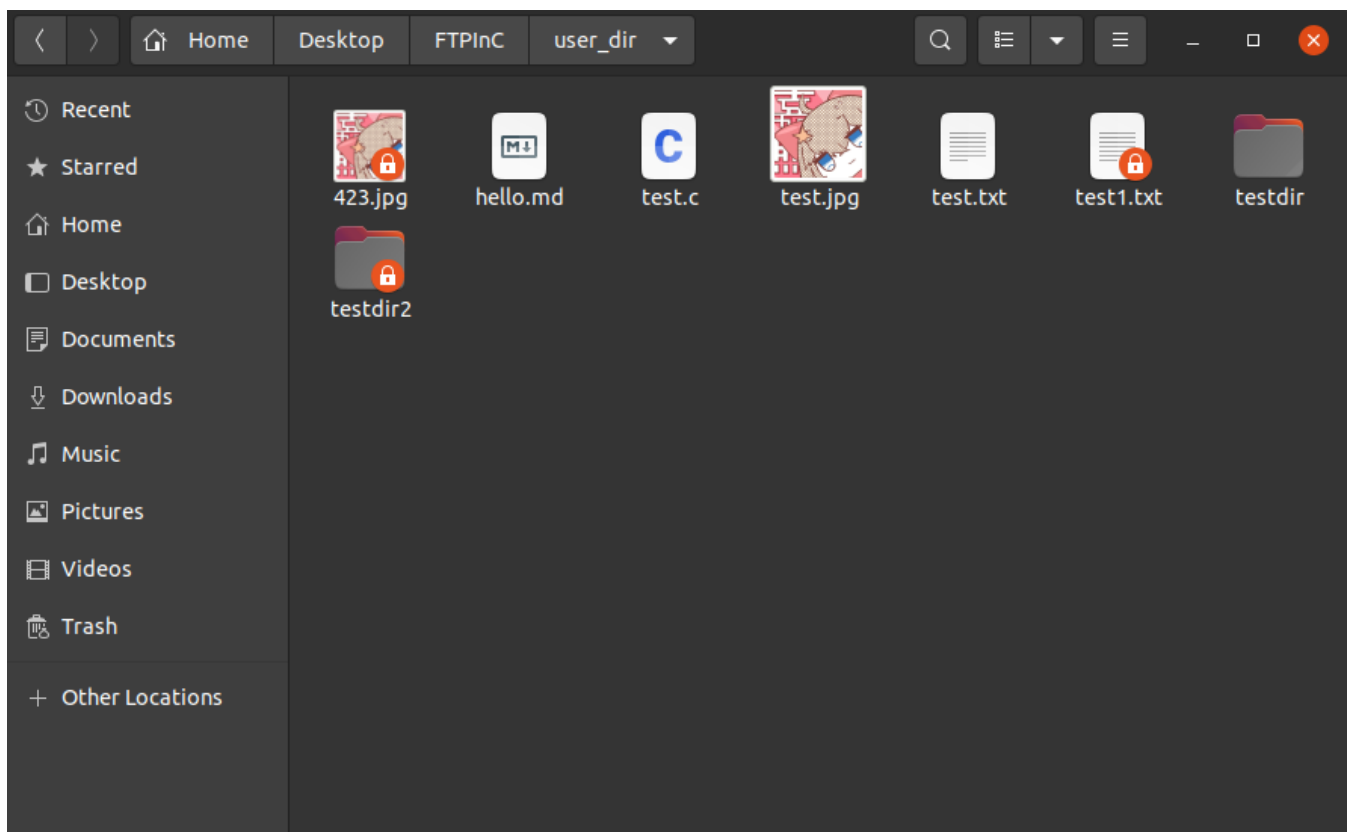


2、put

- **格式:** put [local_filename]
- **功能:** Copy file with the name [local_filename] from local directory to remote directory.
- **权限:**
- **实现思路:** 首先, 客户端发送put命令、并指出想要获取文件的名称。客户端先判断当前文件路径是否有效, 有效的话则进行文件传送、同时服务端进行接收, 将文件保存在当前目录下。接收文件操作结束后, 再将操作结果向客户端进行反馈、完成put操作。
- **实现细节:** 由于实现功能与get一致, 只是文件传输方向不同, 这里不再赘述, 可点击[跳转](#)查看。
- **实现效果:**

```
client> ls
test.c
..
testdir2
hello.md
test.txt
testdir
test.jpg
client> put test.tx
File will be transferred in binary mode
open file error: No such file or directory
client> put testdir/text.txt
File name is not allowed to include '/'
client> put test1.txt
File will be transferred in ASCII mode
File's size: 6
File successfully sent!
client> put 423.jpg
File will be transferred in binary mode
File's size: 53980
File successfully sent!
client> ls
test1.txt
.
423.jpg
test.c
..
testdir2
hello.md
test.txt
testdir
test.jpg
client>
```

接收到了文件都存储在了user_dir目录下:



3、delete

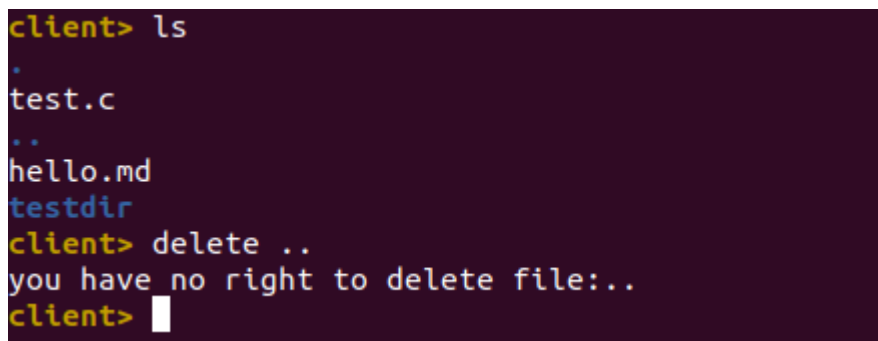
- **格式：**delete [remote_filename]
- **功能：**Delete the file with the name [remote_filename] from the remote directory.
- **权限：**
- **实现思路：**首先，客户端发送delete命令，并指出想要删除的文件的名称到服务端，由服务端进行删除操作。删除操作结束后，再将操作结果向客户端进行反馈、完成删除操作。
- **设计细节：**

1) 拒绝删除系统目录

在Linux系统中，文件目录下都会包含 `..` 和 `.` 两个目录，分别指向当前目录与上一级目录。这个用户无权删除的，尽管如此，为了提高系统的可用性，我们专门对这种情况进行了判别，并给出相应的提示。

```
//服务器代码：
if(!strcmp(file_name,"..")||!strcmp(file_name",".")){
    send_num(sock_fd, OUT_OF_AUTHORITY);
    return 0;
}
```

一旦发现用户试图删除 `..` 和 `.`，系统就返回 `OUT_OF_AUTHORITY`，告诉用户无权删除。



```
client> ls
.
test.c
..
hello.md
testdir
client> delete ..
you have no right to delete file:..
client>
```

2) 拒绝删除目录

根据PJ文档的要求，我们的delete指令只负责删除文件而不是目录，因此，我们需要对用户发过来的文件名进行判断，如果是指向的目录，则拒绝用户删除，返回 `IS_DT_DIR`。具体的实现方面，我们是通过 `readdir(direc)` 函数对当前所在目录中的内容进行遍历，一旦发现名称匹配的内容，首先判断其是否是目录，如果是则拒绝，如果不是则进行删除，如果没找到符合的文件则返回 `0`。

```
//服务器代码：
while((file = readdir(direc)) != NULL){
    if(!strcmp(file->d_name,file_name)){//找到名字匹配的内容
        if(file->d_type==DT_DIR){           //如果是目录
            send_num(sock_fd, IS_DT_DIR);
            return 0;
        }
        else{                               //找到对应文件，指向删除操作
            strcat(delete_path,file_name);
            remove(delete_path);
            return 0;
        }
    }
}
send_num(sock_fd, 0);                      //没有找到匹配的文件
```

```

client> ls
.
test.c
..
hello.md
hhh
testdir
client> delete hh
file:hh not exist
client> delete testdir
you can't delete a dictory
client> delete hhh
delete hhh succeed
client>

```

3) 拒绝删除子、父目录中的文件

为了权限和系统安全考量，我们只允许用户删除当前目录下的文件，而不能删除子目录、父目录中的文件。具体实现代码如下：

```

//服务端代码
strcat(delete_path, "..");
strcat(delete_path, current_dir);
strcat(delete_path, "/");           //拼接好路径
DIR* direc = opendir(delete_path); //打开当前所在目录
if(!strcmp(file->d_name,file_name)&&file->d_type!=DT_DIR){ //直接将参数当作文件名来进行匹配
    strcat(delete_path,file_name); //拼接文件名
    remove(delete_path);           //删除文件
}

```

可见：我们打开目录进行文件名匹配时，默认打开当前目录。因此用户传过来的参数，会全部当作文件名来进行匹配。故如果用户指向的是子、父目录中的文件，一律会返回匹配失败的结果。

```

client> ls
.
test.c
..
hello.md
testdir
client> delete testdir/hh
file:testdir/hh not exist

```

4、ls

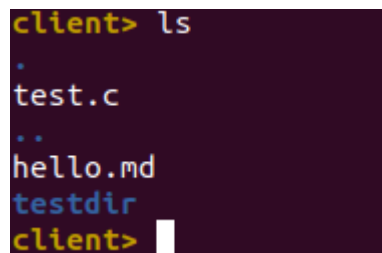
- **格式：**ls
- **功能：**List the files and subdirectories in the remote directory.
- **权限：**
- **实现思路：**首先，客户端发送ls命令，服务端收到指令后，打开当前所在目录，并通过通过**readdir(direc)**函数对当前所在目录中的内容进行遍历，记录每一个内容的名字，组成一个字节流返回客户端。客户端收到反馈后，直接将返回的字节流输出即可
- **设计细节：**

1) 格式控制

为了优化客户端显示、提高用户可用性，我们将输出流的格式都在服务端进行控制，包括：将目录的字体调整为蓝色，输出每个目录/文件的名字后就换行。实现代码如下：

```
//服务器端代码
while((file = readdir(direc)) != NULL)
{
    if(file->d_type==DT_DIR){        //如果是目录，则输出蓝色 控制符格式为 \033[1;34;49m    dir_name
    \033[0m
        type[0]='\0';
        strcat(type, "\033[1;34;49m");
        strcat(type, file->d_name);
        strcat(type, "\033[0m");
        sprintf(data + n, "%s\n", type);    //输出换行符
        n += strlen(type) + 1;
    }
    else{                            //文件则正常输出名称、换行符
        sprintf(data + n, "%s\n", file->d_name);
        n += strlen(file->d_name) + 1;
    }
}
data[n] = '\0';
closedir(direc);
```

实现效果如下：



```
client> ls
.
test.c
..
hello.md
testdir
client>
```

5、cd

- **格式：**cd [remote_directry_name] or cd ..
- **功能：**Change to the [remote_directry_name] on the remote machine or change to the parent directory of the current directory.
- **权限：**
- **实现思路：**首先，客户端发送ls命令，并且指出切换目录的路径、作为参数传递。服务端收到指令后，首先判断路径的类型（相对路径、绝对路径、符号路径、参数缺省四种），并根据不同的路径类型，确定真实的访问路径。切换路径成功/失败后分别进行反馈，完成操作。当前路径存储在一个全局变量**current_dir**字符串中，改变该数组中存储的值，就相当于改变了当前路径。
- **设计细节：**

1) 参数缺省

为了提高客户端的可用性，若用户键入 `cd` 后不输入参数，则默认是回到用户根目录。由于这个过程不涉及到检索，因此在客户端实现。代码如下：

```
if(!strcmp(path_name, "\0"))
    strcpy(path_name, "/user_dir");
```

实现效果：


```

client> pwd
/user_dir/testdir/test1
client> cd
change to /user_dir succeed
client> pwd
/user_dir
client>

```

2) 符号路径

为了提高客户端的可用性，我们为用户键入路径提供了3种符号选择：`..` 回到上级目录；`/` 回到根目录；`~` 回到根目录；这个过程在客户端实现。因为上一级目录、根目录一定存在，所以我们不需要检测输入的路径是否无效。只需要在输入 `cd ..` 回到上级目录的时候，检查是否当前已经在根目录。为了系统安全性考虑，如果前已经在根目录，则不能进入父目录。其实现代码如下：

```

if(get_path[0] == '~' || (strlen(get_path)==1&&get_path[0] == '/'))
{
    if(strlen(get_path)!=1){                                     //防止符号后跟上其他的错误符号
        send_num(sock_fd, 0);
    }
    else{
        current_dir[0]='\0';                                     //回到根目录
        strcat(current_dir, "/user_dir");
        send_num(sock_fd, SERVER_READY);
    }
}
else if(get_path[0] == '.'&& get_path[1] == '.'){ //回到上级目录
    if(strlen(get_path)!=2){
        send_num(sock_fd, 0);
    }
    else{
        for(int l=strlen(current_dir)-1;l>=0;l--){ //通过字符串分析获得上一级目录的路径地址
            if(current_dir[l]=='/'){
                if(l==0){                                     //如果当前已在根目录，则不能继续向上一级目
录切换
                    send_num(sock_fd, PATH_OUT);
                    return 0;
                }
                else{
                    current_dir[l]='\0';
                    send_num(sock_fd, SERVER_READY);
                }
                break;
            }
        }
    }
}
}
}

```

实现效果如下：

```

client> pwd
/user_dir/testdir
client> cd ..
change to .. succeed
client> pwd
/user_dir
client> cd ~
change to ~ succeed
client> pwd
/user_dir
client> cd /
change to / succeed
client> pwd
/user_dir
client>

```

3) 绝对路径

同时，用户也可以键入绝对路径，以 / 符号开头就会被判为是绝对路径输入。此时我们需要检测用户输入的路径是否存在，如不存在则返回 0 给客户端。实现代码如下：

```

if(get_path[0] == '/'){
    strcat(new_path, "..");
    strcat(new_path, get_path);           //拼接好路径
    if (!access(new_path, 0))
    {
        current_dir[0]='\0';
        strcat(current_dir, get_path);    //切换路径
        printf("change path to:%s\n", current_dir);
        send_num(sock_fd, SERVER_READY);
    }
    else                                   //路径不存在
        send_num(sock_fd, 0);
}

```

实现效果如下：

```

client> pwd
/user_dir/testdir
client> ls
.
..
test2
test1
client> cd /user_dir/testdir/test1
change to /user_dir/testdir/test1 succeed
client> pwd
/user_dir/testdir/test1
client>

```

4) 相对路径

同时，用户也可以键入相对路径以切换目录。任何不满足上述三条规则的参数输入，都会被判断为输入的相对路径。此时，我们需要检测用户输入的路径是否存在，如不存在则返回 0 给客户端。其代码实现如下：

```

strcat(new_path, "..");
strcat(new_path, current_dir);
strcat(new_path, "/");
strcat(new_path, get_path);           //拼接好路径
if (!access(new_path, 0))
{
    strcat(current_dir, "/");
    strcat(current_dir, get_path);    //切换路径
    printf("change path to:%s\n", current_dir);
    send_num(sock_fd, SERVER_READY);
}
else                                 //路径不存在
    send_num(sock_fd, 0);

```

实现效果如下：

```

client> ls
.
test.c
..
hello.md
testdir
client> cd testdir/test1
change to testdir/test1 succeed
client> pwd
/user_dir/testdir/test1
client>

```

6、mkdir

- **格式：**mkdir[remote_direcotry_name]
- **功能：**Create directory named [remote_direcotry_name] as the sub-directory of the current working directory on the remote machine.
- **权限：**
- **实现思路：**首先，客户端发送mkdir命令，并且指出想要新建的目录的名称，将其作为参数传递。服务端收到指令后，在当前目录下新建对应的子目录，如果出现名字重复、或者是嵌套建立多级子目录的情况都是不允许的。操作完成后，返回 0 代表创建目录失败、SERVER_READY 代表创建目录成功。
- **设计细节：**

1) 拒绝名称重复、或嵌套地建立多级子目录。通过设置mkdir(new_dir,0775)的创建模式 0775 实现，这个是系统代码所实现的功能。实现的部分代码如下：

```

//服务端代码
strcat(new_dir, "..");
strcat(new_dir, current_dir);
strcat(new_dir, "/");           //拼接好路径
if(recv(work_fd, get_dir, MAX_SIZE, 0) > 0){
    strcat(new_dir, get_dir);
    int isCreate = mkdir(new_dir,0775);           //创建文件夹
    if(!isCreate){                               //创建成功
        send_num(sock_fd, SERVER_READY);
    }
    else{                                         //创建失败
        send_num(sock_fd, 0);
    }
}

```

```
}  
}
```

实现效果如下：

```
client> ls  
.  
test.c  
..  
hello.md  
testdir  
client> mkdir testdir2  
testdir2 creation succeed  
client> ls  
.  
test.c  
..  
testdir2  
hello.md  
testdir  
client> mkdir testdir3/hhh  
testdir3/hhh creation failed  
client> ls  
.  
test.c  
..  
testdir2  
hello.md  
testdir  
client> mkdir testdir2  
testdir2 creation failed
```

7、pwd

- **格式：**pwd
- **功能：**Print the current working directory on the remote machine.
- **权限：**
- **实现思路：**首先，客户端发送pwd命令。服务端收到指令后，直接返回current_dir字符串中存储的当前目录给客户端。实现代码如下：

```
int server_cmd_pwd(int work_fd, int sock_fd) {  
    //打印当前所在目录  
    send_num(sock_fd, SERVER_READY);  
    char data[MAX_SIZE];  
    bzero(data, sizeof(data));  
    strcpy(data, current_dir);  
    if(send(work_fd, data, strlen(data), 0) < 0)  
    {  
        perror("send error");  
    }  
    send_num(sock_fd, RET_SUCCESS);  
    return 0;  
}
```

实现效果如下：

```
client> cd testdir/test1
change to testdir/test1 succeed
client> pwd
/user_dir/testdir/test1
client>
```

8、quit

- **格式：**quit
- **功能：**End the FTP session
- **权限：**
- **实现思路：**首先，客户端发送quit命令。服务端收到指令后，跳出命令等待循环，关闭socket，结束当前的子进程，客户端则显示相应的结束语。实现代码如下：

```
while (1) //服务端代码
{
    //接收连接
    sock_fd = accept_client(listen_fd);
    //创建子进程
    if ((pid = fork()) < 0)
    {
        perror("fork error");
        exit(1);
    }
    else if (pid == 0) // child process
    {
        close(listen_fd);
        work_process(sock_fd);      //键入quit指令后，指令从这里退出
        close(sock_fd);            //关闭当前socket
        exit(0);                   //结束子进程
    }
    close(sock_fd);
}
```

实现效果如下：

```
client> quit
Good bye!
marphownio@ubuntu:~/Desktop/FTPInC/client_linux$
```

四、其他功能设计

1、登录

- **实现思路：**为了更贴近于我们日常生活中使用的FTP系统，我们为系统设置了权限管理。这就要求我们必须先实现登录、注册操作。首先，使用 `sudo ./client localhost` 命令运行起了客户端程序后，系统会提示用户先进行登录或者是注册，用户可以通过输入字符来选择对应的功能。若选择登录功能，用户可以按照提示输入用户名及密码、提交服务端验证通过后，完成登录。
- **实现细节：**

1) 在客户端，若选择登录功能，客户端会首先发送 `LOGIN` 指令到服务端，接着用户先输入对应的用户名、客户端便发送指令 `USER [input]` 到服务器端，然后等待服务器端应答后，用户输入密码、客户端发送指令 `PASS [input]` 到服务器端，并等待服务端应答。若返回 `LOGIN_SUCCESS` 则代表验证通过，跳出循环；返回 `LOGIN_FAILED` 则代表验证失败，则继续向服务器发出 `LOGIN` 请求，继续进行循环。部分代码如下：

```
//客户机
while(1){//获取用户名
    printf("NAME: ");
    scanf("%s",user);
    //发送用户名到服务器
    strcpy(code, "USER");
    strcpy(arg, user);
    client_send_cmd(arg, code);
    //等待应答码,存放在wait
    recv(sock_fd, &wait, sizeof(wait), 0);
    //获取密码
    printf("PASSWORD: ");
    scanf("%s", pass);
    //发送密码到服务器
    strcpy(code, "PASS");
    strcpy(arg, pass);
    client_send_cmd(arg, code);
    //等待响应
    ret_code = get_return_code(sock_fd);
    if(ret_code==LOGIN_SUCCESS){
        printf("Login succeed!\n");
        break;
    }
    else if(ret_code==LOGIN_FAILED){
        printf("invalid username/password. Please try again!\n");
        send(sock_fd, "LOGIN", (int)strlen("LOGIN"), 0);
        continue;
    }
    else{
        perror("Error reading message from server");
        exit(1);
        break;
    }
}
```

2) 在服务端，服务器等待到用户发来的用户名、密码命令后，首先需要截取出参数作为对应的用户名、密码。由于用户名、密码命令的格式是已知的：`USER [input]` 和 `PASS [input]`，因此我们截取参数的时候，直接从buf[5]开始读取到结尾即可。在获取到用户名和密码后，我们需要调用`server_check`函数来检查用户名、密码是否有效。该部分的代码如下：

```

//服务器代码
if((ret = recv_data(sock_fd, buf, sizeof(buf))) < 0){//获取客户端传来的用户名命令
    perror("recv user error:");
    exit(1);
}
int n = 0, i = 5;
while(buf[i]){
    user[n++] = buf[i++];
}
send_num(sock_fd, 331);
if((ret = (recv_data(sock_fd, buf, sizeof(buf)))) < 0){
    perror("recv passwd error:");
    exit(1);
}
i = 5; n = 0;
while(buf[i])
{
    passwd[n++] = buf[i++];
}
return (server_check(user, passwd));

```

server_check作为判断当前用户名密码是否有效的函数、返回1代表有效、0代表无效。本系统中的用户名以及密码都存储在**passwd**文件中，不同的字段用**,**进行分割。因此，这部分涉及到文件的读取与写操作，其部分实现代码如下：

```

while (1) {
    if (feof(file)) {
        result = 0;
        break;
    }
    i = 0; j = 0;
    fgets(temp, 49, file);
    while (temp[i] != ',') {
        name[i] = temp[i];
        i++;
    }
    name[i] = '\0';
    if (strcmp(name, username) != 0) {
        continue;
    }
    else {
        for(i += 1; temp[i] != ','; j++, i++) {
            userpassword[j] = temp[i];
            userpassword[j] = '\0';
            if (strcmp(userpassword, password) != 0) {
                result = 0;
                break;
            }
            else {
                result = 1;
                break;
            }
        }
    }
}
return result;

```

3) 实现效果

```
Connected to localhost

Welcome!!! Please login or register first to use the FTP system.
*****
To login please press 'l' or 'L', to register please press 'r' or 'R'!
YOUR CHOICE: l
NAME: test
PASSWORD: test
invalid username/password. Please try again!
NAME: root
PASSWORD: 12
Login succeed!
client> █
```

2、注册

- **实现思路：**显然，为了实现权限管理我们必须实现登录功能，而实现了登录功能就必须实现注册功能。在我们使用 `sudo ./client localhost` 命令运行客户端程序后，系统会提示用户先进行登录或者是注册，没有注册过账户的用户可以输入 `r/R` 字符来选择注册功能。用户可以根据要求输入自己的用户名、密码，然后提交注册请求、等待管理员审批。在服务器端，管理员有权批准/拒绝用户的注册请求，如果申请被拒绝，那么该用户就无权进行任何操作，只能退出系统。若管理员批准该注册申请，那么管理员需要对用户的权限进行设置，分为 `HIGH`、`MID`、`LOW` 三个档次。在管理员设置成功后，用户便注册成功。然后用户便可以直接登录系统进行使用。
- **实现细节：**

1) 在客户端，首先需要用户输入自己将要注册的用户名。系统要求用户名必须小于20bytes，如果违反会被提醒重新输入。如果是合法输入，那么客户端就立即发送 `USER [input]` 指令到服务端。在服务端，服务器会在 `passwd` 文件中检索是否该名称已被注册，如果是，则提示客户端用户名已存在、需要选一个新的用户名，反之，则进入管理员审批环节。

由于这里的实现与登录功能过分接近XD，因此就不再贴出代码，只给出实现截图（客户端）：

```
Welcome!!! Please login or register first to use the FTP system.
*****
To login please press 'l' or 'L', to register please press 'r' or 'R'!
YOUR CHOICE: r
Please enter your name for the system less than 20 bytes:
NAME: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Illegal input!Please enter your name for the system less than 20 bytes:
NAME: root
Your name has existed! Please change one!
Please enter your name for the system less than 20 bytes:
NAME: mai
Your application has been sent, wait a moment for manager to check!
```

2) 由于我们假设：发出注册申请的用户使用的用户名都能让管理员唯一辨识自己的身份，因此，进入管理员审批环节后，管理员只需要用户名就可以根据进行注册审批、权限设置操作。其中，同意/拒绝通过输入 `y/n` 来标识，`HIGH`、`MID`、`LOW` 权限通过输入 `3`、`2`、`1` 来进行区分。部分代码如下：

```
int manager_check(int sock_fd, char* accountName, char *au){
    char getAnswer[5];
    printf("A user named %s just sent a register application!\n", accountName);
```

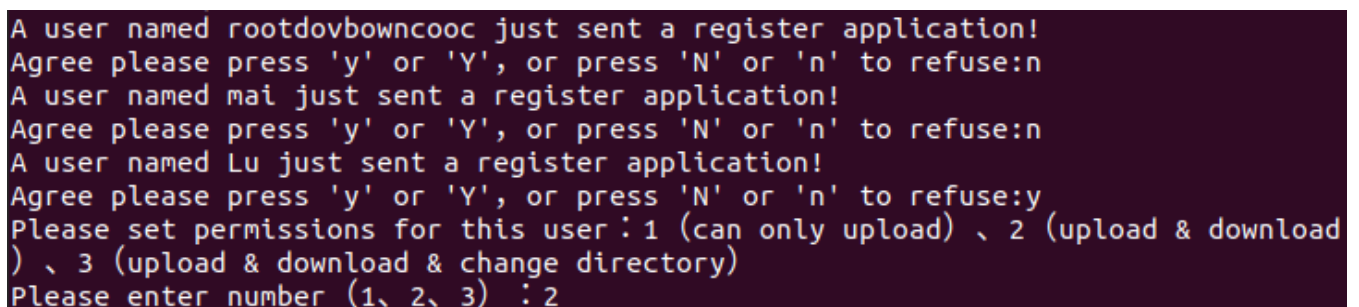


```

while (1) {
    printf("Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:");
    scanf("%s", getAnswer);
    if (getAnswer[0] == 'y' || getAnswer[0] == 'Y') { //如果同意，则进行权限设置操作
        printf("Please set permissions for this user: 1 (can only upload)、2 (upload &
download)、3 (upload & download & change directory) \n");
        while (1) {
            printf("Please enter number (1、2、3) : ");
            scanf("%s", getAnswer);
            if (getAnswer[0] == '1' || getAnswer[0] == '2' || getAnswer[0] == '3') {
                au[0]='\0';
                switch(getAnswer[0]){
                    case '3':
                        strcat(au,"HIGH");
                        break;
                    case '2':
                        strcat(au,"MID");
                        break;
                    case '1':
                        strcat(au,"LOW");
                        break;
                }
                break;
            }
            else {
                printf("Illegal input!\n");
            }
        }
        send_num(sock_fd, REGIST_APPLICATION_OK);
        return 1;
    }
    if (getAnswer[0] == 'n' || getAnswer[0] == 'N') { //如果不同意，则直接返回消息给客户端
        return 0;
    }
    else {
        printf("Illegal input!\n");
    }
}
}

```

实现截图如下（服务器端截图）：



```

A user named rootdovbowncooc just sent a register application!
Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:n
A user named mai just sent a register application!
Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:n
A user named Lu just sent a register application!
Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:y
Please set permissions for this user:1 (can only upload)、2 (upload & download
)、3 (upload & download & change directory)
Please enter number (1、2、3) :2

```

3) 如果管理员批准了注册，且设置好了权限，那么用户则可以开始进行密码的设置。其中，密码也要求不超过 20bytes，且需要2次确认密码。若第二次确认密码与原密码不一致，则会要求再一次确认.....直到输入的确认密码与原密码一致后，将密码提交到服务器端进行最后的.passwd文件写入，注册成功！

然后用户就可以使用刚刚注册的用户名、密码进行登录，其客户端实现效果如下所示：

```

Welcome!!! Please login or register first to use the FTP system.
*****
To login please press 'l' or 'L', to register please press 'r' or 'R'!
YOUR CHOICE: r
Please enter your name for the system less than 20 bytes:
NAME: Lu
Your application has been sent, wait a moment for manager to check!
Your application has passed!
Please enter your password less than 20 bytes.
Password:123
Please re-enter your password:12
Your password entered is different from the last time!
Please re-enter your password:123
Register successfully! Please login!
NAME: Lu
PASSWORD: 123
Login succeed!
client> █

```

3、权限管理

1) 文件访问权限管理。为了避免用户随意访问系统代码文件，我们将用户对服务器上的文件访问限制在了**user_dir**中。即，如果用户访问**user_dir**外的文件会被拒绝，对于用户而言**user_dir**就算根目录。

```

client> pwd
/user_dir
client> cd ..
already in the root dir!
client> pwd
/user_dir
client>

```

2) 指令使用权限

权限级别	可使用指令
HIGH	get、put、delete、mkdir、cd、pwd、ls、quit
MID	get、put、cd、pwd、ls、quit
LOW	cd、pwd、ls、quit

对于权限不够的用户访问高权限指令，系统会拒绝：

```

client> mkdir test
you have no right to MKDIR
client> delete readme
you have no right to DELETE
client>

```

4、cd命令格式补充

为了优化系统的可用性，我们补充了几种不同的cd命令格式，分别为 `cd`、`cd /`、`cd ~`、`cd ..`、`cd [相对路径]`、`cd [绝对路径]`。由于这个部分已经在[核心功能实现cd](#)中进行了说明，这里便不再赘述。

5、颜色显示

使用ls命令时，为了让用户便于区分文件与目录，目录一律显示为蓝色，文件一律显示为白色。此外，为了给用户更好的指令输入提示，每次输入指令前的 `client>` 字符显示为橙色，显示效果如下

```
client> ls
.
test.c
..
hello.md
testdir
client> 
```

五、异常处理方案

1、用户输入非法

本系统全局对用户的非法输入进行了检测，以防止用户、管理员进行非法输入。部分示例如下：

1) 输入了错误的指令，会提示指令错误：

```
client> wrongcommand
invalid command
client> mkdia
invalid command
client> lss
invalid command
client>
```

2) 在选择登录和注册的时候胡乱输入、提示重新选择:

```
Welcome!!! Please login or register first to use the FTP system.
*****
To login please press 'l' or 'L', to register please press 'r' or 'R'!
YOUR CHOICE: sdcsd
Sorry but you have pressed a wrong letter,please choose again!
*****
To login please press 'l' or 'L', to register please press 'r' or 'R'!
```

3) 注册的时候用户名没有按照规则输入、提示重新输入名字:

```
Welcome!!! Please login or register first to use the FTP system.  
*****  
To login please press 'l' or 'L', to register please press 'r' or 'R'!  
YOUR CHOICE: r  
Please enter your name for the system less than 20 bytes:  
NAME: kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk  
Illegal input!Please enter your name for the system less than 20 bytes:  
NAME:
```

4) 注册的时候用户名确认密码与上一次输入不一致, 提示重新确认:

```
Your application has passed !
Please enter your password less than 20 bytes.
Password:123
Please re-enter your password:123456
Your password entered is different from the last time !
Please re-enter your password:123
Register successfully ! Please login!
```

5) 在服务器端，管理员确认用户申请、设置权限的时候输入非法，提示重新选择：

```
A user named No just sent a register application!
Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:qecv
Illegal input!
Agree please press 'y' or 'Y', or press 'N' or 'n' to refuse:y
Please set permissions for this user:1 (can only upload) 、 2 (upload & download
) 、 3 (upload & download & change directory)
Please enter number (1、 2、 3) :kk
Illegal input!
Please enter number (1、 2、 3) :2
```

2、用户操作非法

1) 用户使用cd命令的时候不能切换出user_dir目录:

```
client> pwd
/user_dir
client> cd ..
already in the root dir!
client> pwd
/user_dir
client>
```

2) 用户使用cd、delete、put、get命令的时候, 文件/目录路径错误(只允许对当前目录下的文件进行操作、不能对父、子目录中的文件进行操作):

```
client> cd test/test
path:test/test not exist
client> delete test.cpp
file:test.cpp not exist
client> put test.png
File will be transfered in binary mode
open file error: No such file or directory
client> get test.html
file doesn't exist!
```

3) 使用delete删除目录、或使用put、get传输目录会被拒绝:

```
client> ls
test1.txt
.
423.jpg
test.c
..
testdir2
hello.md
test.txt
hhh
testdir
test.jpg
client> get hhh
Dirctory can not be transfered!
client> put testput
File will be transfered in binary mode
fstat error: Bad file descriptor
Dirctory can not be transfered!
client> delete hhh
you can't delete a dictory
```

4) 使用mkdir嵌套新建文件夹、mkdir新建文件夹名字重复会被拒绝:

```
client> ls
.
..
client> mkdir test
test creation succeed
client> mkdir test
test creation failed
client> mkdir test1/test
test1/test creation failed
```

