# Linux程序设计实验大作业实验报告

## 2017020800080 匡胤鑫

1、实验目的

实现一个FTP服务系统。

具体：基于客户/服务器交互模式，采用TCP/IP通信协议，实现以支持远程文件传输的客户/服务器软件，支持文件的上传和下载。其中客户端软件命名为myftp，服务器软件命名为ftpserver。

2、实验内容

服务器：

1. 启动方式：ftpserver [IP地址] port
2. 用户认证与权限管理
3. 并行执行功能
4. 服务传输模式
5. 服务器支持的客户文件操作：创建/删除目录（mkdir/rmdir）、显示当前路径（pwd）、切换目录（cd）、查看目录下的所有文件（ls）、上传单个/多个文件（put/mput）、下载单个/多个文件（get/mget）
6. 站点计数与用户管理功能：当前用户活动数（count current）、显示系统访客总数（count all）、显示当前在线的所有用户的用户名（list）、强制删除某个用户（kill username）、关闭ftp服务器软件（quit）
7. 其他功能：容错保护
8. 可选功能：文件缓冲区、断点续传

客户端：

1. 登录管理myftp username:passwd@ftp-server-ip:ftp-server-port→you are client #./username doesn’t exist or password is error!
2. 文件操作命令：创建/删除目录（lmkdir/lrmdir）、显示当前路径（lpwd）、切换目录（lcd）、查看当前目录下所有文件（dir）、上传单个/多个文件（put/mput）、下载单个/多个文件（get/mget）
3. 退出系统（quit）

3、实验过程

一、环境准备

实验采用Ubuntu环境开发，由于ftp用户的权限管理需要借助系统自身的权限机制，因此先要，在系统中利用shell脚本建立多名用户和组，相关建立和删除脚本如下：

user\_add.sh：

#!/bin/sh

i=1

groupadd class1

while [ ${i} -le 80 ]

do

if [ ${i} -le 9 ]; then

username=stu0${i}

else

username=stu${i}

fi

useradd -g class1 ${username}

mkdir /home/${username}

chown -R ${username} /home/${username}

chgrp class1 /home/${username}

i=$((${i}+1))

done

exit 0

user\_del.sh：

#!/bin/sh

i=1

while [ ${i} -le 80 ]

do

if [ ${i} -le 9 ]; then

username=stu0${i}

else

username=stu${i}

fi

rm -r /home/${username}

userdel ${username}

i=$((${i}+1))

done

groupdel class1

exit 0

另外在本实验中，权限的设计思路是任何用户可以在ftp根目录和自己创建的目录下上传文件，可以下载任何同组用户上传的文件。

一、框架搭建

思路：根据基本输入输出要求分别完成服务器端和客户端主程序设计，其中，服务器端和客户端都需要检查用户输入的是否为合法ip地址、是否为合法的端口号。以下为相关代码：

检测ip地址是否合法：

int is\_ip(char \*ip)

{

int flag = 0;

int temp[4] = { 0, 0, 0, 0 };

for (int i = 0; ip[i] != '\0' && i < 16; i++)

{

if (i == 15 || i == 0 && ip[i] == '.' || ip[i + 1] == '\0' && ip[i] == '.')

{

return 0;

}

if (ip[i] < '0' && ip[i] != '.' || ip[i] > '9')

{

return 0;

}

else

{

if (ip[i] != '.')

{

temp[flag] = (temp[flag] << 3) + (temp[flag] << 1) + ip[i] - '0';

if (temp[flag] > 255)

{

return 0;

}

}

else

{

flag++;

if (flag == 4 || ip[i - 1] == '.')

{

return 0;

}

}

}

}

if (flag < 3 || temp[0] == 0 || temp[0] > 223)

{

return 0;

}

else

{

if (temp[0] > 0 && temp[0] < 128)

{

if (temp[1] == 0 && temp[2] == 0 && temp[3] == 0 || temp[1] == 255 && temp[2] == 255 && temp[3] == 255)

{

return 0;

}

}

else if (temp[0] > 127 && temp[0] < 192)

{

if (temp[0] == 128 && temp[1] == 0 || temp[2] == 0 && temp[3] == 0 || temp[2] == 255 && temp[3] == 255)

{

return 0;

}

}

else

{

if (temp[0] == 192 && temp[1] == 0 && temp[2] == 0 || temp[3] == 0 || temp[3] == 255)

{

return 0;

}

}

}

return 1;

}

检测端口是否合法，以及将其转换为整形：

int get\_port(char \*port)

{

int temp = 0;

for (int i = 0; port[i] != '\0'; i++)

{

if (port[i] < '0' || port[i] > '9')

{

return -1;

}

temp = (temp << 3) + (temp << 1) + port[i] - '0';

}

if (temp < 0 || temp > 65535)

{

return -1;

}

return temp;

}

另外，在该软件中会多次用到命令输入，由于scanf有溢出的风险，可采用getchar进行有规则的输入，将读取函数单独封装，其功能为从下一个非空字符开始读取下一条字符串，直到回车或者EOF。返回值为最后一个读取的字符，超过事先规定的数组长时返回0。返回值用于上层调用函数判断读取内容是否符合格式要求，或者是否超出最大字符串数组长。具体实现：

char read\_next(char \*buffer)

{

char c;

for (int i = 0; i < STRING\_LENGTH + 1; i++)

{

if (i == STRING\_LENGTH)

{

printf("ERROR: Parameter must be less than %d charcaters!\n", STRING\_LENGTH - 1);

while ((c = getchar()) != '\n' && c != EOF);

c = 0;

break;

}

if (i == 0)

{

c = getchar();

while (c == ' ' || c == '\t' || c == '\n' || c == EOF)

{

c = getchar();

}

}

else

{

c = getchar();

}

if (c == '\n' || c == EOF)

{

buffer[i] = '\0';

break;

}

buffer[i] = c;

}

return c;

}

对于字符串数组长以及其它的一系列常数，应将它们单独定义：

#define STRING\_LENGTH 256

#define BUFFER\_SIZE 1024

#define BACKLOG 128

#define SUCCESS 1

#define FAILED 0

#define BINARY 0

#define ASCII 1

#define LOGIN 0

#define MKDIR 1

#define RMDIR 2

#define PWD 3

#define CD 4

#define LS 5

#define PUT 6

#define GET 7

#define DATA 8

#define END 9

#define QUIT 10

还有，我们也可以给传输加上crc32c的循环冗余码校验，以下是采用打表法实现的crc32c计算：

const unsigned int crc32c\_table[256] = {

0x00000000L, 0xF26B8303L, 0xE13B70F7L, 0x1350F3F4L,

0xC79A971FL, 0x35F1141CL, 0x26A1E7E8L, 0xD4CA64EBL,

0x8AD958CFL, 0x78B2DBCCL, 0x6BE22838L, 0x9989AB3BL,

0x4D43CFD0L, 0xBF284CD3L, 0xAC78BF27L, 0x5E133C24L,

0x105EC76FL, 0xE235446CL, 0xF165B798L, 0x030E349BL,

0xD7C45070L, 0x25AFD373L, 0x36FF2087L, 0xC494A384L,

0x9A879FA0L, 0x68EC1CA3L, 0x7BBCEF57L, 0x89D76C54L,

0x5D1D08BFL, 0xAF768BBCL, 0xBC267848L, 0x4E4DFB4BL,

0x20BD8EDEL, 0xD2D60DDDL, 0xC186FE29L, 0x33ED7D2AL,

0xE72719C1L, 0x154C9AC2L, 0x061C6936L, 0xF477EA35L,

0xAA64D611L, 0x580F5512L, 0x4B5FA6E6L, 0xB93425E5L,

0x6DFE410EL, 0x9F95C20DL, 0x8CC531F9L, 0x7EAEB2FAL,

0x30E349B1L, 0xC288CAB2L, 0xD1D83946L, 0x23B3BA45L,

0xF779DEAEL, 0x05125DADL, 0x1642AE59L, 0xE4292D5AL,

0xBA3A117EL, 0x4851927DL, 0x5B016189L, 0xA96AE28AL,

0x7DA08661L, 0x8FCB0562L, 0x9C9BF696L, 0x6EF07595L,

0x417B1DBCL, 0xB3109EBFL, 0xA0406D4BL, 0x522BEE48L,

0x86E18AA3L, 0x748A09A0L, 0x67DAFA54L, 0x95B17957L,

0xCBA24573L, 0x39C9C670L, 0x2A993584L, 0xD8F2B687L,

0x0C38D26CL, 0xFE53516FL, 0xED03A29BL, 0x1F682198L,

0x5125DAD3L, 0xA34E59D0L, 0xB01EAA24L, 0x42752927L,

0x96BF4DCCL, 0x64D4CECFL, 0x77843D3BL, 0x85EFBE38L,

0xDBFC821CL, 0x2997011FL, 0x3AC7F2EBL, 0xC8AC71E8L,

0x1C661503L, 0xEE0D9600L, 0xFD5D65F4L, 0x0F36E6F7L,

0x61C69362L, 0x93AD1061L, 0x80FDE395L, 0x72966096L,

0xA65C047DL, 0x5437877EL, 0x4767748AL, 0xB50CF789L,

0xEB1FCBADL, 0x197448AEL, 0x0A24BB5AL, 0xF84F3859L,

0x2C855CB2L, 0xDEEEDFB1L, 0xCDBE2C45L, 0x3FD5AF46L,

0x7198540DL, 0x83F3D70EL, 0x90A324FAL, 0x62C8A7F9L,

0xB602C312L, 0x44694011L, 0x5739B3E5L, 0xA55230E6L,

0xFB410CC2L, 0x092A8FC1L, 0x1A7A7C35L, 0xE811FF36L,

0x3CDB9BDDL, 0xCEB018DEL, 0xDDE0EB2AL, 0x2F8B6829L,

0x82F63B78L, 0x709DB87BL, 0x63CD4B8FL, 0x91A6C88CL,

0x456CAC67L, 0xB7072F64L, 0xA457DC90L, 0x563C5F93L,

0x082F63B7L, 0xFA44E0B4L, 0xE9141340L, 0x1B7F9043L,

0xCFB5F4A8L, 0x3DDE77ABL, 0x2E8E845FL, 0xDCE5075CL,

0x92A8FC17L, 0x60C37F14L, 0x73938CE0L, 0x81F80FE3L,

0x55326B08L, 0xA759E80BL, 0xB4091BFFL, 0x466298FCL,

0x1871A4D8L, 0xEA1A27DBL, 0xF94AD42FL, 0x0B21572CL,

0xDFEB33C7L, 0x2D80B0C4L, 0x3ED04330L, 0xCCBBC033L,

0xA24BB5A6L, 0x502036A5L, 0x4370C551L, 0xB11B4652L,

0x65D122B9L, 0x97BAA1BAL, 0x84EA524EL, 0x7681D14DL,

0x2892ED69L, 0xDAF96E6AL, 0xC9A99D9EL, 0x3BC21E9DL,

0xEF087A76L, 0x1D63F975L, 0x0E330A81L, 0xFC588982L,

0xB21572C9L, 0x407EF1CAL, 0x532E023EL, 0xA145813DL,

0x758FE5D6L, 0x87E466D5L, 0x94B49521L, 0x66DF1622L,

0x38CC2A06L, 0xCAA7A905L, 0xD9F75AF1L, 0x2B9CD9F2L,

0xFF56BD19L, 0x0D3D3E1AL, 0x1E6DCDEEL, 0xEC064EEDL,

0xC38D26C4L, 0x31E6A5C7L, 0x22B65633L, 0xD0DDD530L,

0x0417B1DBL, 0xF67C32D8L, 0xE52CC12CL, 0x1747422FL,

0x49547E0BL, 0xBB3FFD08L, 0xA86F0EFCL, 0x5A048DFFL,

0x8ECEE914L, 0x7CA56A17L, 0x6FF599E3L, 0x9D9E1AE0L,

0xD3D3E1ABL, 0x21B862A8L, 0x32E8915CL, 0xC083125FL,

0x144976B4L, 0xE622F5B7L, 0xF5720643L, 0x07198540L,

0x590AB964L, 0xAB613A67L, 0xB831C993L, 0x4A5A4A90L,

0x9E902E7BL, 0x6CFBAD78L, 0x7FAB5E8CL, 0x8DC0DD8FL,

0xE330A81AL, 0x115B2B19L, 0x020BD8EDL, 0xF0605BEEL,

0x24AA3F05L, 0xD6C1BC06L, 0xC5914FF2L, 0x37FACCF1L,

0x69E9F0D5L, 0x9B8273D6L, 0x88D28022L, 0x7AB90321L,

0xAE7367CAL, 0x5C18E4C9L, 0x4F48173DL, 0xBD23943EL,

0xF36E6F75L, 0x0105EC76L, 0x12551F82L, 0xE03E9C81L,

0x34F4F86AL, 0xC69F7B69L, 0xD5CF889DL, 0x27A40B9EL,

0x79B737BAL, 0x8BDCB4B9L, 0x988C474DL, 0x6AE7C44EL,

0xBE2DA0A5L, 0x4C4623A6L, 0x5F16D052L, 0xAD7D5351L

};

unsigned int get\_crc32c(unsigned char \*start, unsigned int size)

{

unsigned int crc32c = 0xFFFFFFFFL;

while (size--)

{

crc32c = crc32c\_table[(crc32c ^ \*start++) & 0xFFL] ^ (crc32c >> 8);

}

return crc32c ^ 0xFFFFFFFFL;

}

将以上内容打包封装，分别做成util.h头文件和uitl.c源文件，借助这些函数，从而实现服务器端和客户端的main函数：

服务器端：

int main(int argc, char \*argv[])

{

char \*ip = "127.0.0.1";

int port;

if (argc == 2)

{

if ((port = get\_port(argv[1])) == -1)

{

printf("ERROR: Invalid port input.\n");

return 0;

}

}

else if (argc == 3)

{

if (!is\_ip(ip = argv[1]))

{

printf("ERROR: Invalid ip input.\n");

return 0;

}

if ((port = get\_port(argv[2])) == -1)

{

printf("ERROR: Invalid port input.\n");

return 0;

}

}

else

{

printf("ERROR: Invalid parameter input. (ftpserver [ip] port)\n");

return 0;

}

ftpserver(ip, port);

return 0;

}

客户端：

int main(int argc, char \*argv[])

{

char username[STRING\_LENGTH], password[STRING\_LENGTH], ip[STRING\_LENGTH], port\_str[STRING\_LENGTH];

int port;

if (argc == 2)

{

int count, flag = 0, i = 0, j = 0, k = 0, l = 0;

for (count = 0; argv[1][count] != '\0'; count++)

{

if (flag == 0)

{

if (argv[1][count] != ':')

{

username[i++] = argv[1][count];

if (i == STRING\_LENGTH)

{

printf("ERROR: Username must be less than %d charcaters!\n", STRING\_LENGTH - 1);

return 0;

}

}

else

{

username[i] = '\0';

i = count;

flag++;

}

}

else if (flag == 1)

{

if (argv[1][count] != '@')

{

password[j++] = argv[1][count];

if (j == STRING\_LENGTH)

{

printf("ERROR: password must be less than %d charcaters!\n", STRING\_LENGTH - 1);

return 0;

}

}

else

{

password[j] = '\0';

j = count;

flag++;

}

}

else if (flag == 2)

{

if (argv[1][count] != ':')

{

ip[k++] = argv[1][count];

if (k == STRING\_LENGTH)

{

printf("ERROR: ip must be less than %d charcaters!\n", STRING\_LENGTH - 1);

return 0;

}

}

else

{

ip[k] = '\0';

k = count;

flag++;

}

}

else

{

port\_str[l++] = argv[1][count];

if (l == STRING\_LENGTH)

{

printf("ERROR: port must be less than %d charcaters!\n", STRING\_LENGTH - 1);

return 0;

}

if (argv[1][count + 1] == '\0')

{

port\_str[l] = '\0';

l = count + 1;

break;

}

}

}

if (i != 0 && i < j && j < k && k < l)

{

if (!is\_ip(ip))

{

printf("ERROR: Invalid ip input.\n");

return 0;

}

if ((port = get\_port(port\_str)) == -1)

{

printf("ERROR: Invalid port input.\n");

return 0;

}

}

else

{

printf("ERROR: Invalid parameter input. (myftp username:password@ip:port)\n");

return 0;

}

}

else

{

printf("ERROR: Invalid parameter input. (myftp username:password@ip:port)\n");

return 0;

}

myftp(username, password, ip, port);

return 0;

}

二、服务器端实现

在服务器端的main函数中，调用了ftpserver(ip, port)，此即为服务器函数的主体， 该函数的设计，首先我们要创建或读取ftp服务运行的文件区，然后根据所给ip和端口进行绑定，创建socket进行监听，同时需要开出两个线程分别处理服务器本身的命令操作、用户请求监听，还要为静态区的用户管理和文件缓冲区管理相关数据分配内存，实现代码如下：

oid ftpserver(char \*ip, int port)

{

if (access("/usr/local/share/ftpserver/ftp", F\_OK) == -1)

{

if (mkdir("/usr/local/share/ftpserver/ftp", S\_IRWXU | S\_IRWXG | S\_IRWXO) == -1)

{

printf("ERROR: Loading data failed!\n");

return;

}

if (chmod("/usr/local/share/ftpserver/ftp", S\_IRWXU | S\_IRWXG | S\_IRWXO) == -1)

{

printf("ERROR: Loading data failed!\n");

return;

}

//不知道为什么不加上这句会导致创建出来的文件权限不对

}

if (chmod("/usr/local/share/ftpserver/users", S\_IRUSR | S\_IWUSR | S\_IRGRP | S\_IROTH) == -1)

{

printf("ERROR: Loading data failed!\n");

return;

}

//初始化ftp数据

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

printf("ERROR: Creating socket failed!\n");

return;

}

int recv\_buf = 1024 \* 32;

setsockopt(sockfd, SOL\_SOCKET, SO\_RCVBUF, (const void\*)&recv\_buf, sizeof(int));

int send\_buf = 1024 \* 32;

setsockopt(sockfd, SOL\_SOCKET, SO\_SNDBUF, (const void\*)&send\_buf, sizeof(int));

//创建socket

struct sockaddr\_in local\_addr;

local\_addr.sin\_family = AF\_INET;

local\_addr.sin\_addr.s\_addr = inet\_addr(ip);

local\_addr.sin\_port = htons(port);

if (bind(sockfd, (struct sockaddr\*)&local\_addr, sizeof(struct sockaddr\_in)) == -1)

{

printf("ERROR: Binding service failed!\n");

return;

}

//绑定服务器

if (listen(sockfd, BACKLOG) == -1)

{

close(sockfd);

printf("ERROR: Listening request failed!\n");

return;

}

//设定监听

memset(users, '\0', sizeof(users));

memset(files, 0, sizeof(files));

if (chdir("/usr/local/share/ftpserver/ftp") == -1)

{

close(sockfd);

printf("ERROR: Loading data failed!\n");

return;

}

while (getcwd(work\_path, STRING\_LENGTH) == NULL);

if (pthread\_create(&tid\_handle, NULL, handle, (void\*)&sockfd) != 0)

{

close(sockfd);

printf("ERROR: Creating handle failed!\n");

return;

}

printf("ftpserver is running on %s:%d\n", ip, port);

if (pthread\_create(&tid\_task, NULL, task, (void\*)&sockfd) != 0)

{

close(sockfd);

printf("ERROR: Creating task failed!\n");

return;

}

void \*tret;

while (pthread\_join(tid\_task, &tret) != 0);

}

其中，task为服务器端的人机会话线程，主要用户输入命令和管理操作，handle为后台接收用户请求线程。对于task的设计，首先需要定义静态区可能会用到的各类变量和公共空间、线程互斥锁：

pthread\_t tid\_handle, tid\_task;

int all = 0, current = 0;

pthread\_mutex\_t current\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

char work\_path[STRING\_LENGTH];

pthread\_mutex\_t path\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

int sock\_users[BACKLOG];

pthread\_t tid\_users[BACKLOG];

char users[BACKLOG][STRING\_LENGTH];

struct file\_times

{

char exist;

ino\_t f\_ino;

time\_t f\_mtime;

int times;

};

struct file\_times files[BACKLOG];

char \*most = NULL;

int max;

pthread\_mutex\_t mem\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

然后，task需要完成的工作是，一个无限循环的命令监听，判断输入的命令以及对其进行识别和反馈。对于实验内容中的count、list命令，直接返回存放在静态区里的在有用户进出就会被修改的相关记录即可。对于kill和quit，则需要关闭对应用户的线程，或者进一步关闭handle线程，退出程序，以下为实现代码：

void \*task(void \*sockfd)

{

while (1)

{

printf("ftpserver> ");

char cmd[STRING\_LENGTH];

char c;

for (int i = 0; i < STRING\_LENGTH + 1; i++)

{

if (i == STRING\_LENGTH)

{

printf("ERROR: Command must be less than %d charcaters!\nftpserver> ", STRING\_LENGTH - 1);

i = -1;

while ((c = getchar()) != '\n' && c != EOF);

continue;

}

if (i == 0)

{

c = getchar();

while (c == ' ' || c == '\t' || c == '\n' || c == EOF)

{

c = getchar();

}

}

else

{

c = getchar();

}

if (c == ' ' || c == '\t' || c == '\n' || c == EOF)

{

cmd[i] = '\0';

break;

}

cmd[i] = c;

}

if (strcmp(cmd, "count") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

if (strcmp(pathname, "current") == 0)

{

printf("Number of currently active users: %d\n", current);

}

else if (strcmp(pathname, "all") == 0)

{

printf("Total system visitors: %d\n", all);

}

else

{

printf("ERROR: Invalid parameter input. (count current|all)\n");

}

}

}

else if (strcmp(cmd, "list") == 0)

{

if (c == '\n' || c == EOF)

{

for (int i = 0; i < BACKLOG; i++)

{

if (users[i][0] != '\0')

{

printf("%s\n", users[i]);

}

}

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (list)\n");

}

}

else if (strcmp(cmd, "kill") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

for (int i = 0; i < BACKLOG; i++)

{

if (strcmp(users[i], pathname) == 0)

{

void \*tret;

pthread\_cancel(tid\_users[i]);

if (pthread\_join(tid\_users[i], &tret) == 0)

{

close(sock\_users[i]);

pthread\_mutex\_lock(&current\_mutex);

current--;

pthread\_mutex\_unlock(&current\_mutex);

users[i][0] = '\0';

}

break;

}

if (i == BACKLOG - 1)

{

printf("ERROR: Username couldn't found!\n");

}

}

}

}

else if (strcmp(cmd, "quit") == 0)

{

if (c == '\n' || c == EOF)

{

if (pthread\_cancel(tid\_handle) != 0)

{

printf("ERROR: ftpserver quit failed!\n");

continue;

}

void \*tret;

while (pthread\_join(tid\_handle, &tret) != 0);

close(\*(int\*)sockfd);

for (int i = 0; i < BACKLOG; i++)

{

if (users[i][0] != '\0')

{

pthread\_cancel(tid\_users[i]);

if (pthread\_join(tid\_users[i], &tret) == 0)

{

close(sock\_users[i]);

}

}

}

if (most != NULL)

{

free(most);

}

pthread\_mutex\_destroy(&current\_mutex);

pthread\_mutex\_destroy(&path\_mutex);

pthread\_mutex\_destroy(&mem\_mutex);

break;

}

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (quit)\n");

}

else

{

if (c != '\n' && c != EOF)

{

while ((c = getchar()) != '\n' && c != EOF);

}

printf("ERROR: Command doesn't exist!\n");

}

}

}

再到对于handle线程的设计，只需不断监听客户端发来的用户请求并accept即可，另外注意适时开闭线程取消监听，保证一些关键操作的原子性：

void \*handle(void \*sockfd)

{

pthread\_setcanceltype(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

while (1)

{

pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL);

pthread\_testcancel();

struct sockaddr\_in remote\_addr;

int len = sizeof(struct sockaddr\_in);

int client\_sockfd = accept(\*(int\*)sockfd, (struct sockaddr\*)&remote\_addr, &len);

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, NULL);

if (current >= BACKLOG)

{

close(client\_sockfd);

continue;

}

if (client\_sockfd != -1)

{

for (int i = 0; i < BACKLOG; i++)

{

if (users[i][0] == '\0')

{

strcpy(users[i], "guest");

all++;

sock\_users[i] = client\_sockfd;

pthread\_create(&tid\_users[i], NULL, client, (void\*)&tid\_users[i]);

break;

}

}

}

}

}

在handle中，我们对于每一个用户发过的请求都创建了一个socket，并将其放入静态区，传入一个编号作为参数给一个单独的client线程进行处理。对于这个client线程，首先第一步就是定义数据包格式，在当前场景下，我们可以采用比较简单的应答模式，从而避免复杂的应用层协议设计。具体原理是，我们将每一次发送过来的数据作为一道命令，第一位标志命令的类型，随后是内容。对于client线程接收到的第一个数据包，必然是用户名和密码，在先前定义了参数LOGIN，用以判断数据包类型，即如果发来的数据包的第一位LOGIN，则这是一个登录请求，其他命令同理。随后是读取本地存放的users文件，查看是否存在符合的用户名和密码，此处users的作用相当于数据库，每一条记录的格式为username:password。若符合的记录存在，则在静态区记录登录信息，返回登录成功的标志，然后进入无限循环来监听用户需求，若不存在则返回失败的标志。

对于client的剩余部分，则是对发过来的命令进行处理和应答（利用定义好的参数匹配标志位判断时哪一种命令请求），首先对于监听循环，我们可以给recv函数设置超时限制，当响应超时的时候默认用户已经退出循环等待，结束线程，其次需要一个字符串保存工作路径位置，初值为ftp服务器文件区的根目录，因为一个进程仅有唯一的工作路径，所以这里用绝对路径的方式保存当前用户的工作目录。然后则是每种命令的处理：

* mkdir/rmdir：先判断权限，通过的话直接在当前目录创建文件。
* pwd：返回当前线程存储的工作路径减去服务器上文件区位置的路径，因为要对用户隐藏ftp服务运行的确切位置。
* cd：对chdir函数上锁，先将进程工作位置保存，重定向到保存的线程工作路径，在此基础上重定向到目标位置，判断权限以及是否可抵达，可抵达则将此时路径覆盖回保存线程工作路径的位置，最后恢复进程工作路径。
* ls：打开当前线程路径所代表的目录，遍历内容，返回结果。
* put/get：此处设计较为复杂，对于服务器而言，put/get对应的既可以是put/get请求，也可以是mput/mget请求，因此两边要保证数据包长度的统一，以便处理中间出错重发造成某一边一次接受多条指令的情况。对于binary模式和ascii模式，默认当ascii模式开启时，认定客户为Windows，将用户所传来的文件转换为Linux格式（在服务器端完成），将用户下载的文件转换成Windows格式（在客户端完成），转换工作由接收方完成，发送方永远按照原文件传输，以便接收方记录符合发送方位置的断点下次续传时告诉给发送方。对于断点续传，实现思路为在发生断开后（发送方发来FAILED信息，或者最后没有收到以END标识打头的数据包），则在本地建立一个名为原文件名.temp的文件，保存一个整型，记录断点位置，如果是ascii模式则再保存上一次传输的数据中最后传输的一位字符以便Linux/Windows之间对\n和\r\n的转换时的需要（转换需要确定出现的\n的上一位是否为\r）。对于文件缓冲区，实现思路为根据ino和mtime确定文件（结构体），统计所有get请求中出现的文件被下载的次数，为最高者malloc一块堆区内存，往后的get请求中先为所请求文件下载次数加1，识别所请求文件是否为最高文件，如果是，则直接从内存中读取数据发送，如果不是则采用传统办法发送。当有新的最高者出现时，free该内存并替换为新的最高者，当统计队列满时，自动去掉最早记录的一个下载量最少的文件，并在该位置记录新文件被下载的次数。
* quit：关闭socket、结束线程、删除存储在静态区的登录状态。

总体实现代码：

void \*client(void \*index)

{

pthread\_setcanceltype(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, NULL);

int i = (pthread\_t\*)index - tid\_users;

char buffer[BUFFER\_SIZE];

if (recv(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

pthread\_detach(pthread\_self());

close(sock\_users[i]);

users[i][0] = '\0';

return NULL;

}

if (buffer[0] != LOGIN)

{

pthread\_detach(pthread\_self());

close(sock\_users[i]);

users[i][0] = '\0';

return NULL;

}

char \*username = buffer + 1;

char \*password = buffer + STRING\_LENGTH + 1;

char identity[STRING\_LENGTH \* 2];

strcpy(identity, username);

strcat(identity, ":");

strcat(identity, password);

strcat(identity, "\n");

//获取登录信息

int ack = htonl(0);

FILE \*fp;

char login\_path[STRING\_LENGTH \* 2];

strcpy(login\_path, work\_path);

login\_path[strlen(login\_path) - 4] = '\0';

strcat(login\_path, "/users");

if ((fp = fopen(login\_path, "r")) == NULL)

{

pthread\_detach(pthread\_self());

close(sock\_users[i]);

users[i][0] = '\0';

return NULL;

}

char login[STRING\_LENGTH \* 2];

while (fgets(login, STRING\_LENGTH \* 2, fp) != NULL)

{

if (strcmp(identity, login) == 0)

{

ack = htonl(i + 1);

break;

}

}

fclose(fp);

if (ack == htonl(0))

{

pthread\_detach(pthread\_self());

send(sock\_users[i], (void\*)&ack, sizeof(int), 0);

close(sock\_users[i]);

users[i][0] = '\0';

return NULL;

}

if (send(sock\_users[i], (void\*)&ack, sizeof(int), 0) > 0)

{

strcpy(users[i], username);

pthread\_mutex\_lock(&current\_mutex);

current++;

pthread\_mutex\_unlock(&current\_mutex);

}

else

{

pthread\_detach(pthread\_self());

send(sock\_users[i], (void\*)&ack, sizeof(int), 0);

close(sock\_users[i]);

users[i][0] = '\0';

return NULL;

}

//验证登录信息

char current\_path[STRING\_LENGTH];

strcpy(current\_path, work\_path);

struct timeval tv\_out;

tv\_out.tv\_sec = 1200;

tv\_out.tv\_usec = 0;

setsockopt(sock\_users[i], SOL\_SOCKET, SO\_RCVTIMEO, &tv\_out, sizeof(tv\_out));

while (1)

{

pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL);

pthread\_testcancel();

if (recv(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0) > 0)

{

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, NULL);

if (buffer[0] == MKDIR)

{

char \*result = buffer + 1;

if (strchr(result, '/') != NULL || strcmp(result, ".") == 0 || strcmp(result, "..") == 0)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: Invalid folder name!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

struct passwd \*pw;

struct stat buf;

if ((pw = getpwnam(users[i])) == NULL || stat(current\_path, &buf) == -1)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver mkdir failed!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

if (buf.st\_uid == pw->pw\_uid || strcmp(current\_path, work\_path) == 0)

{

char temp[STRING\_LENGTH];

strcpy(temp, current\_path);

strcat(temp, "/");

strcat(temp, result);

if (mkdir(temp, S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH | S\_IXOTH) == -1)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver mkdir failed!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

while (chown(temp, pw->pw\_uid, pw->pw\_gid) == -1);

buffer[0] = SUCCESS;

send(sock\_users[i], (void\*)&buffer, 1, 0);

}

else

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver mkdir permission denied!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

}

else if (buffer[0] == RMDIR)

{

char \*result = buffer + 1;

if (strchr(result, '/') != NULL || strcmp(result, ".") == 0 || strcmp(result, "..") == 0)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: Invalid folder name!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

char temp[STRING\_LENGTH];

strcpy(temp, current\_path);

strcat(temp, "/");

strcat(temp, result);

struct passwd \*pw;

struct stat buf;

if ((pw = getpwnam(users[i])) == NULL || stat(temp, &buf) == -1)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver rmdir failed!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

if (buf.st\_uid == pw->pw\_uid)

{

if (rmdir(temp) == -1)

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver rmdir failed!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

buffer[0] = SUCCESS;

send(sock\_users[i], (void\*)&buffer, 1, 0);

}

else

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: ftpserver rmdir permission denied!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

}

else if (buffer[0] == PWD)

{

buffer[0] = SUCCESS;

char \*result = buffer + 1;

strcpy(result, current\_path + strlen(work\_path) - 4);

strcat(result, "\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

else if (buffer[0] == CD)

{

char \*result = buffer + 1;

char last\_path[STRING\_LENGTH];

pthread\_mutex\_lock(&path\_mutex);

if (getcwd(last\_path, STRING\_LENGTH) == NULL)

{

pthread\_mutex\_unlock(&path\_mutex);

buffer[0] = FAILED;

strcpy(result, "ftpserver: Path cannot be reached!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

if (chdir(current\_path) == -1)

{

pthread\_mutex\_unlock(&path\_mutex);

buffer[0] = FAILED;

strcpy(result, "ftpserver: Path cannot be reached!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

continue;

}

if (chdir(result) == 0)

{

char temp[STRING\_LENGTH];

while (getcwd(temp, STRING\_LENGTH) == NULL);

if (strncmp(temp, work\_path, strlen(work\_path)) == 0)

{

buffer[0] = SUCCESS;

strcpy(current\_path, temp);

send(sock\_users[i], (void\*)&buffer, 1, 0);

}

else

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: Path cannot be reached!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

}

else

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: Path cannot be reached!\n");

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

while (chdir(last\_path) == -1);

pthread\_mutex\_unlock(&path\_mutex);

}

else if (buffer[0] == LS)

{

char \*result = buffer + 1;

DIR \*dp;

if ((dp = opendir(current\_path)) != NULL)

{

buffer[0] = SUCCESS;

strcpy(result, "");

struct dirent \*entry;

while ((entry = readdir(dp)) != NULL)

{

if (strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, ".."))

{

strcat(result, entry->d\_name);

strcat(result, "\n");

}

}

closedir(dp);

}

else

{

buffer[0] = FAILED;

strcpy(result, "ftpserver: Directory open failed!\n");

}

send(sock\_users[i], (void\*)&buffer, strlen(result) + 2, 0);

}

else if (buffer[0] == PUT)

{

char \*result = buffer + 1;

char mode = \*result;

if (strchr(result + 1, '/') != NULL || strcmp(result + 1, ".") == 0 || strcmp(result + 1, "..") == 0)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

struct passwd \*pw;

struct stat buf;

if ((pw = getpwnam(users[i])) == NULL || stat(current\_path, &buf) == -1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

if (buf.st\_uid == pw->pw\_uid || strcmp(current\_path, work\_path) == 0)

{

char temp\_name[STRING\_LENGTH \* 2];

strcpy(temp\_name, current\_path);

strcat(temp\_name, "/");

strcat(temp\_name, result + 1);

strcat(temp\_name, ".temp");

int breakpoint = 0;

char last = '\0';

if (access(temp\_name, R\_OK) == 0)

{

FILE \*fp;

if ((fp = fopen(temp\_name, "r")) == NULL)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

while (fread(&breakpoint, sizeof(int), 1, fp) != 1)

{

fseek(fp, 0, SEEK\_SET);

}

if (mode == ASCII && breakpoint > 0)

{

last = fgetc(fp);

}

fclose(fp);

}

FILE \*fp;

temp\_name[strlen(temp\_name) - 5] = '\0';

if (breakpoint == 0 ? ((fp = fopen(temp\_name, "wb+")) == NULL) : ((fp = fopen(temp\_name, "ab+")) == NULL))

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

buffer[0] = SUCCESS;

int \*temp = (int\*)result;

\*temp = htonl(breakpoint);

if (send(sock\_users[i], (void\*)&buffer, sizeof(int) + 1, 0) < 1)

{

fclose(fp);

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

breakpoint = 0;

int ret = recv(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0);

while (ret > 0)

{

if (buffer[0] == DATA)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp);

}

else

{

for (int i = sizeof(int) + 1; i < BUFFER\_SIZE; i++)

{

if (last == '\r' && buffer[i] != '\n')

{

fputc('\r', fp);

}

if (buffer[i] != '\r')

{

fputc(buffer[i], fp);

}

last = buffer[i];

}

}

breakpoint += BUFFER\_SIZE - sizeof(int) - 1;

}

else if (buffer[0] == END)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), ntohl(\*temp), fp);

}

else

{

for (int i = sizeof(int) + 1; i < ntohl(\*temp) + sizeof(int) + 1; i++)

{

if (last == '\r' && buffer[i] != '\n')

{

fputc('\r', fp);

}

if (buffer[i] != '\r')

{

fputc(buffer[i], fp);

}

last = buffer[i];

}

}

temp\_name[strlen(temp\_name)] = '.';

if (access(temp\_name, F\_OK) == 0)

{

remove(temp\_name);

}

temp\_name[strlen(temp\_name) - 5] = '\0';

break;

}

else

{

break;

}

ret = recv(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0);

}

if (buffer[0] != DATA && buffer[0] != END || ret < 1)

{

temp\_name[strlen(temp\_name)] = '.';

FILE \*tfp;

if ((tfp = fopen(temp\_name, "w")) != NULL)

{

fwrite(&breakpoint, sizeof(int), 1, tfp);

if (mode == ASCII)

{

fputc(last, tfp);

}

fclose(tfp);

}

fclose(fp);

continue;

}

fclose(fp);

while (chmod(temp\_name, S\_IRUSR | S\_IWUSR | S\_IRGRP) == -1);

while (chown(temp\_name, pw->pw\_uid, pw->pw\_gid) == -1);

}

else

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

}

}

else if (buffer[0] == GET)

{

char \*result = buffer + 1;

if (strchr(result, '/') != NULL || strcmp(result, ".") == 0 || strcmp(result, "..") == 0)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

char filename[STRING\_LENGTH];

strcpy(filename, current\_path);

strcat(filename, "/");

strcat(filename, result);

if (access(filename, R\_OK) == -1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

struct passwd \*pw;

struct stat buf;

if ((pw = getpwnam(users[i])) == NULL || stat(filename, &buf) == -1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

if (buf.st\_gid == pw->pw\_gid)

{

buffer[0] = SUCCESS;

if (send(sock\_users[i], (void\*)&buffer, 1, 0) < 1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

if (recv(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

if (buffer[0] != SUCCESS)

{

continue;

}

int \*temp = (int\*)result;

int breakpoint = ntohl(\*temp);

FILE \*fp;

if ((fp = fopen(filename, "rb")) == NULL)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

pthread\_mutex\_lock(&mem\_mutex);

int fi;

for (fi = 0; fi < BACKLOG; fi++)

{

if (files[fi].exist == 1)

{

if (files[fi].f\_ino == buf.st\_ino && files[fi].f\_mtime == buf.st\_mtime)

{

files[fi].times++;

break;

}

}

else

{

files[fi].exist = 1;

files[fi].f\_ino = buf.st\_ino;

files[fi].f\_mtime = buf.st\_mtime;

files[fi].times = 1;

break;

}

}

if (most == NULL || fi < BACKLOG && files[fi].times > files[max].times)

{

if (most != NULL)

{

free(most);

}

max = i;

if (buf.st\_size > 0)

{

most = malloc(buf.st\_size);

while (fread(most, sizeof(char), buf.st\_size, fp) < 1)

{

fseek(fp, 0, SEEK\_SET);

}

}

else

{

most = malloc(1);

\*most = 0;

}

}

if (fi >= BACKLOG)

{

int min = 0;

for (fi = 0; fi < BACKLOG; fi++)

{

min = files[fi].times < files[min].times ? fi : min;

}

files[min].f\_ino = buf.st\_ino;

files[min].f\_mtime = buf.st\_mtime;

files[min].times = 1;

}

fseek(fp, breakpoint, SEEK\_SET);

buffer[0] = DATA;

int size;

if (most != NULL && buf.st\_ino == files[max].f\_ino && buf.st\_mtime == files[max].f\_mtime)

{

fclose(fp);

int finished = breakpoint, unfinished = buf.st\_size - breakpoint;

while (unfinished >= BUFFER\_SIZE - sizeof(int) - 1)

{

memcpy(buffer + sizeof(int) + 1, most + finished, BUFFER\_SIZE - sizeof(int) - 1);

finished += BUFFER\_SIZE - sizeof(int) - 1;

unfinished -= BUFFER\_SIZE - sizeof(int) - 1;

if (send(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0) < 1)//此处可以测试断点

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

break;

}

}

if (buffer[0] != FAILED)

{

buffer[0] = END;

\*temp = htonl(unfinished);

memcpy(buffer + sizeof(int) + 1, most + finished, unfinished);

if (send(sock\_users[i], (void\*)&buffer, unfinished + sizeof(int) + 1, 0) >= 1)

{

pthread\_mutex\_unlock(&mem\_mutex);

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

}

pthread\_mutex\_unlock(&mem\_mutex);

}

else

{

pthread\_mutex\_unlock(&mem\_mutex);

while ((size = fread(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp)) == BUFFER\_SIZE - sizeof(int) - 1)

{

if (send(sock\_users[i], (void\*)&buffer, BUFFER\_SIZE, 0) < 1)//此处可以测试断点

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

break;

}

}

fclose(fp);

if (buffer[0] != FAILED)

{

buffer[0] = END;

\*temp = htonl(size);

if (send(sock\_users[i], (void\*)&buffer, size + sizeof(int) + 1, 0) < 1)

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

continue;

}

}

}

}

else

{

buffer[0] = FAILED;

send(sock\_users[i], (void\*)&buffer, 1, 0);

}

}

else if (buffer[0] == QUIT)

{

pthread\_detach(pthread\_self());

close(sock\_users[i]);

pthread\_mutex\_lock(&current\_mutex);

current--;

pthread\_mutex\_unlock(&current\_mutex);

users[i][0] = '\0';

break;

}

}

else

{

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, NULL);

pthread\_detach(pthread\_self());

close(sock\_users[i]);

pthread\_mutex\_lock(&current\_mutex);

current--;

pthread\_mutex\_unlock(&current\_mutex);

users[i][0] = '\0';

break;

}

}

}

void \*handle(void \*sockfd)

{

pthread\_setcanceltype(PTHREAD\_CANCEL\_ASYNCHRONOUS, NULL);

while (1)

{

pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL);

pthread\_testcancel();

struct sockaddr\_in remote\_addr;

int len = sizeof(struct sockaddr\_in);

int client\_sockfd = accept(\*(int\*)sockfd, (struct sockaddr\*)&remote\_addr, &len);

pthread\_setcancelstate(PTHREAD\_CANCEL\_DISABLE, NULL);

if (current >= BACKLOG)

{

close(client\_sockfd);

continue;

}

if (client\_sockfd != -1)

{

for (int i = 0; i < BACKLOG; i++)

{

if (users[i][0] == '\0')

{

strcpy(users[i], "guest");

all++;

sock\_users[i] = client\_sockfd;

pthread\_create(&tid\_users[i], NULL, client, (void\*)&tid\_users[i]);

break;

}

}

}

}

}

最后，将以上内容封装为ftpserver.h和ftpserver.c。

三、客户端实现

在客户端的main函数中，调用了客户端函数的主体myftp(username, password, ip, port)。客户端函数的内容首先为向目标服务器发送用户名密码并取回结果，成功后开始命令监听，对于要在服务器端处理的命令，只需要向服务器发送对应的定义好的参数标志位和命令内容然后回显结果即可，get/mget和put/mput则是服务器端put、get的翻版，原理一致。以下是部分与服务器实现不同的本地命令的处理思路：

* binary/ascii：直接修改放会被用在传输时用到的相关变量，对应处理由负责传输的部分完成。
* quit：向服务器发送消息，然后直接退出。

总体实现代码：

void myftp(char \*username, char \*password, char \*ip, int port)

{

int sockfd;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1)

{

printf("ERROR: Creating socket failed!\n");

return;

}

int recv\_buf = 1024 \* 32;

setsockopt(sockfd, SOL\_SOCKET, SO\_RCVBUF, (const void\*)&recv\_buf, sizeof(int));

int send\_buf = 1024 \* 32;

setsockopt(sockfd, SOL\_SOCKET, SO\_SNDBUF, (const void\*)&send\_buf, sizeof(int));

//创建socket

struct sockaddr\_in dest\_addr;

dest\_addr.sin\_family = AF\_INET;

dest\_addr.sin\_addr.s\_addr = inet\_addr(ip);

dest\_addr.sin\_port = htons(port);

if (connect(sockfd, (struct sockaddr\*)&dest\_addr, sizeof(struct sockaddr\_in)) == -1)

{

printf("ERROR: Creating connection failed!\n");

return;

}

//连接服务器

char login[STRING\_LENGTH \* 2 + 1];

login[0] = LOGIN;

strcpy(login + 1, username);

strcpy(login + STRING\_LENGTH + 1, password);

if (send(sockfd, (void\*)&login, sizeof(login), 0) < 1)

{

close(sockfd);

printf("ERROR: myftp login failed!\n");

return;

}

int ack;

if (recv(sockfd, (void\*)&ack, sizeof(int), 0) < 1)

{

close(sockfd);

printf("ERROR: myftp login failed!\n");

return;

}

//发送用户名和密码

if ((ack = ntohl(ack)) > 0)

{

printf("You are client %d.\n", ack);

}

else

{

printf("Username doesn't exist or password is error!\n");

return;

}

while (chdir(getenv("HOME")) == -1);

char mode = BINARY;

char buffer[BUFFER\_SIZE];

while (1)

{

printf("myftp> ");

char cmd[STRING\_LENGTH];

char c;

for (int i = 0; i < STRING\_LENGTH + 1; i++)

{

if (i == STRING\_LENGTH)

{

printf("ERROR: Command must be less than %d charcaters!\nmyftp> ", STRING\_LENGTH - 1);

i = -1;

while ((c = getchar()) != '\n' && c != EOF);

continue;

}

if (i == 0)

{

c = getchar();

while (c == ' ' || c == '\t' || c == '\n' || c == EOF)

{

c = getchar();

}

}

else

{

c = getchar();

}

if (c == ' ' || c == '\t' || c == '\n' || c == EOF)

{

cmd[i] = '\0';

break;

}

cmd[i] = c;

}

if (strcmp(cmd, "binary") == 0)

{

if (c == '\n' || c == EOF)

{

mode = BINARY;

printf("The transmission mode is \"binary\" now.\n");

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (binary)\n");

}

}

else if (strcmp(cmd, "ascii") == 0)

{

if (c == '\n' || c == EOF)

{

mode = ASCII;

printf("The transmission mode is \"ascii\" now.\n");

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (ascii)\n");

}

}

else if (strcmp(cmd, "mkdir") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

buffer[0] = MKDIR;

strcpy(buffer + 1, pathname);

if (send(sockfd, (void\*)&buffer, strlen(pathname) + 2, 0) < 1)

{

printf("ERROR: myftp mkdir failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp mkdir failed!\n");

continue;

}

if (buffer[0] != SUCCESS)

{

printf("%s", buffer + 1);

}

}

}

else if (strcmp(cmd, "rmdir") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

buffer[0] = RMDIR;

strcpy(buffer + 1, pathname);

if (send(sockfd, (void\*)&buffer, strlen(pathname) + 2, 0) < 1)

{

printf("ERROR: myftp rmdir failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp rmdir failed!\n");

continue;

}

if (buffer[0] != SUCCESS)

{

printf("%s", buffer + 1);

}

}

}

else if (strcmp(cmd, "pwd") == 0)

{

if (c == '\n' || c == EOF)

{

buffer[0] = PWD;

if (send(sockfd, (void\*)&buffer, 1, 0) < 1)

{

printf("ERROR: myftp pwd failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp pwd failed!\n");

continue;

}

printf("%s", buffer + 1);

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (pwd)\n");

}

}

else if (strcmp(cmd, "cd") == 0)

{

char path[STRING\_LENGTH];

char c = read\_next(path);

if (c != 0)

{

buffer[0] = CD;

strcpy(buffer + 1, path);

if (send(sockfd, (void\*)&buffer, strlen(path) + 2, 0) < 1)

{

printf("ERROR: myftp cd failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp cd failed!\n");

continue;

}

if (buffer[0] != SUCCESS)

{

printf("%s", buffer + 1);

}

}

}

else if (strcmp(cmd, "ls") == 0)

{

if (c == '\n' || c == EOF)

{

buffer[0] = LS;

if (send(sockfd, (void\*)&buffer, 1, 0) < 1)

{

printf("ERROR: myftp ls failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp ls failed!\n");

continue;

}

printf("%s", buffer + 1);

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (ls)\n");

}

}

else if (strcmp(cmd, "lmkdir") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

if (mkdir(pathname, S\_IRWXU | S\_IRGRP | S\_IXGRP | S\_IROTH | S\_IXOTH) == -1)

{

printf("ERROR: myftp lmkdir failed!\n");

}

}

}

else if (strcmp(cmd, "lrmdir") == 0)

{

char pathname[STRING\_LENGTH];

char c = read\_next(pathname);

if (c != 0)

{

if (rmdir(pathname) == -1)

{

printf("ERROR: myftp lrmdir failed!\n");

}

}

}

else if (strcmp(cmd, "lpwd") == 0)

{

if (c == '\n' || c == EOF)

{

char \*path;

if ((path = getcwd(NULL, 0)) != NULL)

{

printf("%s\n", path);

free(path);

path = NULL;

}

else

{

printf("ERROR: myftp lpwd failed!\n");

}

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (lpwd)\n");

}

}

else if (strcmp(cmd, "lcd") == 0)

{

char path[STRING\_LENGTH];

char c = read\_next(path);

if (c != 0)

{

if (chdir(path) == -1)

{

printf("ERROR: myftp lcd failed!\n");

}

}

}

else if (strcmp(cmd, "dir") == 0)

{

if (c == '\n' || c == EOF)

{

DIR \*dp;

if ((dp = opendir(".")) != NULL)

{

struct dirent \*entry;

while ((entry = readdir(dp)) != NULL)

{

if (strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, ".."))

{

printf("%s\n", entry->d\_name);

}

}

closedir(dp);

}

else

{

printf("ERROR: myftp dir failed!\n");

}

}

else

{

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (dir)\n");

}

}

else if (strcmp(cmd, "put") == 0)

{

char filename[STRING\_LENGTH];

char c = read\_next(filename);

if (c != 0)

{

if (strchr(filename, '/') != NULL || strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0 || access(filename, R\_OK) == -1)

{

printf("ERROR: myftp put failed!\n");

continue;

}

buffer[0] = PUT;

buffer[1] = mode;

char \*result = buffer + 1;

strcpy(result + 1, filename);

if (send(sockfd, (void\*)&buffer, strlen(result + 1) + 3, 0) < 1)

{

printf("ERROR: myftp put failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp put failed!\n");

continue;

}

if (buffer[0] != SUCCESS)

{

printf("ftpserver: put failed!\n");

continue;

}

int \*temp = (int\*)result;

int breakpoint = ntohl(\*temp);

FILE \*fp;

if ((fp = fopen(filename, "rb")) == NULL)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("ERROR: myftp put failed!\n");

continue;

}

fseek(fp, breakpoint, SEEK\_SET);

buffer[0] = DATA;

int size;

while ((size = fread(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp)) == BUFFER\_SIZE - sizeof(int) - 1)

{

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)//此处可以测试断点

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("ERROR: myftp put failed!\n");

break;

}

}

fclose(fp);

if (buffer[0] != FAILED)

{

buffer[0] = END;

\*temp = htonl(size);

if (send(sockfd, (void\*)&buffer, size + sizeof(int) + 1, 0) < 1)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("gtpserver: put failed!\n");

continue;

}

printf("myftp: 1 file has been put on.\n");

}

}

}

else if (strcmp(cmd, "mput") == 0)

{

char filenames[STRING\_LENGTH];

char filename[STRING\_LENGTH];

char c = read\_next(filenames);

if (c != 0)

{

int i = 0, s = 0, success\_count = 0, failed\_count = 0;

while (filenames[i] != '\0')

{

while (filenames[i] != ' ' && filenames[i] != '\t' && filenames[i] != '\0')

{

i++;

}

strncpy(filename, filenames + s, i - s);

filename[i - s] = '\0';

s = i;

while (filenames[s] == ' ' || filenames[s] == '\t')

{

s++;

}

i = s;

if (strchr(filename, '/') != NULL || strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0 || access(filename, R\_OK) == -1)

{

failed\_count++;

continue;

}

buffer[0] = PUT;

buffer[1] = mode;

char \*result = buffer + 1;

strcpy(result + 1, filename);

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

failed\_count++;

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

failed\_count++;

continue;

}

if (buffer[0] != SUCCESS)

{

failed\_count++;

continue;

}

int \*temp = (int\*)result;

int breakpoint = ntohl(\*temp);

FILE \*fp;

if ((fp = fopen(filename, "rb")) == NULL)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

continue;

}

fseek(fp, breakpoint, SEEK\_SET);

buffer[0] = DATA;

int size;

while ((size = fread(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp)) == BUFFER\_SIZE - sizeof(int) - 1)

{

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)//此处可以测试断点

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

success\_count--;

break;

}

}

fclose(fp);

if (buffer[0] != FAILED)

{

buffer[0] = END;

\*temp = htonl(size);

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

continue;

}

}

success\_count++;

}

printf("myftp: %d file has been put on. %d failed.\n", success\_count, failed\_count);

}

}

else if (strcmp(cmd, "get") == 0)

{

char filename[STRING\_LENGTH];

char c = read\_next(filename);

if (c != 0)

{

buffer[0] = GET;

char \*result = buffer + 1;

strcpy(result, filename);

if (send(sockfd, (void\*)&buffer, strlen(result) + 2, 0) < 1)

{

printf("ERROR: myftp get failed!\n");

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

printf("ERROR: myftp get failed!\n");

continue;

}

if (buffer[0] != SUCCESS)

{

printf("ftpserver: get failed!\n");

continue;

}

strcat(filename, ".temp");

int breakpoint = 0;

char last = '\0';

if (access(filename, R\_OK) == 0)

{

FILE \*fp;

if ((fp = fopen(filename, "r")) == NULL)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("ERROR: myftp get failed!\n");

continue;

}

while (fread(&breakpoint, sizeof(int), 1, fp) != 1)

{

fseek(fp, 0, SEEK\_SET);

}

if (mode == ASCII && breakpoint > 0)

{

last = fgetc(fp);

}

fclose(fp);

}

FILE \*fp;

filename[strlen(filename) - 5] = '\0';

if (breakpoint == 0 ? ((fp = fopen(filename, "wb+")) == NULL) : ((fp = fopen(filename, "ab+")) == NULL))

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("ERROR: myftp get failed!\n");

continue;

}

buffer[0] = SUCCESS;

int \*temp = (int\*)result;

\*temp = htonl(breakpoint);

if (send(sockfd, (void\*)&buffer, sizeof(int) + 1, 0) < 1)

{

fclose(fp);

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, 1, 0);

printf("ERROR: myftp get failed!\n");

continue;

}

breakpoint = 0;

int ret = recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

while (ret > 0)

{

if (buffer[0] == DATA)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp);

}

else

{

for (int i = sizeof(int) + 1; i < BUFFER\_SIZE; i++)

{

if (last != '\r' && buffer[i] == '\n')

{

fputc('\r', fp);

}

fputc(buffer[i], fp);

last = buffer[i];

}

}

breakpoint += BUFFER\_SIZE - sizeof(int) - 1;

}

else if (buffer[0] == END)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), ntohl(\*temp), fp);

}

else

{

for (int i = sizeof(int) + 1; i < ntohl(\*temp) + sizeof(int) + 1; i++)

{

if (last != '\r' && buffer[i] == '\n')

{

fputc('\r', fp);

}

fputc(buffer[i], fp);

last = buffer[i];

}

}

filename[strlen(filename)] = '.';

if (access(filename, F\_OK) == 0)

{

remove(filename);

}

filename[strlen(filename) - 5] = '\0';

break;

}

else

{

break;

}

ret = recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

}

if (buffer[0] != DATA && buffer[0] != END || ret < 1)

{

filename[strlen(filename)] = '.';

FILE \*tfp;

if ((tfp = fopen(filename, "w")) != NULL)

{

fwrite(&breakpoint, sizeof(int), 1, tfp);

if (mode == ASCII)

{

fputc(last, tfp);

}

fclose(tfp);

}

fclose(fp);

printf("ftpserver: get failed!\n");

continue;

}

fclose(fp);

if (buffer[0] != FAILED)

{

printf("myftp: 1 file has been get down.\n");

}

}

}

else if (strcmp(cmd, "mget") == 0)

{

char filenames[STRING\_LENGTH];

char filename[STRING\_LENGTH];

char c = read\_next(filenames);

if (c != 0)

{

int i = 0, s = 0, success\_count = 0, failed\_count = 0;

while (filenames[i] != '\0')

{

while (filenames[i] != ' ' && filenames[i] != '\t' && filenames[i] != '\0')

{

i++;

}

strncpy(filename, filenames + s, i - s);

filename[i - s] = '\0';

s = i;

while (filenames[s] == ' ' || filenames[s] == '\t')

{

s++;

}

i = s;

buffer[0] = GET;

char \*result = buffer + 1;

strcpy(result, filename);

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

failed\_count++;

continue;

}

if (recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

failed\_count++;

continue;

}

if (buffer[0] != SUCCESS)

{

failed\_count++;

continue;

}

strcat(filename, ".temp");

int breakpoint = 0;

char last = '\0';

if (access(filename, R\_OK) == 0)

{

FILE \*fp;

if ((fp = fopen(filename, "r")) == NULL)

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

continue;

}

while (fread(&breakpoint, sizeof(int), 1, fp) != 1)

{

fseek(fp, 0, SEEK\_SET);

}

if (mode == ASCII && breakpoint > 0)

{

last = fgetc(fp);

}

fclose(fp);

}

FILE \*fp;

filename[strlen(filename) - 5] = '\0';

if (breakpoint == 0 ? ((fp = fopen(filename, "wb+")) == NULL) : ((fp = fopen(filename, "ab+")) == NULL))

{

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

continue;

}

buffer[0] = SUCCESS;

int \*temp = (int\*)result;

\*temp = htonl(breakpoint);

if (send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0) < 1)

{

fclose(fp);

buffer[0] = FAILED;

send(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

failed\_count++;

continue;

}

breakpoint = 0;

int ret = recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

while (ret > 0)

{

if (buffer[0] == DATA)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), BUFFER\_SIZE - sizeof(int) - 1, fp);

}

else

{

for (int i = sizeof(int) + 1; i < BUFFER\_SIZE; i++)

{

if (last != '\r' && buffer[i] == '\n')

{

fputc('\r', fp);

}

fputc(buffer[i], fp);

last = buffer[i];

}

}

breakpoint += BUFFER\_SIZE - sizeof(int) - 1;

}

else if (buffer[0] == END)

{

if (mode == BINARY)

{

fwrite(buffer + sizeof(int) + 1, sizeof(char), ntohl(\*temp), fp);

}

else

{

for (int i = sizeof(int) + 1; i < ntohl(\*temp) + sizeof(int) + 1; i++)

{

if (last != '\r' && buffer[i] == '\n')

{

fputc('\r', fp);

}

fputc(buffer[i], fp);

last = buffer[i];

}

}

filename[strlen(filename)] = '.';

if (access(filename, F\_OK) == 0)

{

remove(filename);

}

filename[strlen(filename) - 5] = '\0';

break;

}

else

{

break;

}

ret = recv(sockfd, (void\*)&buffer, BUFFER\_SIZE, 0);

}

if (buffer[0] != DATA && buffer[0] != END || ret < 1)

{

filename[strlen(filename)] = '.';

FILE \*tfp;

if ((tfp = fopen(filename, "w")) != NULL)

{

fwrite(&breakpoint, sizeof(int), 1, tfp);

if (mode == ASCII)

{

fputc(last, tfp);

}

fclose(tfp);

}

fclose(fp);

failed\_count++;

continue;

}

fclose(fp);

if (buffer[0] != FAILED)

{

success\_count++;

}

}

printf("myftp: %d file has been get down. %d failed.\n", success\_count, failed\_count);

}

}

else if (strcmp(cmd, "quit") == 0)

{

if (c == '\n' || c == EOF)

{

buffer[0] = QUIT;

send(sockfd, (void\*)&buffer, 1, 0);

close(sockfd);

return;

}

while ((c = getchar()) != '\n' && c != EOF);

printf("ERROR: Invalid parameter input. (quit)\n");

}

else

{

if (c != '\n' && c != EOF)

{

while ((c = getchar()) != '\n' && c != EOF);

}

printf("ERROR: Command doesn't exist!\n");

}

}

}

最后，将以上内容封装为myftp.h和myftp.c。

四、产品打包

最终产品由automake自动生成Makefile，以下是该项目在automake中相关的一些配置文件设置：

服务器端：

ftpserver/configure.ac：

# -\*- Autoconf -\*-

# Process this file with autoconf to produce a configure script.

AC\_PREREQ([2.69])

AC\_INIT([ftpserver], [1.0], [1253649392@qq.com])

AC\_CONFIG\_SRCDIR([src/util.c])

AC\_CONFIG\_HEADERS([config.h])

AM\_INIT\_AUTOMAKE([-Wall -Werror foreign])

# Checks for programs.

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

AC\_CHECK\_HEADERS([arpa/inet.h stdlib.h string.h unistd.h])

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC\_FUNC\_CHOWN

AC\_FUNC\_MALLOC

AC\_CHECK\_FUNCS([getcwd memset mkdir rmdir socket strchr])

AC\_CONFIG\_FILES([Makefile

inc/Makefile

src/Makefile])

AC\_OUTPUT

ftpserver/Makefile.am：

SUBDIRS = inc src

dist\_doc\_DATA = README

ftpserver/inc/Makefile.am：

noinst\_HEADERS = util.h ftpserver.h

ftpserver/src/Makefile.am：

bin\_PROGRAMS = ftpserver

ftpserver\_LDFLAGS = -pthread

ftpserver\_CPPFLAGS = -I $(top\_srcdir)/inc

ftpserver\_SOURCES = main.c util.c ftpserver.c

ftpserverdatadir = $(datadir)/ftpserver

ftpserverdata\_DATA = users

EXTRA\_DIST = users

客户端：

ftpclient /configure.ac：

# -\*- Autoconf -\*-

# Process this file with autoconf to produce a configure script.

AC\_PREREQ([2.69])

AC\_INIT([myftp], [1.0], [1253649392@qq.com])

AC\_CONFIG\_SRCDIR([src/util.c])

AC\_CONFIG\_HEADERS([config.h])

AC\_PREREQ([2.69])

AM\_INIT\_AUTOMAKE([-Wall -Werror foreign])

# Checks for programs.

AC\_PROG\_CC

# Checks for libraries.

# Checks for header files.

AC\_CHECK\_HEADERS([arpa/inet.h stdlib.h string.h unistd.h])

# Checks for typedefs, structures, and compiler characteristics.

# Checks for library functions.

AC\_CHECK\_FUNCS([getcwd mkdir rmdir socket strchr])

AC\_CONFIG\_FILES([Makefile

inc/Makefile

src/Makefile])

AC\_OUTPUT

ftpclient /Makefile.am：

SUBDIRS = inc src

dist\_doc\_DATA = README

ftpclient/inc/Makefile.am：

noinst\_HEADERS = util.h myftp.h

ftpclient /src/Makefile.am：

bin\_PROGRAMS = myftp

myftp\_CPPFLAGS = -I $(top\_srcdir)/inc

myftp\_SOURCES = main.c util.c myftp.c

五、安装使用

按照默认设置，由make dist生成的安装包的默认安装路径为usr/local/bin，手动设置的默认ftp存储路径为usr/local/share/ftpserver/ftp，默认用户信息存储路径为usr/local/share/ftpserver/users（该文件存储用户名和密码）。使用前需要分别在ftpserver和ftpclient目录以root权限使用make install命令安装，不安装也可以，目录在ftpserver运行时会自动生成，但是所需用户信息文件需手动放置。因为权限相关操作需要借助系统自身，服务器端主机上务必已存在和users中用户名相同的用户。因为涉及到为不同用户文件设置和修改权限问题，服务器端应使用root用户运行。

经测试，试验成功。

4、实验总结

FTP服务系统实验的设计并不复杂，但想要完整地实现一个FTP服务系统需要结合《Linux系统编程》这门课程的多个关键知识点和对Linux系统运行机制的一定了解。通过这一门课程的学习和这一次实验的操作，我对Linux系统的操作的使用有了全新的认识，对多线程和socket编程也有了一定的经验。这一课程设计中遇到并解决的许多困难为我将来的编程学习打下了良好的基础。