安徽工程大学
Anhui Polytechnic University

Linux 操作系统与程序设计

课程设计说明书

设计名称：Linux 课程设计

设计题目：FTP 模拟系统

指导教师：包象琳

专业班级：人工智能 191

姓　　名：武新纪

学　　号：3190707121

起止日期：2021.6.1~2021.6.25

总评成绩：

前言

Linux 操作系统从第一个内核诞生到现在，其开放、安全、稳定的特性得到了越来越多用户的认可。它具有自由开放的源代码，真正的多用户多任务操作系统、良好的用户界面、强大的网络功能、可靠的系统安全、良好的移植性、完整的开发平台、Linux 自由软件的低成本、安全性，促使着 Linux 的广泛应用和蓬勃发展。如今，Linux 应用在各行各业中，从最早的 Web、FTP、邮件服务开始,桌布扩展到诸如个人桌面应用、网络安全、电子商务、远程教育、集群运算、网络计算、嵌入式系统等各个领域。日常生活中，我们使用的安卓手机、路由器，甚至各种物联网设备上，都有 Linux 的身影。

Linux 有多种发行版，每种发行版本都有着自己独特的功能。如 Ubuntu 对个人桌面应用支持的比较好，而 Centos 因为其稳定性更适合作为服务器的操作系统，kali 系统自带了很多黑客工具，常作为黑客使用的操作系统。Linux 的发行版本有很多，但是他们的系统内核都是一样的，都是 Linux 的内核，因此，很多操作都是适用的，熟悉基本的操作之后，即使是更换发行版本，也只需要很少的学习，便能够快速掌握。学习和掌握 Linux 操作系统，将会给我们工作和学习带来很大的便利。近些年来，随着人工智能与大数据技术的发展，Linux 也得到了进一步地发展，因为很多人工智能与大数据的应用都是部署在 Linux 上的，在保证稳定性的同时，又不失其效率，因此得到了广泛的应用。

在学习了 Linux 操作系统之后，我见识到了 Linux 系统的强大，通过研究和使用 Linux 操作系统，将会更有利于我们对其他知识的学习和掌握，从而进一步促进我们的学习和工作。这次的 Linux 课程设计题目我选择的是 FTP 模拟系统。FTP，英文名叫 File Transfer Protocol，即文件传输协议，是 TCP/IP 协议组中的协议之一。FTP 协议包括两个组成部分，其一为 FTP 服务器，其二为 FTP 客户端。其中 FTP 服务器用来存储文件，用户可以使用 FTP 客户端通过 FTP 协议访问位于 FTP 服务器上的资源。此外，由于 FTP 传输效率非常高，在网络上传输大的文件时，一般也采用该协议。FTP 协议涉及 Linux 操作系统多方面的知识，如文件管理，网络通信、进程控制、线程管理，通过了解 FTP 的原理并自己使用 C 语言去实现自己的 FTP 模拟系统，将是对这段时间 Linux 学习的一种总结，也是对已经学 Linux 知识的一种巩固。

关键字:Linux 操作系统 发行版 FTP 网络传输

目录

Linux 操作系统与程序设计课程设计任务书

1、 选题。每位同学任选 1 题(选题表另见：Linux 操作系统与程序设计指导书)，并到所在自然班的班长处登记，每个题目，每个班级选择人数不超过 5 人。如果自拟题目，需要经指导老师确认。

2、 课程设计成绩分为 5 级：优秀（5 分）、良好（4 分）、中等（3 分）、及格（2 分）、不及格（1 分）。

3、 题目有难易、工作量有大小之分，请同学们结合自身情况选择题目。为体现公平，教师在评定最终成绩时会考虑到这些因素。

4、 课程设计的报告和工程源代码应该独立完成，严禁抄袭。

5、 课程设计报告要严格遵照以下模板来撰写：

(1) 前言

(2) 目录

(3) 任务书：打印本页作为任务书。

(4) **需求分析**：描述系统各模块的主要功能性需求，画出相应的用例图或数据流图。

(5) **概要设计**：系统的主要功能模块、模块间的关系、主要界面、主要运行流程等。

(6) **详细设计和实现**：系统使用的有关技术简介；具体开发工具和运行环境；主要功能的主要源码；文件目录结构、清单文件等。

(7) **运行和测试**：主要功能/模块的测试、运行界面截图（要写操作过程）。

(8) **小结**：整个课程设计过程的收获、思考，系统的不足与展望等。

(9) **参考文献**：不少于 5 篇。

(10) **附录**：主要源代码。

6、 推荐使用 Linux 下的 QT 工具，Linux 下的 make 工具，gcc 编译工具等作为开发工具。

7、 课程设计成果提交。提交：电子版工程项目+课程设计报告和纸质版课程设计报告。电子版要用压缩为单一文件，命名如：人工智能 191-学号-姓名-题目.rar，所有课程设计成果先在学习委员处汇总，然后统一提交给指导老师。

8、 提交时间。报告和系统应于 2021 年 6 月　日前完成，报告手写或打印后提交。

指导教师(签名) _____

2021 年 6 月　日

一、概述

1.1、目的与要求

1.1.1 目的

利用 Linux 环境下的 Socket API 实现 FTP 模拟系统，有客户端和服务端两部分程序，客户端和服务端之间可以进行简单的文件传输，并且能够使用一些命令对客户端和服务端的文件进行简单操作，如显示文件列表、切换工作目录，删除文件和打印当前工作目录等。

1.1.2 要求

（1）、实验者先要独立进行需求分析，确定任务需求；
（2）、根据任务需求，完成应用的功能设计；
（3）、根据功能设计，完成能够实现所需功能的详细设计；
（4）. 使用 gcc，gdb 等编程工具，实现 FTP 模拟系统的编写、调试工作；
（5）. 将设计过程写出完整的课程设计报告，报告要求必须附上主要程序界面和程序源代码。

1.2、选题内容

利用 Linux 环境下 Socket API 实现 FTP 模拟系统，该系统分为服务器和客户端两部分，服务器端依据 FTP 协议运行，接受客户端连接请求，当接受请求后建立一个服务器分线程，并新建新的 Socket 连接处理文件传输工作。客户端提供 FTP 服务器连接请求，并提供 FTP 命令相关功能。

1.3、设计的过程

（1）、FTP 协议包括两个组成部分，其一为 FTP 服务器，其二为 FTP 客户端。其中 FTP 服务器用来存储文件，用户可以使用 FTP 客户端通过 FTP 协议访问位于 FTP 服务器上的资源。默认情况下 FTP 协议使用 TCP 端口中的 20 和 21 这两个端口，其中 20 用于传输数据，21 用于传输控制信息。但是本次 Linux 课程设计是 FTP 模拟系统，为使程序更利于理解，简化了 FTP 协议过程，原本 ftp 使用两个端口分别传输数据和控制信息，现在只用了一个端口同时完成这两个功能。除文件传输功能，还实现了简单的文件管理功能，如 ls 显示文件列表、cd 切换文件目录、rm 删除文件等。

(2)、该系统在设计过程中没有过多使用多线程和多进程的相关知识，因此可能不能够轻松处理太多客户端同时上传或下载文件的要求，后续会考虑为其

添加相应的功能，以实现对同一时间内多请求、高并发的处理。该系统客户端与服务端连接时采用 TCP 的连接方式，但是部署客户端程序并不是运行之后就一直和服务端相连，而是每执行一次命令，则进行一次连接，当命令执行完毕后，会将连接断开，下次再执行命令时，再发起连接。虽然看起来繁琐，但却是不使用多进程多线程时对高并发情况的一种解决方法。因为程序使用 C 语言进行编写，程序执行效率非常高，因此并不会感受到有任何的延迟情况。

(3)、服务端。程序运行时先创建一个 socket，并将其与本地的设备信息绑定，并绑定本地的通信端口，监听端口发来的信息，然后对从获取的信息进行处理。接受到的信息有两种表现形式，一种是客户端发送过来的命令，包含命令本身和命令的参数，服务端执行命令之后将执行结果通过 socket 发回客户端；另一种是客户端发送过来的文件内容，服务端会进行判断，如果发送过来的是文件内容，则会在服务器端创建一个指定文件，并将从 socket 获取到的内容写入到新文件，从而实现文件的上传。文件的下载也很类似，获取客户端发来的下载文件的请求之后，读取指定的文件的内容后，通过 socket 将文件内容发送给客户端，再由客户端进行一系列处理。

（4)、客户端程序。客户端程序有一个模拟 ftp 终端，当用户运行客户端程序之后，就可以在 ftp 模拟终端中进行操作。通过输入相关命令，来实现对客户端文件或服务端文件的管理。目前已经实现的文件管理功能有文件的展示和删除，目录的切换和打印等。ftp 模拟终端的命令可以分成三类，一类是需要建立 socket 并与服务端进行交互的命令，这类命令会将命令处理后发送给服务端，然后服务端执行命令后将结果返回给客户端，客户端再将结果通过 ftp 模拟终端展示给用户，这类命令有 ls,cd 和 pwd 命令等；另一类则是客户端文件处理命令，这些命令在解析之后调用相关的系统函数进行处理，处理之后将结果通过 ftp 模拟终端直接展现给用户，如 lls、lcd 和 lpwd 等命令；最后一种，则是 help 和 exit 命令，若用户输入 help 则给出所有命令的使用方式，若用户输入 exit 则退出 ftp 模拟终端并结束客户端程序。用户可以使用 get 加文件名的方式从服务端下载指定文件，也可以使用 put 加文件名的方式将本地文件上传到服务器，当然，两端的文件都必须存在，否则会报错。get 和 put 两个命令属于第一类命令，需要建立 socket 并于服务端进行交互的命令。

（5)、程序全部使用 C 语言进行编写，使用了 Linux 系统下 Socket API 和 Linux 系统下一些简单的文件管理函数。C 语言程序使用 Linux 系统下的 gcc 程序进行编译，使用 gdb 工具进行调试。

1.4、开发环境

操作系统：Ubuntu 21.04
硬件环境:AMD 4800U 8G 内存
编译和调试工具:gcc gdb

二、需求分析

2.1、功能需求分析

- 实现客户端本地的一些文件操作的基础命令，如 lls 列出当前工作目录下的所有文件，lcd 进行当前工作目录的切换，lrm 实现对本地文件的删除。
- 实现对服务器端一些文件操作的基础命令，如 ls 列出服务器端当前工作目录下的所有文件，cd 实现对服务器端当前工作目录的切换，rm 实现对服务器文件的文件的删除。
- 实现文件传输的两个命令——get 和 put 命令。get 是将服务器端的文件下载到本地，而 put 是将客户端的文件上传的服务器。
- 要实现的所有命令如下:

命令	功能
ls	列出服务端当前的文件和目录
cd	切换服务端目录
pwd	打印当前所在服务端目录
get	下载服务端文件
rm	删除服务端文件
lls	列出客户端当前的文件和目录
lcd	切换客户端目录
lpwd	打印当前客户端所在目录
put	将客户端文件上传到服务端
lrm	删除客户端文件
exit	退出 ftp 模拟系统
help	帮助命令，查看所有命令用法

三、系统设计

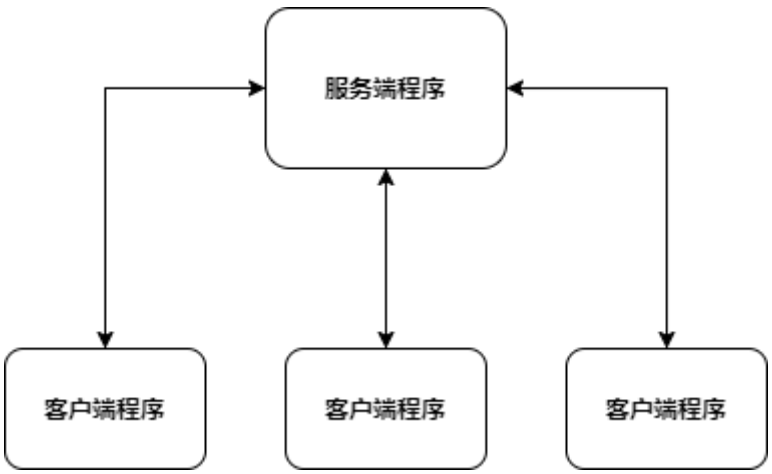
3.1 设计概述

该系统分为两部分，服务端和客户端，服务端可以接收多个客户端连接，客户端与服务端之间使用 TCP 的方式进行连接，每个客户端在与服务端相连时均可以执行相应操作。服务端启动后就进入了循环，没有设置终止方式，用户可以使用<Ctrl+C>或通过 Linux 系统命令 kill 来结束服务端程序。服务端程序一般运行在后台，不与用户进行交互。客户端程序在启动后会进入一个模拟的 ftp 终端，在这里可以输入一些 ftp 命令，以此来进行本地和服务器端的文件管理以及文件和文件的上传和下载功能，执行命令之后，ftp 模拟终端会将执行后的结果通过 ftp 模拟终端显示给用户。

服务端和客户端的程序，文件列表的显示、文件目录的切换和文件的删除，均使用 Linux 下的系统函数来完成。文件上传和下载时对文件的读写使用不带缓存的 I/O 进行，这种方法速度较快，可以保证文件的传输速度，但实际的传输速度还取决于网络、系统内存、CPU 性能等多个条件。客户端和服务端是两个程序，只有当服务端启动时客户端才可以进行服务端的相关操作，客户端和服务端直接使用 socket 进行交换数据。

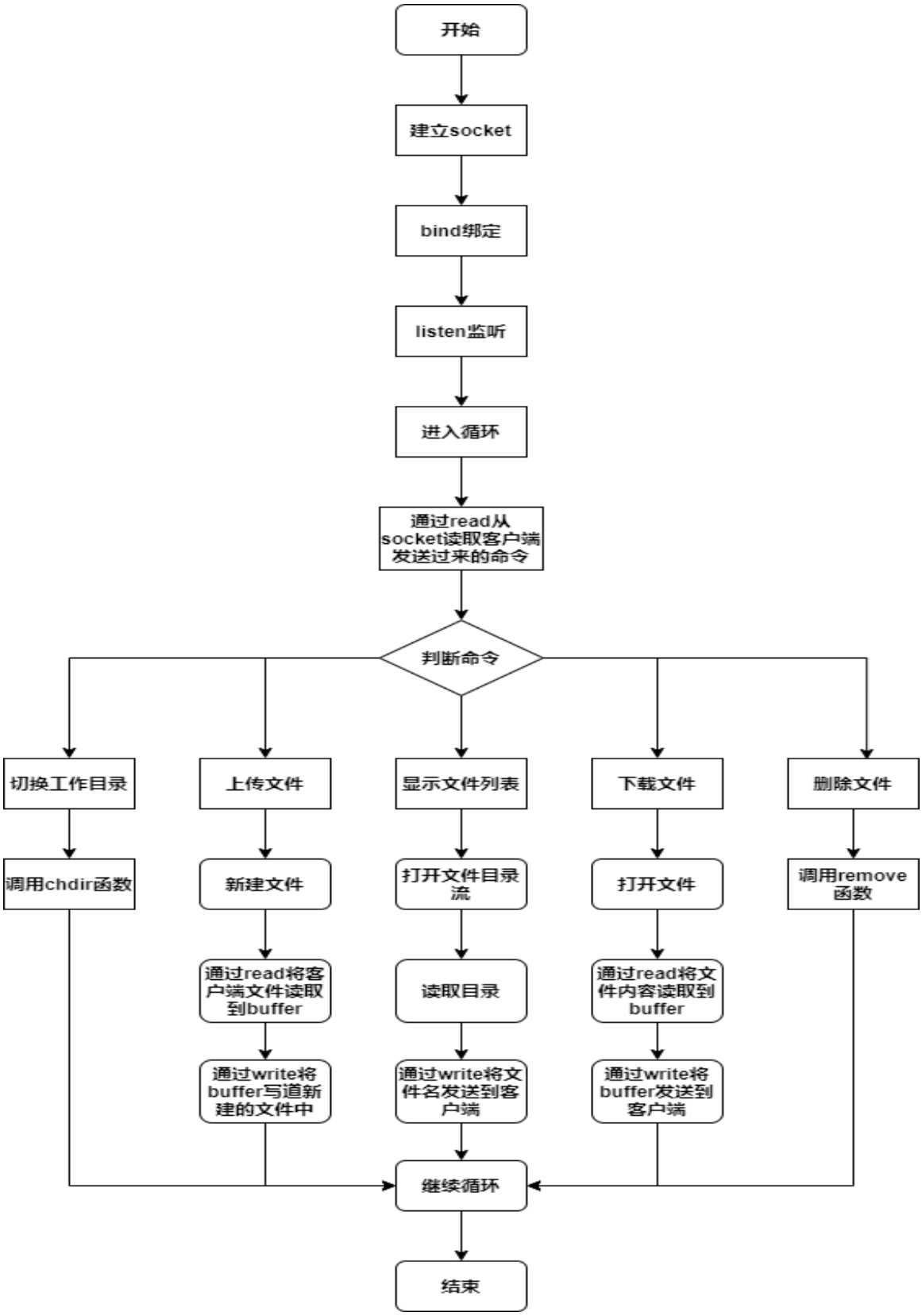
3.2 设计流程图

3.2.1 总体设计流程图

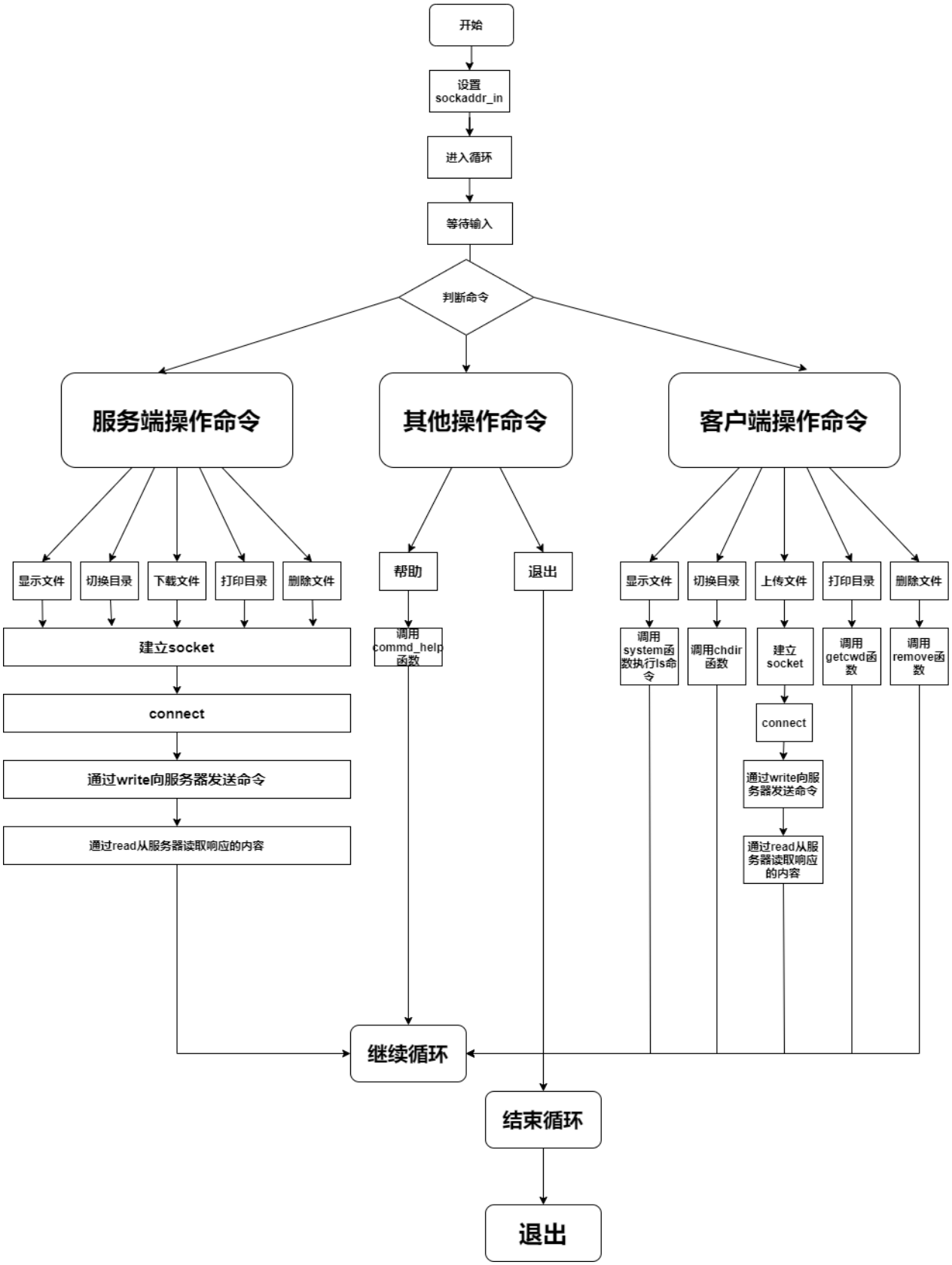


3.2.2 详细设计流程图

(1)服务端设计流程图



(2)客户端设计流程图


开始
设置sockaddr_in
进入循环
等待输入
判断命令
服务端操作命令
其他操作命令
客户端操作命令
显示文件
切换目录
下载文件
打印目录
删除文件
建立socket
connect
通过write向服务器发送命令
通过read从服务器读取响应的内容
帮助
退出
调用commd_help函数
显示文件
切换目录
上传文件
打印目录
删除文件
调用system函数执行ls命令
调用chdir函数
建立socket
调用getcwd函数
调用remove函数
connect
通过write向服务器发送命令
通过read从服务器读取响应的内容
继续循环
结束循环
退出

四、详细设计与实现

4.1 主要功能模块声明

```
//定义的命令结构体，包含命令和参数
typedef struct command
{
    char commd[MAXDATASIZE];
    char arg[MAXDATASIZE];
} Commd;

// 服务端函数声明
void commd_ls(int sockfd, char *arg);  //显示文件列表
void commd_get(int sockfd, char *arg); //下载文件
void commd_put(int sockfd, char *arg); //上传文件
void commd_rm(int sockfd, char *arg);  //删除文件
void commd_cd(int sockfd, char *arg);  //切换工作目录
void commd_pwd(int sockfd);            //打印当前工作目录
int split(char *str, Commd *commd);    //分割客户端发来的信息获得命令和参数

//客户端函数声明
int commd_lls(char *arg);          //列出客户端当前目录下的文件和目录
int commd_lcd(char *arg);          //切换客户端当前所在的目录
int commd_lpwd();                  //打印客户端当前所在目录完整路径名
int commd_lrm(char *arg);          //删除当前所在目录下的指定文件
void commd_get();                 //下载文件
//显示服务端文件列表
void commd_ls(struct sockaddr_in my_addr, char *arg);
//切换服务端当前工作目录
void commd_cd(struct sockaddr_in my_addr, char *arg);
//打印服务端端当前工作目录
void commd_pwd(struct sockaddr_in my_addr);
void commd_put(struct sockaddr_in my_addr, char *arg); //上传文件
void commd_rm(struct sockaddr_in my_addr, char *arg);  //删除服务端文件
int split(char *str, Commd *commd);         //分割用户输入，获得命令和参数
void commd_help();                          //帮助命令
```

4.2 主要功能模块的具体实现

服务端：

```
//分割客户端发来的信息，获得命令和参数
int split(char *str, Commd *commd){
    int n = 0;
    char *ret;
    char dataList[10][20];
    ret = strtok(str, " ");
    while (ret != NULL){
        strcpy(dataList[n++], ret);
        ret = strtok(NULL, " ");
    }
    for (int i = 0; i < 2; i++)
        if (dataList[i][strlen(dataList[i]) - 1] == '\n')
            dataList[i][strlen(dataList[i]) - 1] = '\0';
    strcpy(commd->commd, dataList[0]);
    strcpy(commd->arg, dataList[1]);
    return n;
}

// 实现文件的下载
void commd_get(int sockfd, char *filename){
    int fd, nbytes;
    char buffer[N];
    bzero(buffer, N);
    printf("get filename : [ %s ]\n", filename);
    if ((fd = open(filename, O_RDONLY)) < 0){
    //以只读的方式打开 client 要下载的文件
        perror("Open file Error!\n");
        buffer[0] = 'N';
        if (write(sockfd, buffer, N) < 0){
            perror("Write Error!At commd_get 1\n");
            exit(1);
        }
        return;
    }
    buffer[0] = 'Y'; //此处标示出文件读取成功
    if (write(sockfd, buffer, N) < 0){
        perror("Write Error! At commd_get 2!\n");
        close(fd);
        exit(1);
    }
    while ((nbytes = read(fd, buffer, N)) > 0){ //将文件内容读到 buffer 中
```

```
        if (write(sockfd, buffer, nbytes) < 0){ //将 buffer 发送回 client
            perror("Write Error! At commd_get 3!\n");
            close(fd);
            exit(1);
        }
    }
    close(fd);
    close(sockfd);
    return;
}

// 实现文件的上传
void commd_put(int sockfd, char *filename){
    int fd, nbytes;
    char buffer[N];
    bzero(buffer, N);
    printf("get filename : [ %s ]\n", filename);
    if ((fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0){
    //以只写的方式打开文件，若文件存在则清空，若文件不存在则新建文件
        perror("打开文件错误!\n");
        return;
    }
    while ((nbytes = read(sockfd, buffer, N)) > 0) {
    //将 client 发送的文件写入 buffer
       if (write(fd, buffer, nbytes) < 0) { //将 buffer 中的内容写到文件中
            perror("写入错误!\n");
            close(fd);
            exit(1);
        }
    }
    close(fd);
    close(sockfd);
    return;
}
```

客户端：

```
int commd_lls(char *arg) {//列出客户端当前目录下的文件和目录
    int fd, err;
    char buf[MAXBUF], commd_str[100];
    if (arg != NULL)
        sprintf(commd_str, "ls -la %s > /tmp/buf", arg);
    err = system(commd_str);
    if (err < 0)
        return -1;
    fd = open("/tmp/buf", O_RDONLY);
    if (fd < 0)
        return -1;
    read(fd, buf, sizeof(buf));
    close(fd);
    remove("rm /tmp/buf");
    printf("%s\n", buf);
    return 0;
}

//显示服务端文件列表
void commd_ls(struct sockaddr_in my_addr, char *arg){
    int sockfd;
    char commd[N];
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){ //创建套接字
        printf("Socket Error!\n");
        return;
    }
    if (connect(sockfd,(struct sockaddr *)&my_addr,sizeof(my_addr))<0){
        printf("Connect Error!\n");
        return;
    }
    sprintf(commd, "ls %s", arg);
    if (write(sockfd, commd, N) < 0){
        printf("写入错误\n");
        exit(1);
    }
    //利用 read 函数来接受服务器发来的数据
    while (read(sockfd, commd, N) > 0)
        printf("%s ", commd);
    printf("\n");
    close(sockfd);
    return;
}
```

```
//实现上传文件的功能
void commd_put(struct sockaddr_in my_addr, char *arg){
    int fd;
    int sockfd;
    char buffer[N], commd[N];
    int nbytes;
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        printf("Socket Error!\n");
        exit(1);
    }
    //客户端与服务端连接
    if (connect(sockfd,(struct sockaddr *)&my_addr,sizeof(my_addr))<0){
        printf("Connect Error!\n");
        exit(1);
    }
    sprintf(commd, "put %s", arg);
    if (write(sockfd, commd, N)<0){//从 commd 中读取 N 字节数据写入套接字中
        printf("Wrtie Error!At commd_put 1\n");
        exit(1);
    }
    //open 函数从(commd+4)中，读取文件路径，以只读的方式打开
    if ((fd = open(arg, O_RDONLY)) < 0){
        printf("Open Error!\n");
        exit(1);
    }
    //从 fd 指向的文件中读取 N 个字节数据
    while ((nbytes = read(fd, buffer, N)) > 0){
        //从 buffer 中读取 nbytes 字节数据，写入套接字中
        if (write(sockfd, buffer, nbytes) < 0){
            printf("Write Error!At commd_put 2");
        }
    }
    close(fd);
    close(sockfd);
    return;
}

// 实现文件下载功能
void commd_get(struct sockaddr_in my_addr, char *arg){
    int fd;
    int sockfd;
    char buffer[N], commd[N];
    int nbytes;
```

```
    //创建套接字，并进行错误检测
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        printf("Socket Error!\n");
        exit(1);
    }
    //connect 函数用于实现客户端与服务端的连接,此处还进行了错误检测
    if (connect(sockfd,(struct sockaddr*)&my_addr,sizeof(my_addr))<0){
        printf("Connect Error!\n");
        exit(1);
    }
    sprintf(commd, "get %s", arg);
    if (write(sockfd, commd, N) < 0){  //通过 write 函数向服务端发送数据
        printf("Write Error!At commd_get 1\n");
        exit(1);
    }
    if (read(sockfd, buffer, N) < 0){//利用 read 函数来接受服务器发来的数据
        printf("Read Error!At commd_get 1\n");
        return;
    }
    if (buffer[0] == 'N'){ //用于检测服务器端文件是否打开成功
        close(sockfd);
        printf("Can't Open The File!\n");
        return;
    }
    //open 函数创建一个文件，文件地址为(commd+4)，该地址从命令行输入获取
    if ((fd = open(arg, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0){
        printf("Open Error!\n");
        exit(1);
    }
    //read 函数从套接字中获取 N 字节数据放入 buffer 中，返回值为读取的字节数
    while ((nbytes = read(sockfd, buffer, N)) > 0){
    //write 函数将 buffer 中的内容读取后写入 fd 所指向的文件
        if (write(fd, buffer, nbytes) < 0){
            printf("Write Error!At commd_get 2");
        }
    }
    close(fd);
    close(sockfd);
    return;
}
```

五、运行与调试

5.1 运行调试与截图

1、帮助命令

```
root@ubuntu:/home/xjfyt/ftp_client# ./client
ftp>help
                    **********************ftp命令一览**************************************
                    *                                                                      *
                    *       <1>.ls      <文件或目录>        列出服务端当前的文件和目录    *
                    *       <2>.lls     <文件或目录>        列出客户端当前的文件和目录    *
                    *       <3>.rm      <文件或目录>        删除服务端文件                *
                    *       <4>.lrm     <文件或目录>        删除客户端文件                *
                    *       <5>.cd        <目录>            切换服务端目录                *
                    *       <6>.lcd       <目录>            切换客户端目录                *
                    *       <7>.get       <文件>            下载服务端文件                *
                    *       <8>.put       <文件>            将客户端文件上传到服务端      *
                    *       <9>.pwd                         打印当前所在服务端目录        *
                    *       <10>.lpwd                       打印当前客户端所在目录        *
                    *       <12>.help                       帮助命令，查看所有命令用法    *
                    *                                                欢迎使用ftp模拟系统！ *
                    ************************************************************************
ftp>
```

2、列出客户端当前目录下所有文件

```
ftp>lls
总用量 76
drwxrwxrwx  2 root  root   4096  6月 25 04:57 .
drwxr-x--- 23 xjfyt xjfyt  4096  6月 25 04:29 ..
-rwxrwxrwx  1 root  root  21568  6月 25 04:55 client
-rwxrwxrwx  1 root  root    896  6月 25 04:56 ifconfig
-rwxrwxrwx  1 root  root   2470  6月 25 04:57 neofetch
-rwxrwxrwx  1 root  root      6  6月 25 04:56 test.c
-rwxrwxrwx  1 root  root  28742  6月 25 04:55 test.log
```

3、列出服务端当前目录下所有文件

```
ftp>ls
test.c path file . 1.cpp .. server
ftp>
```

4、分别打印客户端和服务端当前工作目录的完整路径

```
ftp>lpwd
/home/xjfyt/ftp_client
ftp>pwd
/home/xjfyt/ftp_server
ftp>
```

5、切换服务端工作目录

```
ftp>pwd
/home/xjfyt/ftp_server
ftp>cd FTP
切换成功
ftp>pwd
/home/xjfyt/ftp_server/FTP
ftp>
```

6、切换客户端工作目录

```
ftp>lpwd
/home/xjfyt/ftp_client
ftp>lcd Code
ftp>lpwd
/home/xjfyt/ftp_client/Code
ftp>
```

7、下载文件

```
ftp>lls
总用量 72
drwxrwxrwx  3 root  root   4096  6月 25 05:48 .
drwxr-x--- 23 xjfyt xjfyt  4096  6月 25 04:29 ..
-rwxrwxrwx  1 root  root  21568  6月 25 04:55 client
drwxrwxr-x  2 xjfyt xjfyt  4096  6月 25 05:05 Code
-rwxrwxrwx  1 root  root      6  6月 25 04:56 test.c
-rwxrwxrwx  1 root  root  28742  6月 25 04:55 test.log

ftp>ls
test.c path file . FTP 1.cpp .. server
ftp>get path
ftp>lls
总用量 76
drwxrwxrwx  3 root  root   4096  6月 25 05:50 .
drwxr-x--- 23 xjfyt xjfyt  4096  6月 25 04:29 ..
-rwxrwxrwx  1 root  root  21568  6月 25 04:55 client
drwxrwxr-x  2 xjfyt xjfyt  4096  6月 25 05:05 Code
-rw-r--r--  1 xjfyt xjfyt    23  6月 25 05:50 path
-rwxrwxrwx  1 root  root      6  6月 25 04:56 test.c
-rwxrwxrwx  1 root  root  28742  6月 25 04:55 test.log
```

未下载文件前客户端文件列表

服务端文件列表

下载文件命令

下载文件后客户端文件列表

已经将文件下载到本地

8、上传文件

```
ftp>ls
test.c path file . FTP 1.cpp .. server
ftp>lls
总用量 80
drwxrwxrwx  3 root  root   4096  6月 25 05:56 .
drwxr-x--- 23 xjfyt xjfyt  4096  6月 25 04:29 ..
-rwxrwxrwx  1 root  root  21568  6月 25 04:55 client
drwxrwxr-x  2 xjfyt xjfyt  4096  6月 25 05:05 Code
-rw-rw-r--  1 xjfyt xjfyt  2526  6月 25 05:56 neofetch
-rw-r--r--  1 xjfyt xjfyt    23  6月 25 05:50 path
-rwxrwxrwx  1 root  root      6  6月 25 04:56 test.c
-rwxrwxrwx  1 root  root  28742  6月 25 04:55 test.log

ftp>put neofetch
ftp>ls
test.c path neofetch file . FTP 1.cpp .. server
ftp>
```

未上传前服务端文件列表

客户端文件列表

上传文件

上传文件后服务端文件列表

文件已经被上传到服务端

六、小结

通过这次 Linux 课程设计，我对 Linux 操作系统的知识的理解又更近了一步，之前的我只能够使用些基本命令实现对 Linux 系统的操作，而现在我能够熟练地使用 C 语言这种高级语言去操作 Linux 系统。众所周知，Linux 的内核是使用 C 语言去写的，那么熟练使用 C 语言去操作 Linux 系统将会更有利于接下来操作系统和 Linux 内核源码等知识的学习。在本次课程设计过程中，我学会了使用 Linux 下的 Socket API 进行一些简单的程序设计，在设计过程中，我还学会了熟练地使用 Linux 系统函数，并能够熟练使用 C 语言操作 Linux 下的文件和时间，真的 Linux 学习的一大进步。

但是因为这次课程设计和期末考试相冲突，时间紧，任务重，程序中有很多可以再进一步优化的地方我并没有来得及去深入优化，但之后我会仔细研究，对其进行进一步优化，以设计出优秀的 FTP 模拟系统。此外，这次我选择的 Linux 课程设计题目是 FTP 模拟系统，涉及的 Linux 知识主要是 Linux 系统下 Socket API 和系统 API 的使用，因此在做课程设计时，我侧重了对这些方面的知识的学习。但对 Linux 下进程、线程的创建和管理的知识涉及的比较少，但是也去研究了，力求使自己掌握的 Linux 知识更全面。另外，相比于其它 Linux 课程设计题目，我设计的程序只是终端程序，并没有使用 Qt 或 SDL 去设计相应的图形界面。图形界面相比于终端程序，将更容易实现用户与机器的交互，带来更优的用户体验。这次没有做图形界面的程序，属实是一种遗憾，之后有时间，一定会考虑做一个有图形界面的 FTP 程序，甚至让其能够跨平台，来实现不同平台之间的 FTP 文件传输。

最初看 Linux 课程设计题目的时候还是感觉时很难的，觉得无从下手，可能是因为在进行 Linux 操作系统课程学习的时候，学习的知识太零散，没有形成一个完整的知识体系。通过做课程设计，我学会了把在课堂上学到的 Linux 知识整合到一起，并实现了综合应用，所以非常感谢老师给了我们这次做课程设计的机会。Linux 课程学习和 Linux 课程设计虽然结束了，但是对于 Linux 知识的学习却不会因此而结束，相信在我接下来的学习和工作中，将会经常和 Linux 操作系统打交道，因此我会不断地学习的 Linux 操作系统的知识，并将其深入应用到工作和学习中去，也会将其与自己的专业知识相结合，从而发挥更大的作用。

参考文献

[1] 金国庆、刘加海.《Linux 程序设计（第三版）》.浙江大学出版社,2020.8
[2] 钟志水、周鸣争.《C 语言程序设计》.电子科技大学出版社,2018.12
[3] [美] Dennis M. Ritchie.《C 程序设计语言[M]》.机械工业出版社,2004.
[4] [美]詹姆斯·F.库罗斯.《计算机网络》.机械工程出版社,2018.6
[5] 谢希人.《计算机网络》.电子工业出版社,2017.1
[6] 鸟哥.《鸟哥的 Linux 私房菜》.人民邮电出版社,2018.11
[7] 零壹快学.《零基础 Linux 从入门到精通》.广东人民出版社,2019.10

附录

//服务端代码

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <dirent.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#define MAXDATASIZE 100
#define N 256 //文件名和命令名最长为256字节

//定义的命令结构体，包含命令和参数
typedef struct command
{
    char commd[MAXDATASIZE];
    char arg[MAXDATASIZE];
} Commd;

// 服务端函数声明
void commd_ls(int sockfd, char *arg);  //显示文件列表
void commd_get(int sockfd, char *arg); //下载文件
void commd_put(int sockfd, char *arg); //上传文件
void commd_rm(int sockfd, char *arg);  //删除文件
void commd_cd(int sockfd, char *arg);  //切换工作目录
void commd_pwd(int sockfd);            //打印当前工作目录
int split(char *str, Commd *commd);    //分割客户端发来的信息，获得命令和参数

int main(int arg, char *argv[])
{
    int ser_sockfd, cli_sockfd;
    pthread_t nid;
    struct sockaddr_in ser_addr, cli_addr;
    int ser_len, cli_len;
    char commd_str[N];
    Commd commd;
    bzero(commd_str, N); //将commd所指向的字符串的前N个字节置为0，包括'\0'
```

```
    if ((ser_sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Sokcet Error!\n");
        return -1;
    }

    bzero(&ser_addr, sizeof(ser_addr));
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY); //本地 ip 地址
    ser_addr.sin_port = htons(8989);              //转换成网络字节
    ser_len = sizeof(ser_addr);
    //将 ip 地址与套接字绑定
    if ((bind(ser_sockfd, (struct sockaddr *)&ser_addr, ser_len)) < 0)
    {
        perror("Bind Error!\n");
        return -1;
    }
    //服务器端监听
    if (listen(ser_sockfd, 5) < 0)
    {
        perror("Linsten Error!\n");
        return -1;
    }

    bzero(&cli_addr, sizeof(cli_addr));
    ser_len = sizeof(cli_addr);

    while (1)
    {
        printf("server>");
        //服务器端接受来自客户端的连接，返回一个套接字，此套接字为新建的一个，
并将客户端的地址等信息存入 cli_addr 中
        //原来的套接字仍处于监听中
        if ((cli_sockfd = accept(ser_sockfd, (struct sockaddr *)&cli_ad
dr, &cli_len)) < 0)
        {
            perror("响应错误!\n");
            exit(1);
        }
        //由套接字接收数据时，套接字把接收的数据放在套接字缓冲区，再由用户程序
把它们复制到用户缓冲区，然后由 read 函数读取
        //write 函数同理
```

```
        if (read(cli_sockfd, commd_str, N) < 0) //read 函数从 cli_sockfd
中读取 N 个字节数据放入 commd 中
        {
            perror("读取错误!\n");
            exit(1);
        }
        split(commd_str, &commd);
        if (strcmp(commd.commd, "ls") == 0)
        {
            commd_ls(cli_sockfd, commd.arg);
        }
        else if (strcmp(commd.commd, "cd") == 0)
            commd_cd(cli_sockfd, commd.arg);
        else if (strcmp(commd.commd, "pwd") == 0)
            commd_pwd(cli_sockfd);
        else if (strcmp(commd.commd, "rm") == 0)
            commd_rm(cli_sockfd, commd.arg);
        else if (strcmp(commd.commd, "get") == 0)
            commd_get(cli_sockfd, commd.arg);
        else if (strcmp(commd.commd, "put") == 0)
            commd_put(cli_sockfd, commd.arg);
        else
            printf("命令错误!\n");
    }
    return 0;
}
//分割客户端发来的信息，获得命令和参数
int split(char *str, Commd *commd)
{
    int n = 0;
    char *ret;
    char dataList[10][20];
    ret = strtok(str, " ");
    while (ret != NULL)
    {
        strcpy(dataList[n++], ret);
        ret = strtok(NULL, " ");
    }
    for (int i = 0; i < 2; i++)
        if (dataList[i][strlen(dataList[i]) - 1] == '\n')
            dataList[i][strlen(dataList[i]) - 1] = '\0';
    strcpy(commd->commd, dataList[0]);
    strcpy(commd->arg, dataList[1]);
    return n;
```

```
}

// 显示文件列表
void commd_ls(int sockfd, char *arg)
{
    DIR *mydir = NULL;
    struct dirent *myitem = NULL;
    char commd[N];
    bzero(commd, N);
    //opendir 为用来打开参数 name 指定的目录，并返回 DIR*形态的目录流
    //mydir 中存有相关目录的信息
    if ((mydir = opendir(arg)) == NULL)
    {
        perror("OpenDir Error!\n");
        exit(1);
    }

    while ((myitem = readdir(mydir)) != NULL) //用来读取目录,返回是 dirent
结构体指针
    {
        if (sprintf(commd, myitem->d_name, N) < 0) //把文件名写入 commd 指
向的缓冲区
        {
            perror("Sprintf Error!\n");
            exit(1);
        }

        if (write(sockfd, commd, N) < 0) //将 commd 缓冲区的内容发送会
client
        {
            perror("Write Error!\n");
            exit(1);
        }
    }

    closedir(mydir); //关闭目录流
    close(sockfd);
    return;
}

// 实现文件的下载
void commd_get(int sockfd, char *filename)
{
    int fd, nbytes;
```

```
    char buffer[N];
    bzero(buffer, N);
    printf("get filename : [ %s ]\n", filename);
    if ((fd = open(filename, O_RDONLY)) < 0) //以只读的方式打开 client 要
下载的文件
    {
        perror("Open file Error!\n");
        buffer[0] = 'N';
        if (write(sockfd, buffer, N) < 0)
        {
            perror("Write Error!At commd_get 1\n");
            exit(1);
        }
        return;
    }
    buffer[0] = 'Y'; //此处标示出文件读取成功
    if (write(sockfd, buffer, N) < 0)
    {
        perror("Write Error! At commd_get 2!\n");
        close(fd);
        exit(1);
    }
    while ((nbytes = read(fd, buffer, N)) > 0) //将文件内容读到 buffer 中
    {
        if (write(sockfd, buffer, nbytes) < 0) //将 buffer 发送回 client
        {
            perror("Write Error! At commd_get 3!\n");
            close(fd);
            exit(1);
        }
    }
    close(fd);
    close(sockfd);
    return;
}

// 实现文件的上传
void commd_put(int sockfd, char *filename)
{
    int fd, nbytes;
    char buffer[N];
    bzero(buffer, N);
    printf("get filename : [ %s ]\n", filename);
```

```
    if ((fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0)
//以只写的方式打开文件，若文件存在则清空，若文件不存在则新建文件
    {
        perror("打开文件错误!\n");
        return;
    }
    while ((nbytes = read(sockfd, buffer, N)) > 0) //将 client 发送的文件
写入 buffer
    {
        if (write(fd, buffer, nbytes) < 0) //将 buffer 中的内容写到文件中
        {
            perror("写入错误!\n");
            close(fd);
            exit(1);
        }
    }
    close(fd);
    close(sockfd);
    return;
}

//实现服务器端文件的删除
void commd_rm(int sockfd, char *arg)
{
    int err;
    char buf[100];
    err = remove(arg);
    sprintf(buf, "%d", err);
    if (write(sockfd, buf, 100) < 0) //将 commd 缓冲区的内容发送会 client
    {
        perror("写入错误!\n");
        exit(1);
    }
}

// 实现服务器端目录的切换
void commd_cd(int sockfd, char *arg)
{
    int err;
    char buf[100];
    err = chdir(arg);
    sprintf(buf, "%d", err);
    if (write(sockfd, buf, 100) < 0) //将 commd 缓冲区的内容发送会 client
    {
```

```
        perror("写入错误!\n");
        exit(1);
    }
}

// 实现打印服务器端当前所在目录
void commd_pwd(int sockfd)
{
    char buf[100];
    getcwd(buf, sizeof(buf));
    if (write(sockfd, buf, 100) < 0) //将 commd 缓冲区的内容发送会 client
    {
        perror("写入错误!\n");
        exit(1);
    }
}


//客户端代码
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <ctype.h>
#include <string.h>
#define SERVERPORT 21
#define MAXDATASIZE 100
#define MAXBUF 1000
#define N 256

//定义的命令结构体，包含命令和参数
typedef struct command
{
    char commd[MAXDATASIZE]; //基本命令
    char arg[MAXDATASIZE];   //命令参数
} Commd;

//客户端函数声明
```

```
int commd_lls(char *arg);                                   //列出客户端当前目
录下的文件和目录
int commd_lcd(char *arg);                                   //切换客户端当前所
在的目录
int commd_lpwd();                                           //打印客户端当前所
在目录完整路径名
int commd_lrm(char *arg);                                   //删除当前所在目录
下的指定文件
void commd_get();                                           //下载文件
void commd_ls(struct sockaddr_in my_addr, char *arg);       //显示服务端文件列
表
void commd_cd(struct sockaddr_in my_addr, char *arg);       //切换服务端当前工
作目录
void commd_pwd(struct sockaddr_in my_addr);                 //打印服务端端当前
工作目录
void commd_put(struct sockaddr_in my_addr, char *arg);      //上传文件
void commd_rm(struct sockaddr_in my_addr, char *arg);       //删除服务端文件
int split(char *str, Commd *commd);                         //分割用户输入，获
得命令和参数
void commd_help();                                          //帮助命令

int main(int argc, char *argv[])
{
    int sockfd, err, n;
    Commd commd;
    char input[MAXDATASIZE];
    struct sockaddr_in my_addr;

    //初始化 sockaddr_in 类型
    bzero(&my_addr, sizeof(my_addr));                       //将&addr 中的前
sizeof（addr）字节置为 0，包括'\0'
    my_addr.sin_family = AF_INET;                           //AF_INET 代表 TCP
/IP 协议
    my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");       //将点间隔地址转换
为网络字节顺序
    my_addr.sin_port = htons(8989);                         //转换为网络字节顺
序

    while (1)
    {
        printf("ftp>");
        bzero(input, MAXDATASIZE);
        if (fgets(input, MAXDATASIZE, stdin) == NULL)
        {
```

```
        printf("读取错误!\n");
        return -1;
    }
    n = split(input, &commd); //分割用户的输入，获得命令和参数

    //以下是对用户输入命令的判断，从而执行相应的操作
    if (strcmp(commd.commd, "ls") == 0) //处理 ls 命令
    {
        if (n == 2)
            commd_ls(my_addr, commd.arg);
        else
            commd_ls(my_addr, "./");
    }
    else if (strcmp(commd.commd, "cd") == 0) //处理 cd 命令
        commd_cd(my_addr, commd.arg);
    else if (strcmp(commd.commd, "pwd") == 0) //处理 pwd 命令
        commd_pwd(my_addr);
    else if (strcmp(commd.commd, "get") == 0) //处理 get 命令
        commd_get(my_addr, commd.arg);
    else if (strcmp(commd.commd, "rm") == 0) //处理 rm 命令
        commd_rm(my_addr, commd.arg);
    else if (strcmp(commd.commd, "lls") == 0) //处理 lls 命令
    {
        if (n == 2)
            commd_lls(commd.arg);
        else
            commd_lls("./");
    }
    else if (strcmp(commd.commd, "lcd") == 0) //处理 lcd 命令
        commd_lcd(commd.arg);
    else if (strcmp(commd.commd, "lpwd") == 0) //处理 lpwd 命令
        commd_lpwd();
    else if (strcmp(commd.commd, "put") == 0) //处理 put 命令
        commd_put(my_addr, commd.arg);
    else if (strcmp(commd.commd, "lrm") == 0) //处理 lrm 命令
        commd_lrm(commd.arg);
    else if (strcmp(commd.commd, "help") == 0) //处理 help 命令
        commd_help();
    else if (strcmp(commd.commd, "exit") == 0) //处理 exit 命令
    {
        exit(0);
        printf("谢谢您使用 ftp 模拟系统\n");
    }
    else //输入命令错误的处理方式
```

```
            printf("输入的命令错误，请重新输入!\n");
    }
    return 0;
}

//列出客户端当前目录下的文件和目录
int commd_lls(char *arg)
{
    int fd, err;
    char buf[MAXBUF], commd_str[100];
    if (arg != NULL)
        sprintf(commd_str, "ls -la %s > /tmp/buf", arg);
    err = system(commd_str);
    if (err < 0)
        return -1;
    fd = open("/tmp/buf", O_RDONLY);
    if (fd < 0)
        return -1;
    read(fd, buf, sizeof(buf));
    close(fd);
    remove("rm /tmp/buf");
    printf("%s\n", buf);
    return 0;
}

int commd_lcd(char *arg) //切换客户端当前所在的目录
{
    int err;
    err = chdir(arg);
    if (err < 0)
        return -1;
    return 0;
}

int commd_lpwd() //打印客户端当前所在目录完整路径名
{
    char buf[100];
    getcwd(buf, sizeof(buf));
    printf("%s\n", buf);
    return 0;
}
int commd_lrm(char *arg) //删除当前所在目录下的指定文件
{
    int err;
```

```
    err = remove(arg);
    if (err < 0)
        return -1;
    return 0;
}

// 实现文件下载功能
void commd_get(struct sockaddr_in my_addr, char *arg)
{
    int fd;
    int sockfd;
    char buffer[N], commd[N];
    int nbytes;
    //创建套接字，并进行错误检测
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        exit(1);
    }
    //connect 函数用于实现客户端与服务端的连接,此处还进行了错误检测
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        exit(1);
    }
    sprintf(commd, "get %s", arg);
    //通过 write 函数向服务端发送数据
    if (write(sockfd, commd, N) < 0)
    {
        printf("Write Error!At commd_get 1\n");
        exit(1);
    }
    //利用 read 函数来接受服务器发来的数据
    if (read(sockfd, buffer, N) < 0)
    {
        printf("Read Error!At commd_get 1\n");
        return;
    }
    //用于检测服务器端文件是否打开成功
    if (buffer[0] == 'N')
    {
        close(sockfd);
        printf("Can't Open The File!\n");
```

```
        return;
    }
    //open 函数创建一个文件，文件地址为(commd+4)，该地址从命令行输入获取
    if ((fd = open(arg, O_WRONLY | O_CREAT | O_TRUNC, 0644)) < 0)
    {
        printf("Open Error!\n");
        exit(1);
    }
    //read 函数从套接字中获取 N 字节数据放入 buffer 中，返回值为读取的字节数
    while ((nbytes = read(sockfd, buffer, N)) > 0)
    {
        //write 函数将 buffer 中的内容读取出来写入 fd 所指向的文件，返回值为实
际写入的字节数
        if (write(fd, buffer, nbytes) < 0)
        {
            printf("Write Error!At commd_get 2");
        }
    }
    close(fd);
    close(sockfd);
    return;
}

//显示服务端文件列表
void commd_ls(struct sockaddr_in my_addr, char *arg)
{
    int sockfd;
    char commd[N];
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        return;
    }
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        return;
    }
    sprintf(commd, "ls %s", arg);
    if (write(sockfd, commd, N) < 0)
    {
        printf("写入错误\n");
```

```
        exit(1);
    }
    //利用 read 函数来接受服务器发来的数据
    while (read(sockfd, commd, N) > 0)
        printf("%s ", commd);
    printf("\n");
    close(sockfd);
    return;
}

void commd_cd(struct sockaddr_in my_addr, char *arg)
{
    int sockfd;
    char commd[N];
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        return;
    }
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        return;
    }
    sprintf(commd, "cd %s", arg);
    if (write(sockfd, commd, N) < 0)
    {
        printf("写入错误\n");
        return;
    }
    //利用 read 函数来接受服务器发来的数据
    read(sockfd, commd, N);
    if (strncmp(commd, "0", 1) == 0)
        printf("切换成功\n");
    else
        printf("切换失败!\n");
}

void commd_pwd(struct sockaddr_in my_addr)
{
    int sockfd;
    char commd[N];
```

```
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        return;
    }
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        return;
    }
    sprintf(commd, "pwd");
    if (write(sockfd, commd, N) < 0)
    {
        printf("写入错误\n");
        exit(1);
    }
    //利用 read 函数来接受服务器发来的数据
    read(sockfd, commd, N);
    printf("%s\n", commd);
}

//实现上传文件的功能
void commd_put(struct sockaddr_in my_addr, char *arg)
{
    int fd;
    int sockfd;
    char buffer[N], commd[N];
    int nbytes;
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        exit(1);
    }
    //客户端与服务端连接
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        exit(1);
    }
    sprintf(commd, "put %s", arg);
```

```
    //从 commd 中读取 N 字节数据，写入套接字中
    if (write(sockfd, commd, N) < 0)
    {
        printf("Wrtie Error!At commd_put 1\n");
        exit(1);
    }
    //open 函数从(commd+4)中，读取文件路径，以只读的方式打开
    if ((fd = open(arg, O_RDONLY)) < 0)
    {
        printf("Open Error!\n");
        exit(1);
    }
    //从 fd 指向的文件中读取 N 个字节数据
    while ((nbytes = read(fd, buffer, N)) > 0)
    {
        //从 buffer 中读取 nbytes 字节数据，写入套接字中
        if (write(sockfd, buffer, nbytes) < 0)
        {
            printf("Write Error!At commd_put 2");
        }
    }
    close(fd);
    close(sockfd);
    return;
}

void commd_rm(struct sockaddr_in my_addr, char *arg)
{
    int sockfd;
    char commd[N];
    //创建套接字
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("Socket Error!\n");
        return;
    }
    if (connect(sockfd, (struct sockaddr *)&my_addr, sizeof(my_addr)) <
 0)
    {
        printf("Connect Error!\n");
        return;
    }
    sprintf(commd, "rm -rf %s", arg);
    if (write(sockfd, commd, N) < 0)
```

```
    {
        printf("写入错误\n");
        return;
    }
    //利用 read 函数来接受服务器发来的数据
    read(sockfd, commd, N);
    if (strncmp(commd, "0", 1) == 0)
        printf("删除成功!\n");
    else
        printf("删除失败!\n");
    close(sockfd);
    return;
}

int split(char *str, Commd *commd)
{
    int n = 0;
    char *ret;
    char dataList[10][20];
    ret = strtok(str, " ");
    while (ret != NULL)
    {
        strcpy(dataList[n++], ret);
        ret = strtok(NULL, " ");
    }
    for (int i = 0; i < 2; i++)
        if (dataList[i][strlen(dataList[i]) - 1] == '\n')
            dataList[i][strlen(dataList[i]) - 1] = '\0';
    strcpy(commd->commd, dataList[0]);
    strcpy(commd->arg, dataList[1]);
    return n;
}

void commd_help()
{
    printf("\t\t\t**********************ftp 命令一览
**************************************\n");
    printf("\t\t\t*
                *\n");
    printf("\t\t\t*        <1>.ls      <文件或目录>        列出服务端当前
的文件和目录    *\n");
    printf("\t\t\t*        <2>.lls     <文件或目录>        列出客户端当前
的文件和目录    *\n");
```

```
    printf("\t\t\t*        <3>.rm        <文件或目录>          删除服务端文
件                *\n");
    printf("\t\t\t*        <4>.lrm     <文件或目录>          删除客户端文
件                *\n");
    printf("\t\t\t*        <5>.cd         <目录>              切换服务端目
录                *\n");
    printf("\t\t\t*        <6>.lcd        <目录>              切换客户端目
录                *\n");
    printf("\t\t\t*        <7>.get        <文件>              下载服务端文
件                *\n");
    printf("\t\t\t*        <8>.put        <文件>              将客户端文件上
传到服务端       *\n");
    printf("\t\t\t*        <9>.pwd                      打印当前所在服务端目
录         *\n");
    printf("\t\t\t*        <10>.lpwd                      打印当前客户端所
在目录         *\n");
    printf("\t\t\t*        <12>.help                      帮助命令，查看所
有命令用法     *\n");
    printf("\t\t\t*                                                欢迎
使用 ftp 模拟系统！  *\n");
    printf("\t\t\t**************************************************
*****************\n");
}
```