

课程设计任务书

专业：计算机科学与技术 学号：2153626 学生姓名（签名）：

设计题目：一个简单 **FTP** 服务器的实现

一、设计实验条件

1208 实验室

二、设计任务及要求

设计要求：任选一门自己熟悉的程序设计语言，利用Socket 网络编程机制实现一个简单 FTP 服务器。

要求实现的功能包括：

上传、下载、选择数据传输模式，改变目录等，并给出相应的提示。

三、设计报告的内容

1. 设计题目与设计任务

1.1 设计题目

一个简单 FTP 服务器的实现。

1.2 设计任务

任选一门自己熟悉的程序设计语言，利用 Socket 网络编程机制实现一个简单 FTP 服务器。要求实现的功能包括：上传、下载、选择数据传输模式，改变目录等，并给出相应的提示。

2. 前言

2.1 FTP 协议

Ftp 服务是最常用的网络服务之一，虽然在 www 风行的今天，Ftp 已经远不如以前使用得广泛，但是在许多大学等科研单位，Ftp 仍然是最常用的文件交换方式。

FTP（File Transfer Protocol，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。Ftp 协议是基于 TCP 协议的，因此，在一个Ftp 会话开始前，客户

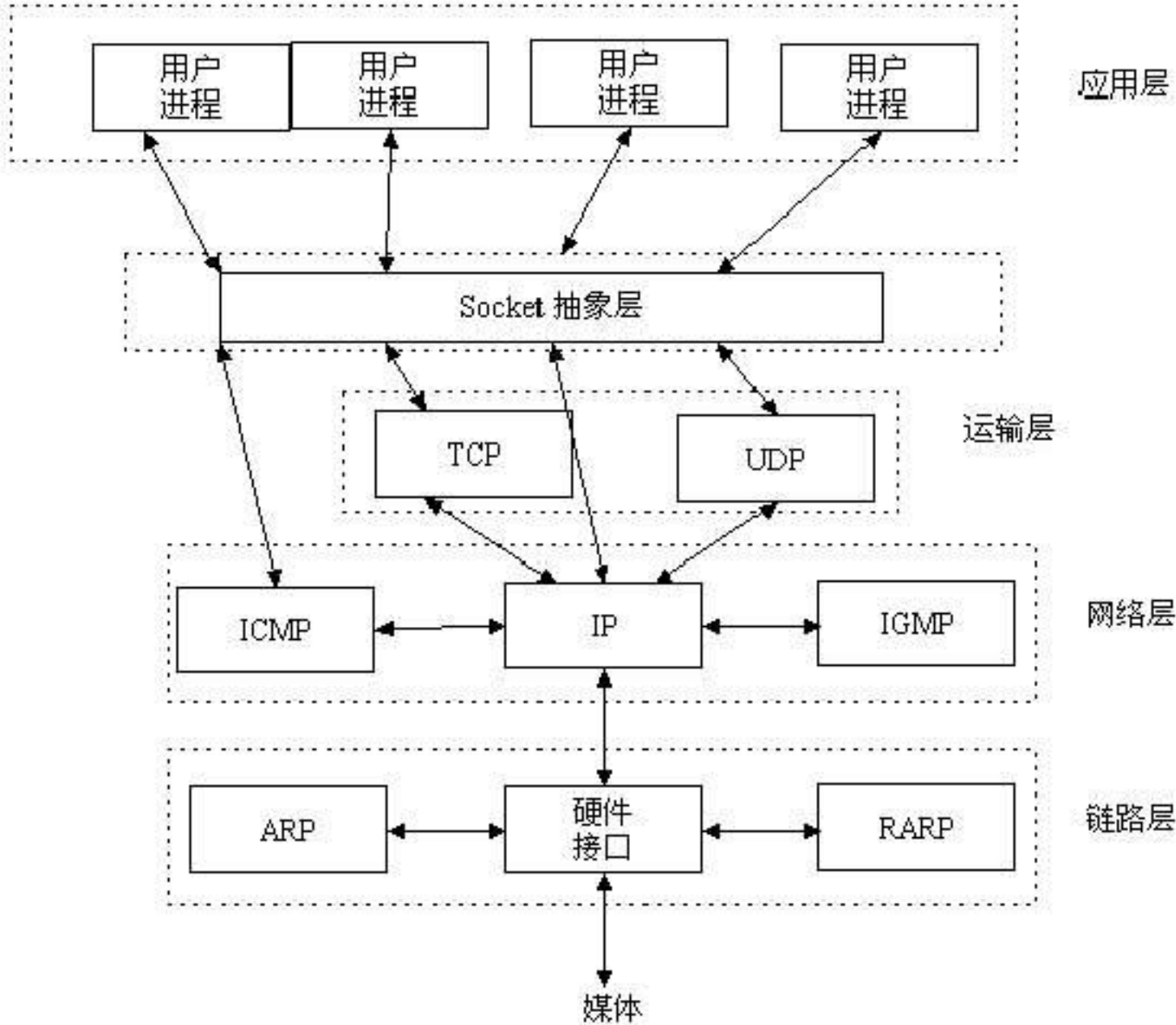
端和服务端必须首先建立一个 TCP 连接，这个 TCP 连接通常被称作控制连接，客户端通过此连接向服务器发送 FTP 命令，服务器处理命令后，将返回一个响应码。

一个 Ftp 会话过程中，始终有一个控制连接，如果客户端请求文件，则会有一个数据连接，但 FTP 协议规定：只要关闭了控制连接，数据连接（如果有）也必须关闭。

2.2 Socket

Socket 起源于 Unix，而 Unix/Linux 基本哲学之一就是“一切皆文件”，都可以用“打开open ->读写 write/read ->关闭 close”模式来操作。Socket 就是该模式的一个实现，Socket 即是一种特殊的文件，一些 socket 函数就是对其进行的操作（读/写 IO、打开、关闭）。说白了Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，它是一组接口。在设计模式中，Socket 其实就是一个门面模式，它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。

Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层。



套接字 API 最初是作为 UNIX 操作系统的一部分而开发的，所以套接字 API 与系统的其他 I/O 设备集成在一起。特别是，当应用程序要为因特网通信而创建一

个套接字（socket）时，操作系统就返回一个小整数作为描述符（descriptor）来标识这个套接字。然后，应用程序以该描述符作为传递参数，通过调用函数来完成某种操作（例如通过网络传送数据或接收输入的数据）。

2.3 传输模式

FTP 的传输有两种方式：ASCII 传输模式和二进制数据传输模式。

ASCII 传输模式：假定用户正在拷贝的文件包含的简单ASCII 码文本，如果在远程机器上运行的是不同的操作系统，当文件传输时 ftp 通常会自动地调整文件的内容以便于把文件解释成另外那台计算机存储文本文件的格式。即 ASCII 模式下会转换文件，不能说是不同系统对回车换行解释不同，而是不同的系统有不同的行结束符。UNIX 系统下行结束符是一个字节，即十六进制的 0A，而Windows 的系统是两个字节，即十六进制的 0D0A，所以当你用 ASCII 方式从UNIX 的FTP Server下载文件到 Windows 系统上时(不管是二进制或者文本文件)，每检测到一个字节是 0A，就会自动插入一个 0D，所以如果你的文件是二进制文件，比如可执行文件、压缩包什么的，就肯定不能用了。如果你的文件就是UNIX 下的文本文件，你用ASCII 模式是正确的，要是误用了Binary 模式，你在Windows 上看这个文件是没有换行的，里面是一个个的黑方块。

二进制传输模式：在二进制传输中，保存文件的位序，以便原始和拷贝的是逐位一一对应的。即使目的地机器上包含位序列的文件是没意义的。例如，macintosh 以二进制方式传送可执行文件到 Windows 系统，在对方系统上，此文件不能执行。如果你在 ASCII 方式下传输二进制文件，即使不需要也仍会转译。这会使传输稍微变慢，也会损坏数据，使文件变得不能用。（在大多数计算机上，ASCII 方式一般假设每一字符的第一有效位无意义，因为 ASCII 字符组合不使用它。如果你传输二进制文件，所有的位都是重要的。）如果你知道这两台机器是同样的，则二进制方式对文本文件和数据文件都是有效的。

2.4 设计目的

21 世纪是网络的时代，是信息的时代，是多媒体的时代。Intertnet 技术的迅猛发展与普及，推动了世界范围的信息传输和信息交流。Internet 如此流行，其中 FTP 功不可没。成千上万的数据、软件分布在世界各地，有了 ftp，足不出户，就能轻而易举地得到想要的。FTP 文件传送服务，主要用于存放大量的网络公用软件，常用工具箱技术文档，以及一些著名 FTP 服务的景象，现在，已经有许

多互联网站点都建立了可供大众访问的资料库,这些资料都可以被通过 FTP 获取。建立匿名 FTP 服务器,可以使用户有机会接触到世界上最大的信息库,这个信息库是日积月累起来的,并且还在不断增长,永不关闭,涉及到几乎所有主题。而且,这一切是免费的。Internet 之所以能延续到今天,是因为人们使用通过标准协议提供标准服务的程序。匿名 FTP 是 Internet 网上发布软件的常用方法。Internet 上的很多程序是由个人创造和维护的,他们通过匿名 FTP 把它们分发给世界各地的人们。也可以找到电子杂志、用户网讨论组的档案、技术文件等等。

2.5 设计意义

互联网的一大特点是实现信息共享,其中文件传输是信息共享的十分重要的内容之一。FTP 是实现文件传输服务的最主要的规范,并且当需要考虑到文件传输安全、传输质量、访问控制等诸多因素时,FTP 服务器就成了解决文件传输问题的关键所在。在这种情形下,就需要有一个良好的 FTP 服务器平台来满足用户日益增长的服务需求。因此,研究 FTP 服务器相关技术及实现具有重要的意义。

3. 设计主体

3.1 系统简介

(1) 常用命令

FTP 的主要操作都是基于各种命令基础之上的。常用的命令有:

- (1) 设置传输模式,它包括 ASCII (文本) 和 BINARY 二进制模式。
- (2) 目录操作,改变或显示远程计算机的当前目录(cd、dir/ls 命令)。
- (3) 连接操作,open 命令用于建立同远程计算机的连接;close 命令用于关闭连接。
- (4) 发送操作,put 命令用于传送文件到远程计算机。
- (5) 获取操作,get 命令用于接收一个文件。

(2) 工作原理

服务器端先初始化 Socket,然后与端口绑定(bind),对端口进行监听(listen),调用 accept 阻塞,等待客户端连接。在这时如果有个客户端初始化一个 Socket,然后连接服务器(connect),如果连接成功,这时客户端与服务器端的连接就建立了。客户端发送数据请求,服务器端接收请求并处理请求,然后把回应数据发送给客户端,客户端读取数据,最后关闭连接,一次交互结束。

(3) 常用函数

accept() 函数

TCP 服务器监听到客户端请求之后，就会调用 `accept()` 函数取接收请求，这样连接就建立好了。之后就可以开始网络 I/O 操作了，即类同于普通文件的读写 I/O 操作。`accept` 默认会阻塞进程，直到有一个客户连接建立后返回，它返回的是已连接 `socket` 描述字(一个连接套接字)，它代表着一个网络已经存在的点点连接。

一个服务器通常通常仅仅只创建一个监听 `socket` 描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接 `socket` 描述字，当服务器完成了对某个客户的服务，相应的已连接 `socket` 描述字就被关闭。

read()、write() 等函数

`read` 函数是负责从 `fd` 中读取内容. 当读成功时，`read` 返回实际所读的字节数，如果返回的值是 0 表示已经读到文件的结束了，小于 0 表示出现了错误。如果错误为 `EINTR` 说明读是由中断引起的，如果是 `ECONNRESET` 表示网络连接出了问题。

`write` 函数将 `buf` 中的 `nbytes` 字节内容写入文件描述符 `fd`，成功时返回写的字节数。失败时返回-1，并设置 `errno` 变量。在网络程序中，当我们向套接字文件描述符写时有两种可能。1) `write` 的返回值大于 0，表示写了部分或者是全部的数据。2) 返回的值小于 0，此时出现了错误。我们要根据错误类型来处理。如果错误为 `EINTR` 表示在写的时候出现了中断错误。如果为 `EPIPE` 表示网络连接出现了问题(对方已经关闭了连接)。

close() 函数

在服务器与客户端建立连接之后，会进行一些读写操作，完成了读写操作就要关闭相应的 `socket` 描述字，好比操作完打开的文件要调用 `fclose` 关闭打开的文件。`close` 一个 TCP `socket` 的缺省行为时把该 `socket` 标记为以关闭，然后立即返回到调用进程，该描述字不能再由调用进程使用。

(4) Socket 中 TCP 的建立（三次握手）

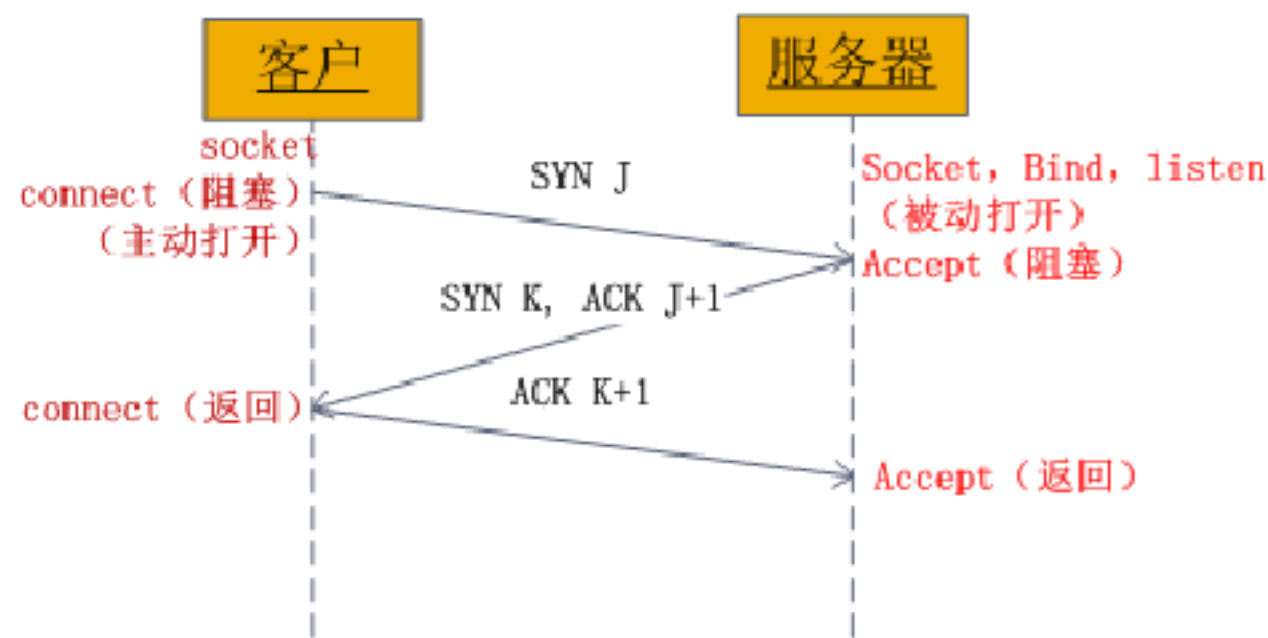
TCP 协议通过三个报文段完成连接的建立，这个过程称为三次握手(three-way handshake)，过程如下图所示。

第一次握手：建立连接时，客户端发送 `syn` 包 (`syn=j`) 到服务器，并进入 `SYN_SEND` 状态，等待服务器确认； `SYN`：同步序列编号 (Synchronize Sequence Numbers)。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN (ack=j+1)，同时自己也发送一个 SYN 包 (syn=k)，即 SYN+ACK 包，此时服务器进入SYN_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK(ack=k+1)，此包发送完毕，客户端和服务器进入 ESTABLISHED 状态，完成三次握手。

一个完整的三次握手也就是： 请求---应答---再次确认。



当客户端调用 `connect` 时，触发了连接请求，向服务器发送了SYN J 包，这时 `connect` 进入阻塞状态；服务器监听到连接请求，即收到 SYN J 包，调用 `accept` 函数接收请求向客户端发送 SYN K ， ACK J+1，这时 `accept` 进入阻塞状态；客户端收到服务器的 SYN K ， ACK J+1 之后，这时 `connect` 返回，并对 SYN K 进行确认；服务器收到 ACK K+1 时，`accept` 返回，至此三次握手完毕，连接建立。

3.2 功能设计

3.2.1 基本部分

Client.java (客户端)

成员变量：

```
String cmd = ""; // 从标准输入流接收字符串，放在 cmd 中
Socket s; //
BufferedReader br; // 传输数据
DataOutputStream dos; // 数据输出流
DataInputStream dis; // 数据输入流
```

client 方法

```
public client(String serName) {
```

```

catalogue();
try {
    boolean flag = true;
    while (flag) {
        s = new Socket(serName, 8888);
        // 从标准 IO 中获得输入的命令
        System.out.println("连接上!!");
        br = new BufferedReader(new InputStreamReader(System.in));
        // 将输入的命令放到 BufferedReader 类变量 br 中
        cmd = br.readLine(); // 将 br 的内容读出放到 cmd 中
        // 发送命令
        DataOutputStream dos = new DataOutputStream(new
BufferedOutputStream(s.getOutputStream())); // 向服务器发送相关命令, 如
get, put, cd, dir
        // 将字符集转换成字节序列
        byte[] buf = cmd.getBytes(); // 使用平台默认的字符集将此 String
解码为字节序列, 并将结果存储到一个新的字节数组中。
        // 返回: 结果字节数组

        dos.writeUTF(cmd); // 用 UTF 编码将一个字符串写入基础输入流, 即写
到服务端的输入流
        dos.flush(); // 清空此数据输出流
        dos.close(); // 关闭输出流
        s.close(); // 关闭 socket

        // 进行对于应得操作:
        if (cmd.equals("get"))
            get(serName);
        else if (cmd.equals("put"))
            put(serName);
        else if (cmd.equals("cd"))
            cd(serName);
        else if (cmd.equals("dir"))
            dir(serName);
        else if (cmd == "quit")
            flag = false;
    }
} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

```

```

    } finally {

        if (br != null) {
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (s != null) {
            try {
                s.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

主函数:

```

public static void main(String[] args) {
    new client("127.0.0.1")
}

```

Server.java (服务器端)

成员变量

```

ServerSocket ss = null; //服务器端 socket 对象
    // 服务器套接字等待请求通过网络传入。它基于该请求执行某些操作，然后可能向请求者返回结果。
    String dir = "";
    String cmd = "";
    DataOutputStream dos; // 数据输出流
    String direcFile = "";
    File rootDirectory;
    String shareFile;
    DataInputStream dis; // 数据输入流
    ArrayList<File> fileArrayList = new ArrayList<File>(); // 文件列表
    private String shareFileDirectory; //共享文件目录字符串

```

Server 方法

```

public server(String shareFileDirectory) {

```

```

    this.shareFileDirectory = shareFileDirectory;
    // 建立套接字
    try {
        ss = new ServerSocket(8888);//// 创建一个服务器端 Socket, 即 ss,
        指定绑定的端口, 并监听此端口
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    try {
        while (true) {
            boolean flag = true;
            while (flag) {
                Socket s = ss.accept();// 阻塞式, 等待客户端请求连接, 即接受
                客户端请求

                System.out.println("已有客户端连接");

                dis = new DataInputStream(new
                BufferedInputStream(s.getInputStream()));// BufferedInputStream, 利用缓冲
                区来提高读效率,

                // 从此套接字读取字节的输入流

                // BufferedInputStream(InputStream in) 参数 in 指定需要被
                装饰的输入流

                cmd = dis.readUTF();// 从 dis 输入流读取若干字节, 把它转换为
                采用 UTF-8 字符编码的字符串, 并将其放在 cmd String 变量里
                // UTF-8 对 ASCII 字符采用一个字节形式的编码, 对非 ASCII 字符则
                采用两个或两个以上字节形式的编码

                dis.close();// 关闭输入流
                s.close();// 关闭 socket

                System.out.println("输出读入的字符串: " + cmd);

                // 接受后放在 cmd 里用于判断:
                if (cmd.equals("get"))//调用 get 方法
                    get();
                else if (cmd.equals("put"))//调用 put 方法
                    put();
                else if (cmd.equals("cd"))//调用 cd 方法
                    cd();
                else if (cmd.equals("dir"))//调用 dir 方法
                    dir();
            }
        }
    } catch (IOException e) {

```

