# 计算机网络第三次实验报告——FTP 实验

黄家晖 2014011330

## 1 实验目标

1. 深入了解 FTP 的原理和协议细节;
2. 学会利用 Socket 接口设计简单的应用层协议;
3. 掌握 TCP/IP 网络应用程序的基本设计方法和实现细节。

## 2 实验原理

本实验要求在 Linux 系统上使用 Socket 编程技术实现一个简化的 FTP 服务器和客户端程序。FTP 协议由 RFC959 定义，基于 TCP 有连接传输。一个 FTP 会话包括指令和数据两个通路，需要建立两个 Socket 连接。指令通路为用户主动连接服务器，而数据通路的建立则分为主动模式和被动模式。

在主动模式下，客户端发送 PORT 命令让服务器连接自己的端口，而在被动模式下，服务器开放端口提供给用户进行连接。

## 3 程序设计

本次实验的服务器和客户端之间的通信协议依照 RFC959 标准进行实现，且实现了实验书中所要求的功能。其中服务端的命令根据 RFC 中对服务器的最小实现进行，并支持主动模式和被动模式及其切换。下面对每条指令的实现方式进行介绍。

**取远方的一个文件** 取远端文件的时候，如果采用被动模式，则客户端首先要打开数据端口并将端口信息发送到服务器。否则，采用主动模式，此时服务器发送自己开放的端口供客户端连接。

数据连接建立之后，即开始传送文件。考虑到大文件的传输需求，在实际传输文件的时候以 4MB 作为缓冲区，保证每个线程占用的内存空间不至于过大。每发送一个缓冲区的内容之前先传输 4 字节的整形数据用来标志接下来要传输的缓冲区的大小，保证传输大文件时信息不丢失。

**传给远方一个文件** 传送文件的逻辑实际上和接受文件相同，都需要先建立数据连接，然后进行数据传输，这里不在赘述。

**显示、改变远方的当前目录** 在服务器上为每一个 Socket 维持其当前在的工作目录。处理用户请求的时候进行相关相应即可。在 const.h 中实现了 DirUtil 类，能根据当前的工作目录变换到指定的工作目录。

**列出远方当前目录** 根据相关标准，显示目录的命令通过数据通路进行传输。而具体显示目录内容的方式则是调用 Linux 系统的 ls 命令。

# 4 思考题

**1** 使用两个连接而非一个连接简化了 FTP 的控制流程、加快了数据传输的效率并给予 FTP 更多的功能。

首先，使用单独的指令连接可以方便地对数据传输进行控制。假如数据在传输过程中需要终止，则仅需要通过指令连接发送相关指令即可，这其中也不会出现单个连接中的数据冲突。

第二，如果将数据和指令合并到一个连接中，则需要不断进行数据或指令的判断。例如在传送数据时需要判断何时文件传输终止，这个时候如果仅采用一个连接，则需要不断判断文件内容中是否含有文件终止符，同时判断是否为指令，加大了客户端和服务器 CPU 的开销。相反地，如果采用两个连接，由操作系统根据 Socket 编号进行内容分发，则无需考虑上述问题，加快数据传输效率。

最后，二者分离给予了 FTP 很好的并行性扩展，一条指令连接可以对应多条数据连接同时传输数据；同时，FTP 自身的功能也因此得到扩展，例如可以通过指令通道控制两台服务器主机建立数据连接实现服务器数据互传；而且，对于某些机器字长比较特殊的机器来说，建立适配的数据连接而非 TELNET 连接能进一步提升 FTP 的效率。

**2** 在主动模式下，客户端发送 PORT 命令让服务器连接自己的端口，而在被动模式下，服务器开放自己的端口提供给用户进行连接。由于服务器和客户端有着不同的安全策略和防火墙配置，这两种方式的设计即时为了适应这些不同的配置。

主动模式中如果用户处于 NAT 环境中，无法获得自己真实的 IP 和端口号，服务器无法建立连接，只能使用被动模式。然而被动模式使得服务器主动接受高位端口连接，容易受到攻击，所以安全性差。

**3** 每次使用 FTP 发送下载文件请求时，首先会通过控制连接发送控制信息，再重新建立数据连接，待双方确认之后，才能开始数据传输，真正数

据传输的开销占总开销比例很小，有很大部分的网络资源浪费，因此传输多个小文件的效率相比传输一个大文件效率要低很多。

一个比较好的解决方法是使用压缩工具提前将小文件合成为大文件，待下载完毕之后再进行解压。这项操作可以由 FTP 的客户端和服务器隐式完成，也可以由用户主动完成。

# 5 附录：核心部分代码

```
1  void ftp::Server::sendViaDataConnection(std::istream &
       istream) {
2
3      LOG(INFO) << "Start␣transferring␣data";
4
5      fileTrunk* ft = new fileTrunk();
6      ft->size = htonl(FILE_TRANSFER_BUFFER);
7      istream.seekg(0, istream.beg);
8
9      while (true) {
10         istream.read(ft->payload, FILE_TRANSFER_BUFFER
               );
11         if (istream.eof()) {
12             unsigned int sendSize = (unsigned int)
                   istream.gcount();
13             LOG(INFO) << sendSize;
14             ft->size = htonl(sendSize);
15             send(data_socket_fd, ft, sendSize + 4, 0);
16             break;
17         }
18         send(data_socket_fd, ft, FILE_TRANSFER_BUFFER
               + 4, 0);
19     }
20
21     delete ft;
22
23     LOG(INFO) << "Data␣transfer␣complete.";
24
25 }
26
27 void ftp::Server::receiveViaDataConnection(std::
       ostream &ostream) {
```

```
28
29      fileTrunk* ft = new fileTrunk();
30
31      unsigned int sizeToRead;
32      do {
33          recv(data_socket_fd, ft, 4, 0);
34          sizeToRead = ntohl(ft->size);
35
36          unsigned int leftSizeToRead = sizeToRead;
37          unsigned int payloadOffset = 0;
38
39          while (leftSizeToRead != 0) {
40              unsigned int realSizeRead = recv(
                    data_socket_fd,
41  ft->payload + payloadOffset, leftSizeToRead, 0);
42              leftSizeToRead -= realSizeRead;
43              payloadOffset += realSizeRead;
44          }
45
46          ostream.write(ft->payload, sizeToRead);
47      } while (sizeToRead == FILE_TRANSFER_BUFFER);
48
49      delete ft;
50  }
```

注：完整源代码请参阅附件中的代码包。