华南理工大学

South China University of Technology

网络通信原理实验报告

局域网内点对点文件传输的设计与实现

学　　院：电子与信息学院

专　　业：通信与信息系统

班　　级：15 级硕士 2 班

学生姓名：吴朝东

学　　号：201520108713

课程教师：冯穗力

华 南 理 工 大 学

二〇一六年六月

目录

1 实验综述

1.1 设计背景

现实生活中，通过网络进行文件传输是非常普遍的，人们通过这种方式节省了时间，也提高了工作效率。但是文件传输的效率取决于网络性能的优劣，如人们常用的QQ通讯工具可以实现实时文件传输，但是当文件较大或者网速较慢时，传输的效率就相当慢，因此考虑设计一款在局域网内可以高效传输大文件的应用，解决传统工具在特殊情况下效率低下的问题。

1.2 设计目标

文件传输是将一个文件或其中的一部分从一个计算机系统传到另一个计算机系统。将一个文件或其中的一部分从一个计算机系统传到另一个计算机系统。它可能把文件传输至另一计算机中去存储，或访问远程计算机上的文件，或把文件传输至另一计算机上去运行或处理，或把文件传输至打印机去打印。由于网路中各个计算机的文件系统往往不相同，因此，要建立全网公用的文件传输规则，称作文件传输协议（FTP）。

本次实验所设计的文件传输应用是基于TCP/IP协议的Winsock编程原理的，对于这种编程模型一般都是采用客户机/服务器（Client/Server）方式，该应用除了可以解决传统的文件传输需求，还可以实现在局域网内高效地传输大文件，设计目标是在极短的时间内完成文件的传输，并且保证文件不受损、传输过程中无差错。

1.3 实现功能

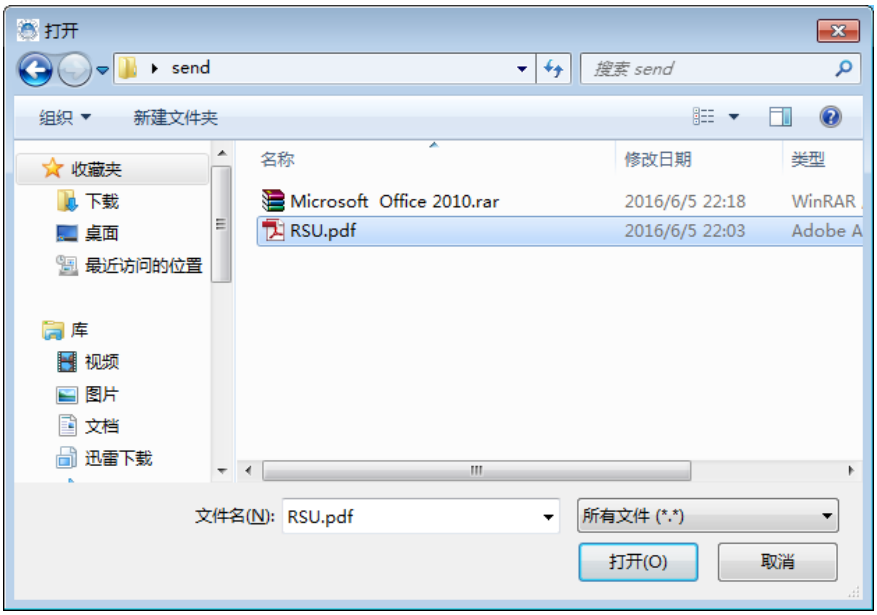
本次设计的核心思想是用MFC实现局域网内点对点的大文件传输，该设计实现局域网内的文件传输功能，包括服务器端程序和客户端程序两部分。客户端程序的功能：可连接到服务器，并将文件发送到服务器端和接受服务器端发送来的文件。服务器端程序的功能：负责相应客户端的消息和接收客户端发送来的文件。该设计可以在两台主机上建立连接，一台作为客户端，另一台作为服务器，客户端与服务器之间进行文件的传输，可以实现在局域网内高效地传输大文件，运行效率高且无差错。

1.4 实现效果

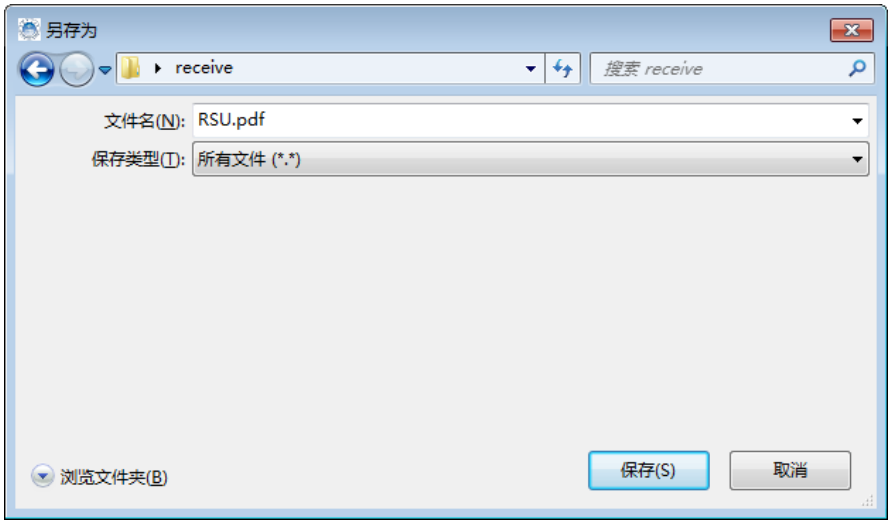
本次设计是基于MFC实现局域网内点对点的大文件传输，用C++作为编程语言，在VC6.0上进行调试、编译、测试等。测试时，在两台不同的主机上进行文件的传输，打开可执行文件，如图所示：

该应用可以自动获取主机的 IP 地址，该 IP 地址为 116.57.119.26，端口号设定为 9600，另一台主机也打开该应用，得到相应的 IP 地址,另一台主机的 IP 地址为 116.57.119.25。

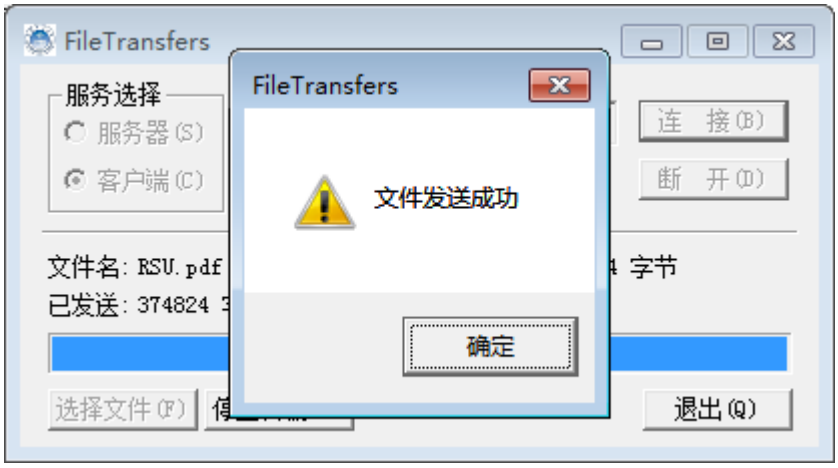
将第一台主机作为服务器，第二台主机作为客户端，建立连接，保证端口一致。当两台主机建立连接后，就可以进行文件传输。点击“选择文件”，选择要传输的文件，如下图：

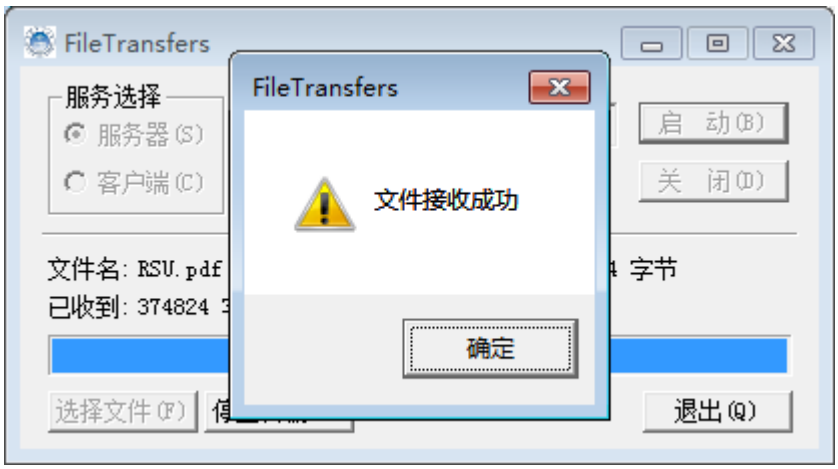


接收方需要选择接收文件存放的路径，如下图，将文件存放在 receive 文件夹里。

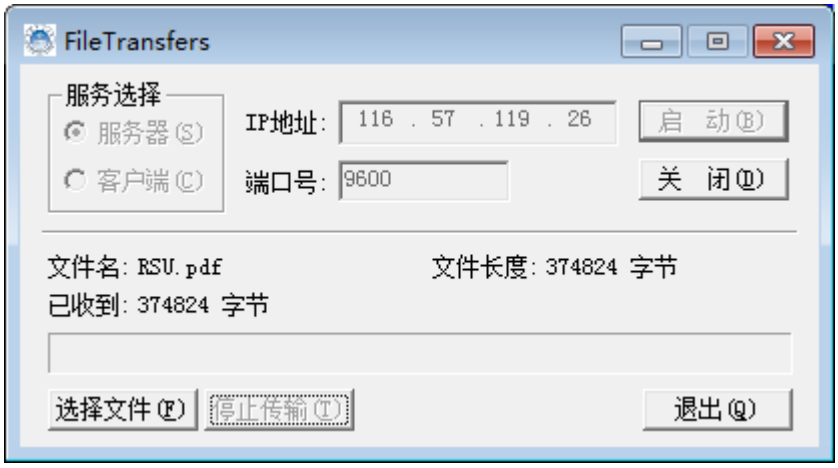


确认完成后文件开始传输，可以发现文件在极短的时间内完成传输，完成后显示 “文件发送成功”以及“文件接收成功”。



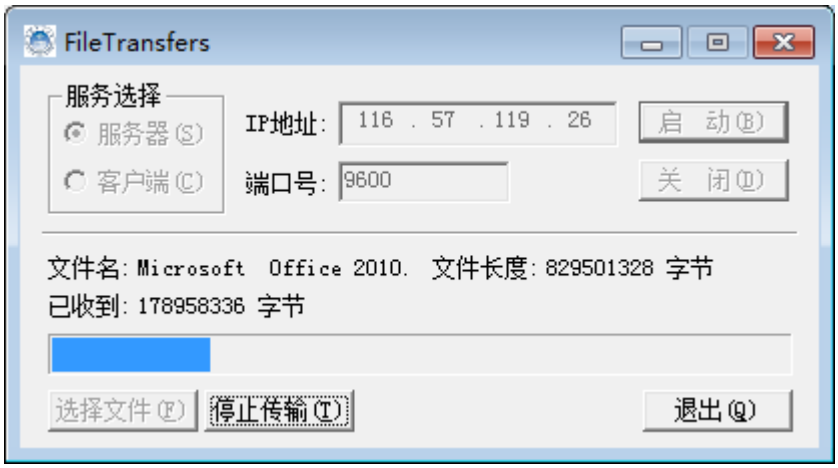


本次测试传输文件为 RSU.pdf，可以从界面看出文件长度为 374824 字节，传输时间很短且无差错。

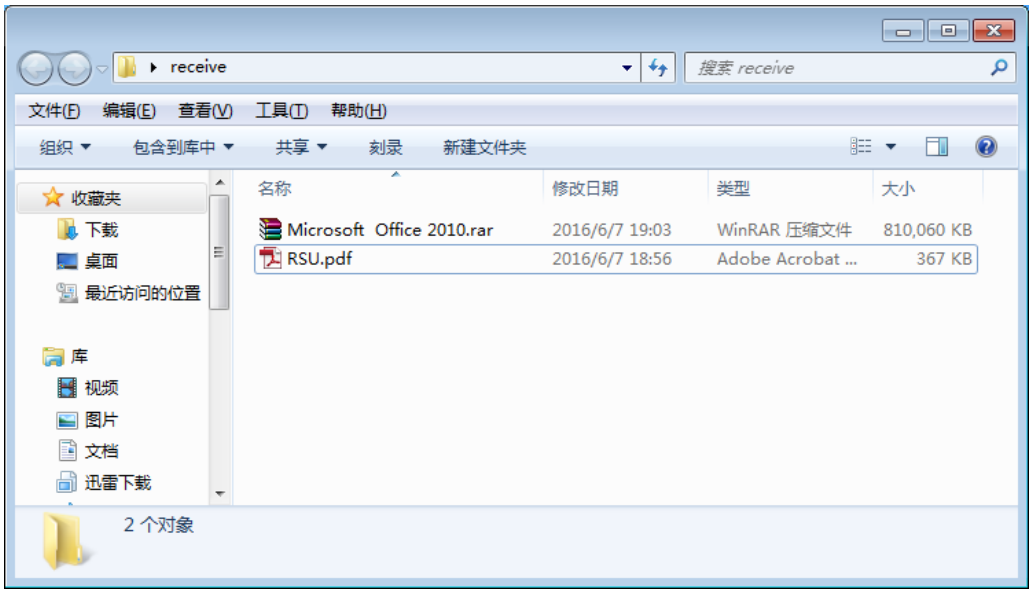


当传输大文件时，如传输 Microsoft Office 2010.rar 文件，文件长度为 829501328 字节，传输时间稍长，接近 1 分钟，传输过程稳定无差错。

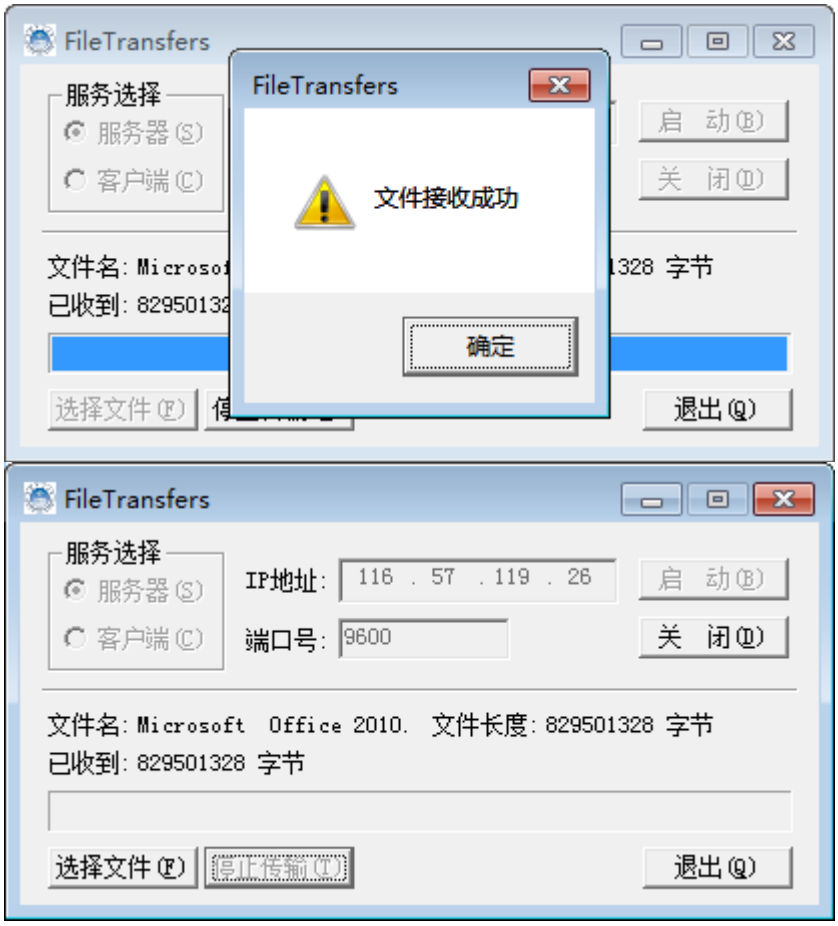
传输过程如图所示：



传输完毕后再 receive 文件夹中接收到该大文件。



相比传统的传输工具，该应用更高效且传输更稳定，不会出现传输过程中卡顿的现象。传输完成后结果图如下：



2 实验原理

2.1 网络传输协议介绍

在TCP/IP协议栈中，有两个高级协议是在网络应用程序编写中应该了解的，它们是"传输控制协议"（Transmission Control Protocol,简称TCP）和"用户数据报协议"（User Datagrm Protocol,简称UDP）。

TCP是面向连接的通信协议，TCP提供两台计算机之间的可靠无错的数据传输。应用程序利用TCP进行通信时，源和目标之间会建立一个虚拟连接。这个连接一但建立，两台计算机之间就可以把数据当作一个双向字节流进行交换。

UDP是无连接通信协议，UDP不保证可靠数据的传输，但能够向若干个目标发送数据，接收发自若干个源的数据。简单地说，如果一个主机向另外一台主机发送数据，这一数据就会立即发出，而不管另外一台主机是否已准备接收数据。如果另外一台主机收到了数据，它不会确认收到与否。

为了使两台计算机之间传输的文件数据不会丢失或发生错误，应该采用TCP协议。

下面对TCP/IP协议进行介绍：

TCP/IP是Transmission Control Protocol/Internet Protocol的简写，中文译名为传输控制协议/因特网互联协议，又叫网络通讯协议，这个协议是Internet最基本的协议、Internet国际互联网络的基础，简单地说，就是由网络层的IP协议和传输层的TCP协议组成的。TCP/IP 定义了电子设备（比如计算机）如何连入因特网，以及数据如何在它们之间传输的标准。TCP/IP是一个四层的分层体系结构。高层为传输控制协议，它负责聚集信息或把文件拆分成更小的包。低层是网际协议，它处理每个包的地址部分，使这些包正确的到达目的地。

从协议分层模型方面来讲，TCP/IP由四个层次组成：网络接口层、网络层、传输层、应用层。 具体介绍如下

（a） 网络接口层

物理层定义物理介质的各种特性：1、机械特性；2、电子特性；3、功能特性；4、规程特性。

数据链路层是负责接收IP数据报并通过网络发送之，或者从网络上接收物理帧，抽出IP数据报，交给IP层。

常见的接口层协议有： Ethernet 802.3、Token Ring 802.5、X.25、Frame relay、HDLC、PPP ATM 等。

(b) 网络层

网络层负责相邻计算机之间的通信。其功能包括三方面。

（1)、处理来自传输层的分组发送请求，收到请求后，将分组装入IP数据报，填充报头，选择去往信宿机的路径，然后将数据报发往适当的网络接口。

（2)、处理输入数据报：首先检查其合法性，然后进行寻径--假如该数据报已到达信宿机，则去掉报头，将剩下部分交给适当的传输协议；假如该数据报尚未到达信宿，则转发该数据报。

（3)、处理路径、流控、拥塞等问题。

(c) 传输层

传输层负责提供应用程序间的通信。其功能包括：一、格式化信息流；二、提供可靠传输。为实现后者，传输层协议规定接收端必须发回确认，并且假如分组丢失，必须重新发送。

传输层协议主要是：传输控制协议TCP(Transmission Control Protocol)和用户数据报协议UDP(User Datagram protocol)。

(d) 应用层

应用层负责向用户提供一组常用的应用程序，比如电子邮件、文件传输访问、远程登录等。远程登录TELNET使用TELNET协议提供在网络其它主机上注册的接口。TELNET会话提供了基于字符的虚拟终端。文件传输访问FTP使用FTP协议来提供网络内机器间的文件拷贝功能。

应用层一般是面向用户的服务。如FTP、TELNET、DNS、SMTP、POP3。

2.2 传输协议的具体实现

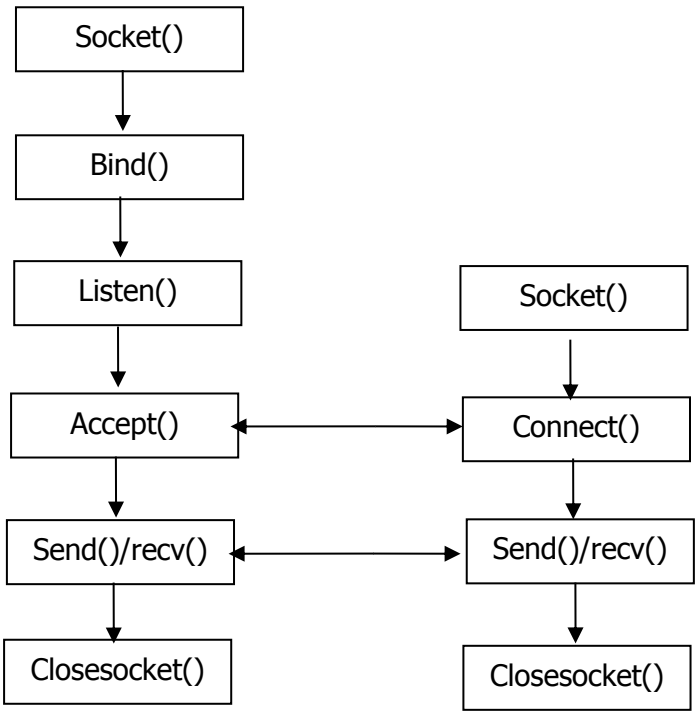
在 VC++中，网络协议的实现有以下几种方式：

(a)采用 WinSocket API 函数。

API 函数中提供了基本 Socket 的系统调用，具体实现方法为服务器端首先要调用 socket()函数建立一个流式套接字，用 bind()函数与本机的一个端口建立关联，继续调用 listen()函数将套接字置于被动的侦听方式以监听连接，然后调用 accept()函数进入等待状态之后才可以接收来自客户端的请求，一旦接收到客户端通过 connect 发出的连接请求，accept 将返回一个新的套接字描述符。通过此套接字描述符调用 send()或 recv()函数即可与客户端进行数据收发。待数据传送完成，服务器客户端调用 closesocket()关闭套接字。

该方法在编程过程中需要注意 socket 连接的整个过程，编程工作量大，编程效率低，单却可以加深对网络协议的认识。

程序流程示意图如下：



(b) 采用 VC++中提供的 MFC 类，CAsyncSocket 或 CSocket.

两个类都对 WinSocket API 进行了封装，CSocket 对它的封装比 CAsyncSocket 更深，使得对于从未接触过 WinSockets API 的编程程序员，也能够编写网络程序。

而本次设计也是采用了 CSocket 类进行编程。

2.3 客户/服务器模式介绍

客户端（Client）：

客户端或称为用户端，是指与服务器相对应，为客户提供本地服务的程序。一般安装在普通的客户机上，需要与服务端互相配合运行。因特网发展以后，较常用的用户端包括了如万维网使用的网页浏览器，收寄电子邮件时的电子邮件客户端，以及即时通讯的客户端软件等。

服务器（Server）：

服务器指一个管理资源并为用户提供服务的计算机软件，通常分为文件服务器、数据库服务器和应用程序服务器。运行以上软件的计算机或计算机系统也被称为服务器。相对于普通 PC 来说，服务器在稳定性、安全性、性能等方面都要求更高，因此 CPU、芯片组、内存、磁盘系统、网络等硬件和普通 PC 有所不同。

客户/服务器模式在操作过程中采取的是主动请求方式。

（a）服务器方的具体操作步骤如下：

（1）、首先服务器方要先启动，并根据请求提供相应服务。

（2）、打开一个通信通道并告知本地主机，它愿意在某一端口上接收客户请求。

（3）、等待客户请求到达该端口。

（4）、接收到重复请求，处理该请求并发送应答信号。接收到并发服务请求，要激活一新进程来处理这个客户请求。新进程处理此客户请求，并不需要对其他请求做出应答。服务完成后，关闭此进程与客户的通信链接，并终止该进程。

（5）、返回第二步，等待另一客户请求。

（6）、关闭服务器。

（b）客户方的主要操作步骤如下：

（1）、打开一通信通道，并链接到服务器所在主机指定端口。

（2）、向服务器发服务请求报文，等待并接收应答；继续提出请求。

（3）、请求结束后关闭通信通道并终止。

2.4 编程环境介绍

在 Microsoft Visual C++ 6.0 开发环境 Developer Studio 是在 Windows 98/2000/XP/7 环境下运行的一套集成工具，由文本编辑、资源编辑器、项目建立工具、优化编辑器、增量连接器、源代码浏览器、集成调试器等组成。
Visual C++ 6.0 的软、硬件配置需求如下：

（1）操作系统：Windows 98、Windows2000、Windows XP 或者 Windows 7 及更高版本。

（2）内存：根据不同操作系统设定，最好在 128M 以上。

（3）硬盘：最小需要 300MB 以上空间。

（4）CD-ROM 驱动器(用于联机信息)。

3 总体设计

3.1 总体设计思路

本程序的使用者为局域网用户。程序实现的主要功能是局域网的常见格式的文件的传输。为了实现局域网文件传输，本次设计具体可分为 4 部分：

（1） 文件

主要功能包括对文件名、文件路径以及文件大小的设置与获取。

（2） 服务端

提供对方与本地连接的套接字，响应对方连接的请求，通过网络通信，处理接收文件的过程。

（3） 客户端

通过 IP 地址和端口号向目标主机发送连接请求，通过网络通信，处理发送文件的过程。

（4） 用户界面

给用户提供友好的图形化的界面，响应用户的操作。包括与目标主机的连接、发送文件、选择文件、接收文件、消息提示等几部分。

3.2 总体通信流程

文件传输需要经过两个步骤，一是网络的链接；二是文件的传输。

本文提出的方案主要是在满足局域网内链路状态良好，ip 地址已知的条件下的文件传输。

主要步骤如下：

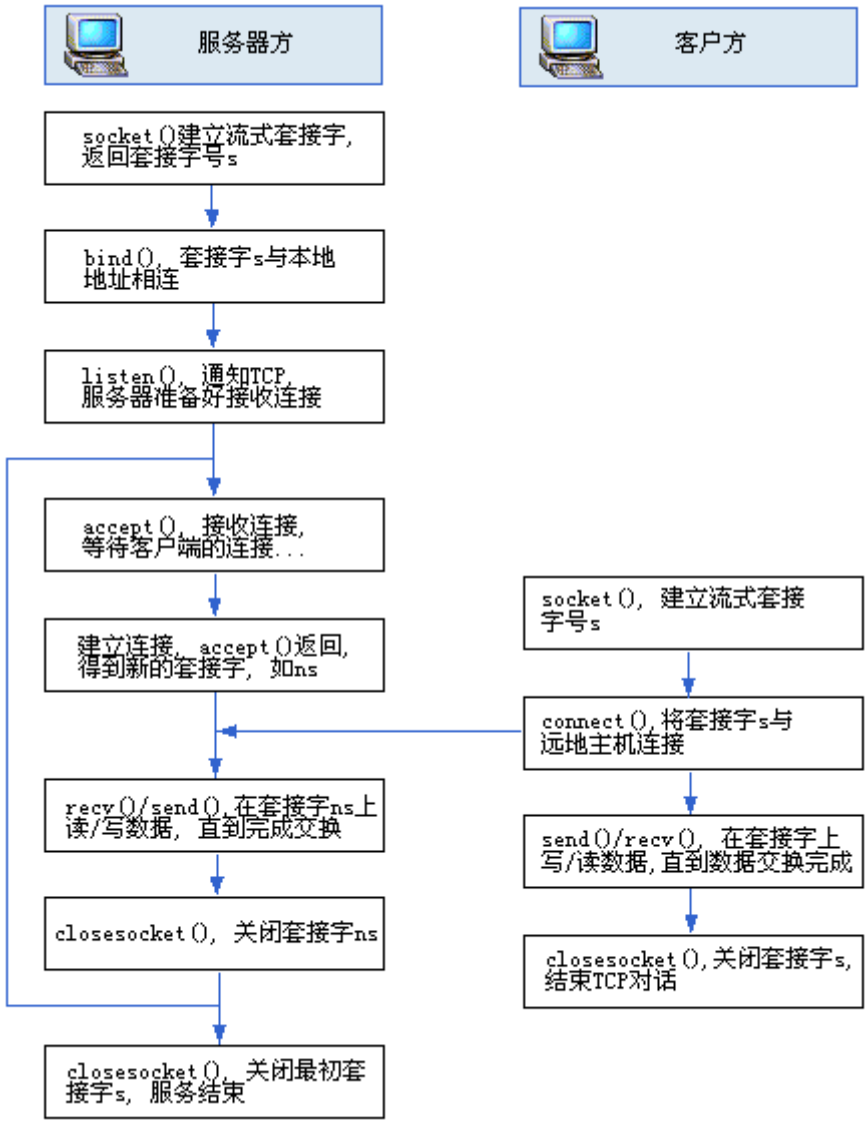
（1）运行软件，建立服务器；

（2）打开所要发送的文件，使其处于准备状态；

（3）输入 IP 地址、端口号，链接服务器；

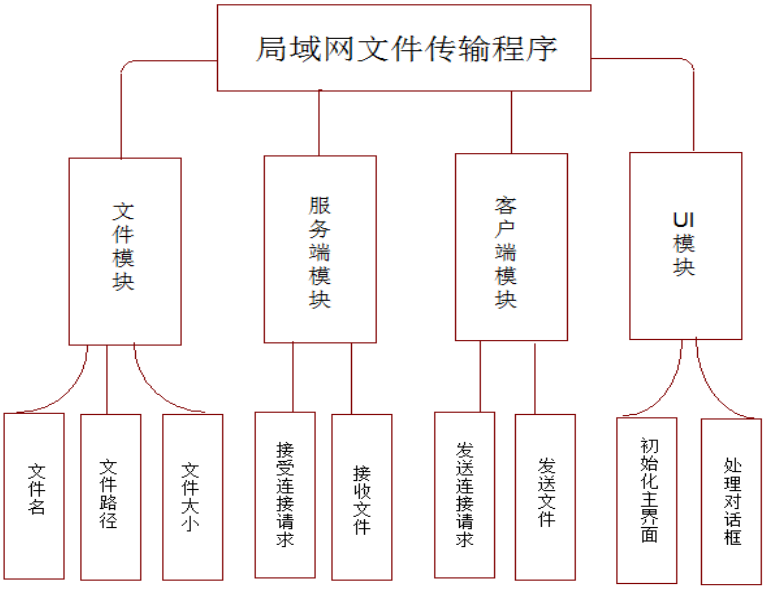
（4）发送文件。

具体的通信流程图如下所示：

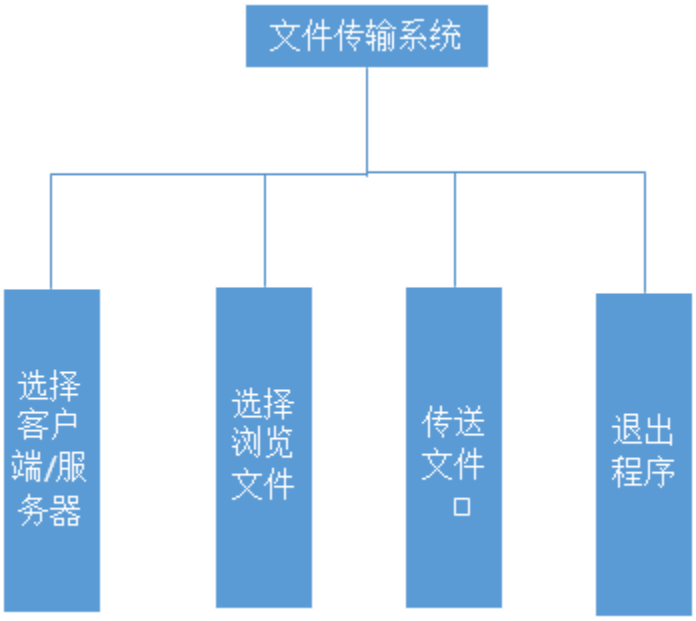


3.3 总体结构框架

局域网文件传输程序分为文件模块、服务端模块、客户端模块和 UI 模块，其功能结构图如图所示。

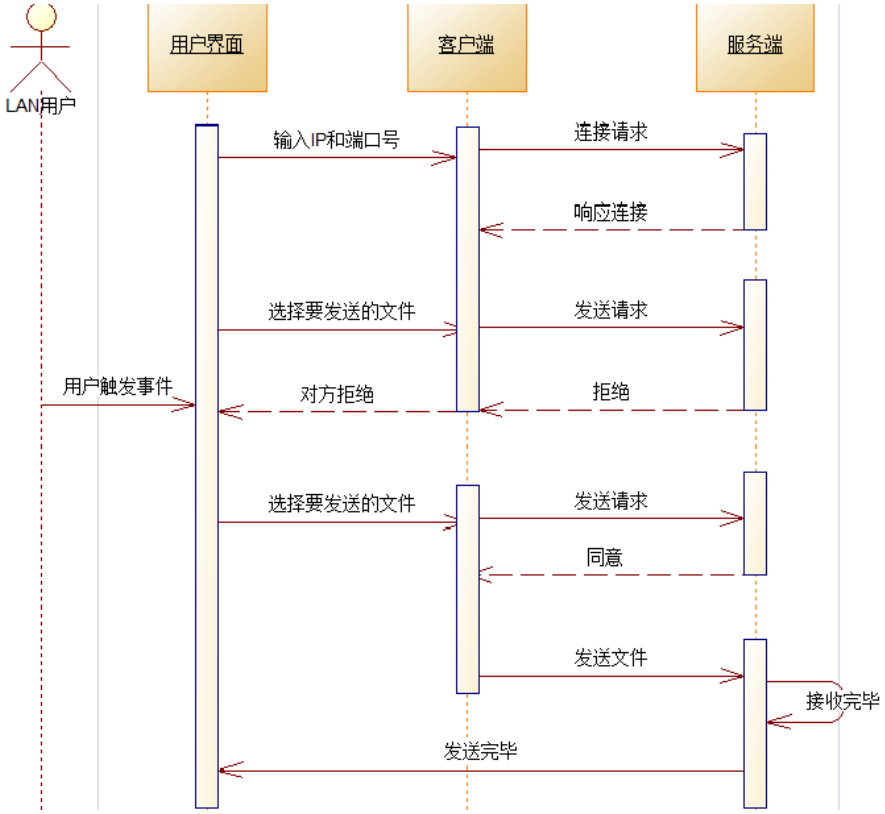


总体的文件传输系统运行时需要从客户端/服务器部分开始，先将两者建立连接，再选择要传输的文件，之后进行传输，传输完毕后即可退出程序，大致过程如下框图所示：



具体的请求/相应过程可以从下图中清晰看出，首先用户通过用户界面触发事件，客户端首先发起连接请求，等待服务器响应，若服务器在较长的时间内无响应，则客户端的请求自动取消，若服务器响应连接，则双方建立连接，客户端选择要发送的文件，等待服务器响应，若服务器在较长的时间内无响应，则客户端的请求自动取消，若服务器响应同意接受文件，则文件开始传输，在较短的时间内完成传输，实现了一次文件的成功传输。若仍然需要传输文件，则重复同样的步骤，若无需传输文件，则退出应用程序，此时自动断开连接。

具体的事件时序图如下所示：



4 具体实现

4.1 实现的核心思想

在通信的时候主要可以分为两个部分，一个部分是用于传送控制信息，例如发送文件的请求，文件的名称、大小等，由于这方面的数据量比较小，为了方便起见本次设计采用了 MFC 所提供的 CSocket 类的串行化技术来实现；而另一个部分就是文件的传输部分，对于文件的传输，由于数据量相对来说比较大，所以这里分别写了两个线程，一个用于发送，一个用于接收。这两个部分在具体实现的时候分别建立有自己的套接字（Socket）。

WinSock 编程分为同步(Sync)和异步（Async）。同步方式指的是发送方不等接收方响应，便接着发下一个数据包得通信方式。同步方式逻辑清晰，编程专注于应用，在抢先式的多任务操作系统中(Win NT、Win 2000)，采用多线程方式效率基本达到异步方式的水平。而异步指发送方发出数据以后，等收到接收方发回的响应，才发出下一个数据包的通信方式。

阻塞套接字是指执行此套接字的网络调用时，直到成功才返回，否则一直阻塞在此网路调用上，比如调用 recv()函数读取网路缓冲区中的数据，如果没有数据到达，将一直用 recv()这个函数调用上，知道读到一些数据，此函数才返回；而非阻塞套接字是指此套接字的网络调用时，不管是否执行成功，都立即返回。比如调用 recv()函数读取网络缓冲区中的数据，不管是否读到数据都

立即返回，而不会一直挂在此函数调用上。在实际 Windows 网络通信软件中，异步非阻塞套接字是用的最多的。平常所用的 C/S 结构软件用到的就是异步非阻塞模式的。

Windows 提供了一系列的 API 来支持 Sockets，主要包括两类，一类是 berkeley 类型的函数，另一类是已经认可的作为 Windows Sockets 2 的一部分的 Windows 特殊的扩展函数。下面对一些常用的 Windows Sockets API 函数进行简单说明：

(1)socket()：创建一个 Socket。

所有的通信在建立连接之前都要创建一个 Socket，该函数的功能与文件操作中的 fopen 类似。

(2)bind()：为创建的 Socket 指定通讯对象。

成功建立 Socket 之后，就应该选定通信的对象。首先是自己的程序需要与网络上的哪台计算机通话；其次，在多任务的系统下，该台计算机可能会有几个程序在工作，必须指出与那个程序通信。前者可以通过 Intenet 的网络 IP 地址来确定，而后者就需要指出端口号，用端口号来表示同一台计算机上不同的应用程序，可以从 0-65536 之间任选，不同功能的应用程序使用不同的端口号，这样一台计算机上可以有几个程序同时使用一个 IP 地址而互不干扰。

(3)listen()：设置等待连接状态。

对于服务器的程序，当申请到 Socket，并制定通信对象为 INADDR-ANY 之后，就应该等待一个客户机的程序来要求连接。而 listen()就是把一个 Socket 设置这种状态的函数。

(4)accept()：接收请求连接。

当没有连接请求时，对于阻塞方式，就进入等待状态，直至有一个请求到达为止。

(5)connect()：主动提出请求连接。

以上的 bind()、listen()、和 accept()函数一般都用于服务程序，属于被动等待的函数；对于客户端，要主动提出连接请求，应使用 connect()函数。

(6)sent()/recv():发送、接受数据。

(7)connect()：直接通信。

(8)closesocket(SOCKET s):通信结束，关闭指定的 Socket。

4.2 软件设计

(一)建立一个新工程 FileTransfers

使用 MFC AppWizard (exe) 建立一个新项目 FileTransfers，选择基于对话框的应用，并在向导的第四步中，选择“Windows Socket”选项，其它步骤中都使用缺省值，然后按下“Finish”按钮，创建新的工程。

AppWizard 将自动创建如下的类。

类名	定义文件	实现文件
CAboutDlg	FileTransfers.cpp	FileTransfers.cpp
CFileTransfersApp	FileTransfers.h	FileTransfers.cpp
CFileTransfersDlg	FileTransfersDlg.h	FileTransfersDlg.cpp

(二)修改资源

（1）修改主对话框风格

修改 AppWizard 是创建的对话框模风格。点击位于 Dialog Properties 对话框上面的 Style 标签，然后设置风格属性。

（2）添加控件

在对话框中加入相应的控件。

□“服务选择”分组框(Group Box)。用于表明其中的两个单选按钮是属于一组的。它的标题(Caption)为“服务选择”，使用默认的 ID 值就行了。

□“服务器(S)”和“客户端(C)”单选按钮(Radio Button)。这两个单选按钮定位在“服务选择”分组框(Group Box)中，用于确定软件当前是服务器或是客户端。将“服务器(S)”按钮的 ID 设置为 IDC_RADIO_SERVER,“客户端(C)”按钮的 ID 设置为 IDC_RADIO_CLIENT，其它的属性分别进行设置。

□“IP 地址”IP 地址控件(IP Address)。当软件做为服务器端时，该控件所显示的是本地的 IP 地址；当软件做为客户端时，用于输入服务器的 IP 地址。其 ID 为 IDC_IPADDRESS，其余属性采用默认值。

□“端口号”编辑控件(Edit Box)。用于确定通讯的端口号。所有属性均按照默认值既可。

□对话框中部分静态文本控件(Text)的属性如下所示。

ID	标题(Caption)	用途
IDC_FILE_NAME	FileName	显示当前正在传输的文件名
IDC_FILE_SIZE	0 字节	显示当前正在传输的文件尺寸
IDC_RECEIVE_SIZE	0 字节	显示已经发送或接受的文件尺寸

□先删除对话框中原有的确定按钮，然后再添加和修改按钮的属性，最终对话框中按钮的属性如下所示。

ID	标题(Caption)	用途
IDC_BEGIN	启　动(&B)	启动服务程序/连接到服务器
IDC_DISCONNECT	关　闭(&D)	关闭服务程序/断开服务器连接
IDC_SELECT_FILE	选择文件(&F)	选择要发送的文件
IDC_STOP_TRANSFERS	停止传输(&T)	停止文件的传输
IDCANCEL	退出(&Q)	退出程序

（三）几个辅助类的介绍

（1）CMessage 类

在前面部分说过程序传送控制信息的时候采用的是 CSocket 类的串行化技术，这样一来使得发送和接收网络数据就像普通的数据串行化一样简单。因此封装一个可以串行化的消息类是必要的，后面将会看到有了这个类，消息的发送和接收只需使用流操作符对缓冲区进行存取就可以了。根据程序的需要，消息类 CMessage 的定义如下：

```
class CMessage : public CObject
{
public:
	void Serialize(CArchive& ar);
	CMessage();
	CMessage(int nType);
	CMessage(int nType, CString strFileName, DWORD dwFileSize);
	virtual ~CMessage();
public:
	int m_nType;
	CString m_strFileName;
	DWORD m_dwFileSize;
};
```

其中，m_nType 用于标识消息的类型；m_strFileName 为文件的名称；m_dwFileSize 为文件的大小。

为了方便使用，我对消息类的 CMessage 的构造函数进行了重载，CMessage()为默认的构造函数，如果只是发送一般的控制信息可以使用 CMessage(int nType)构造函数，当需要发送文件名及大小的时候可以使用 CMessage(int nType, CString strFileName, DWORD dwFileSize)构造函数，三个构造函数的源代码如下：

```
//默认的构造函数
CMessage::CMessage()
{
	m_nType = -1;
	m_strFileName = _T("");
	m_dwFileSize = 0;
}

//只需发送一般的控制信息是使用
```

```
CMessage::CMessage(int nType)
{
	m_nType = nType;
	m_strFileName = _T("");
	m_dwFileSize = 0;
}

//需要发送文件名及大小时使用
CMessage::CMessage(int nType, CString strFileName, DWORD dwFileSize)
{
	m_nType = nType;
	m_strFileName = strFileName;
	m_dwFileSize = dwFileSize;
}
```

重载 CObject 类中的 Serialize 函数，其源代码如下：

```
void CMessage::Serialize(CArchive &ar)
{
	if (ar.IsStoring())
	{
		ar << m_nType;
		ar << m_strFileName;
		ar << m_dwFileSize;
	}
	else
	{
		ar >> m_nType;
		ar >> m_strFileName;
		ar >> m_dwFileSize;
	}
}
```

在写这个类的时候，可以使用任何一个文本编(如记事本或是 UltraEdit 等)辑器进行编写，把类的定义保存在 Message.h 中，把类的实现保存在 Message.cpp 文件中。要注意的是这里还需要在 Message.cpp 文件的首部加上

#include "stdafx.h"和#include "Message.h"两行代码，将 stdafx.h 文件和 Message.h 文件包含进来。最后还需要把这个类加到工程中，先把 Message.h 和 Message.cpp 文件复制到工程目录下，然后在 VC 中通过 Project 菜单->Add To Project->Files 把这两个文件添加到工程中。

（2）CListenSocket 类

负责监听管理的套接字类 CListenSocket。使用 ClassView 或 ClassWizard 进行创建，如图 7 所示。

使用 ClassView 或手工加入如下的函数及成员变量的定义：

```
public:
    CListenSocket(CFileTransfersDlg* pdlgMain);
protected:
    CFileTransfersDlg* m_pdlgMain;
```

CListenSocket(CFileTransfersDlg* pdlgMain)为重载的构造函数；m_pdlgMain 为指向主对话框类 CFileTransfersDlg 的指针。添加完后还需要对类中的两个构造函数的内容进行修改以实现对类的初始化，其源代码如下：

```
CListenSocket::CListenSocket(CFileTransfersDlg* pdlgMain)
{
    m_pdlgMain = pdlgMain;
}
CListenSocket::CListenSocket()
{
    m_pdlgMain = NULL;
}
```

重载基类的 OnAccept 函数以使对来自客户的连接请求作出响应，OnAccept 函数的源代码如下：

```
void CListenSocket::OnAccept(int nErrorCode)
{
    m_pdlgMain->ProcessAccept();
    CSocket::OnAccept(nErrorCode);
}
```

当该套接字接收到客户的连接请求时，就调用 CFileTransfersDlg 对象的 ProcessAccept()函数进行处理。由于该类中使用到了 CFileTransfersDlg 类，因此在文件 ListenSocket.h 的首部还需加入如下头文件的包含语句：

```
#include "FileTransfersDlg.h"
```

（3）CClientSocket 类

该类用于连接的管理，其创建的方法与 CListenSocket 相似，其定义如下：

```
class CClientSocket : public CSocket
{
// Attributes
public:
// Operations
public:
	CClientSocket();
	virtual ~CClientSocket();
// Overrides
public:
	CSocketFile* m_pFile;
	CArchive* m_pArchiveIn;
	CArchive* m_pArchiveOut;
	void Init();
	void Abort();
	BOOL SendMsg(CMessage* pMsg);
	void ReceiveMsg(CMessage* pMsg);
	CClientSocket(CFileTransfersDlg* pdlgMain);
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CClientSocket)
	public:
	virtual void OnReceive(int nErrorCode);
	//}}AFX_VIRTUAL
	// Generated message map functions
	//{{AFX_MSG(CClientSocket)
		// NOTE - the ClassWizard will add and remove member
functions here.
	//}}AFX_MSG
// Implementation
protected:
	CFileTransfersDlg* m_pdlgMain;
```

```
};
```

在 CClientSocket 套接字类封装了串行化功能，这也就是说，该类封装了客户应用程序的大部分功能，因此可以认为该套接字类所管理的就是一个客户应用程序。其中，m_pFile 为一个 CSocketFile 类型的指针用于连接到一个 CSocket 对象；m_pArchiveIn 和 m_pArchiveOut 均为 CArchive 类型的指针，分别用于接受和发送。

两个构造函数的源代码如下：

```
CClientSocket::CClientSocket(CFileTransfersDlg* pdlgMain)
{
	m_pdlgMain = pdlgMain;
	m_pFile = NULL;
	m_pArchiveIn = NULL;
	m_pArchiveOut = NULL;
}
CClientSocket::CClientSocket()
{
	m_pdlgMain = NULL;
	m_pFile = NULL;
	m_pArchiveIn = NULL;
	m_pArchiveOut = NULL;
}
```

Init 成员函数用于串行化的初始化，其源代码如下：

```
void CClientSocket::Init()
{
	m_pFile = new CSocketFile(this);
	m_pArchiveIn = new CArchive(m_pFile,CArchive::load);
	m_pArchiveOut = new CArchive(m_pFile,CArchive::store);
}
```

Abort 成员函数用于对 m_pArchiveOut 指针进行释放，其源代码如下：

```
void CClientSocket::Abort()
{
	if (m_pArchiveOut != NULL)
	{
```

```
            m_pArchiveOut->Abort();
            delete m_pArchiveOut;
            m_pArchiveOut = NULL;
      }
}
```

SendMsg 成员函数用于发送信息，其源代码如下：

```
BOOL CClientSocket::SendMsg(CMessage *pMsg)
{
      if (m_pArchiveOut != NULL)
      {
            TRY
            {
                  //采用串行化技术进行信息的发送
                        pMsg->Serialize(*m_pArchiveOut);
                  m_pArchiveOut->Flush();
                        return TRUE;
            }
            CATCH(CFileException, e)
            {
                  m_pArchiveOut->Abort();
                  delete m_pArchiveOut;
                  m_pArchiveOut = NULL;
            }
            END_CATCH
      }
      return FALSE;
}
```

ReceiveMsg 成员函数用于接受信息，其源代码如下：

```
void CClientSocket::ReceiveMsg(CMessage *pMsg)
{
      //采用串行化技术进行信息的接收
        pMsg->Serialize(*m_pArchiveIn);
```

```
}
```

重载基类函数 OnReceive，使用此函数接受 Socket 连接另一端发送的信息，其源代码如下：

```
void CClientSocket::OnReceive(int nErrorCode)
{
      m_pdlgMain->ProcessReceive(this);
      CSocket::OnReceive(nErrorCode);
}
```

代码的作用是当有信息发送到时，调用主对话框类的 ProcessReceive 函数进行信息的接收。由于用到了 CMessage 和 CFileTransfersDlg 类，所以在 ClientSocket.h 文件首部还需加入如下的头文件包含：

```
#include "Message.h"
#include "FileTransfersDlg.h"
```

（四）两个线程

上面所定义的类只实现了控制信息的传送，此时还需要写两个线程 _SendThread 和_ListenThread，它们分别用于发送和接收文件。

（1）发送文件线程

```
UINT _SendThread(LPVOID lparam)
{
      CFileTransfersDlg* pDlg = (CFileTransfersDlg*) lparam;

      //创建套接字
        CSocket sockClient;
      if(!sockClient.Create())
      {
            pDlg->TransfersFailed();
           ::MessageBox((HWND)lparam, pDlg->GetError(GetLastError()), _T("错误"), MB_ICONHAND|MB_OK);
            return -1;
      }

      CString strIPAddress;
      UINT nPort;
      pDlg->m_psockClient->GetPeerName(strIPAddress, nPort);
```

```
	//连接到服务器端
	  if(!sockClient.Connect(strIPAddress, pDlg->m_wPort + PORT))
	{
		pDlg->TransfersFailed();
		::MessageBox((HWND)lparam, pDlg->GetError(GetLastError()),
_T("错误"), MB_ICONHAND|MB_OK);
		return -1;
	}

	  //调用主对话框类中的 SendFile 成员函数进行文件的发送
	pDlg->SendFile(sockClient);

	return 0;
}
```

（2）接收文件线程

```
UINT _ListenThread(LPVOID lparam)
{
	CFileTransfersDlg* pDlg = (CFileTransfersDlg*)lparam;

	//创建套接字
	CSocket sockSrvr;
	if(!sockSrvr.Create(pDlg->m_wPort + PORT))
	{
		pDlg->TransfersFailed();
		::MessageBox((HWND)lparam, pDlg->GetError(GetLastError()),
_T("错误"), MB_ICONHAND|MB_OK);
		return -1;
	}

	//开始监听
	if(!sockSrvr.Listen())
	{
		pDlg->TransfersFailed();
```

```
            ::MessageBox((HWND)lparam, pDlg->GetError(GetLastError()),
_T("错误"), MB_ICONHAND|MB_OK);
            return -1;
      }
        //向主对话框发送一个自定义消息 WM_ACCEPT_TRANSFERS
        //发送一个信息告诉发送方可以开始发送文件
      pDlg->SendMessage(WM_ACCEPT_TRANSFERS);
      //接受连接
      CSocket recSo;
      if(!sockSrvr.Accept(recSo))
      {
            ::MessageBox((HWND)lparam, pDlg->GetError(GetLastError()),
_T("错误"), MB_ICONHAND|MB_OK);
            return -1;
      }
      sockSrvr.Close();
      //调用主对话框类中的 ReceiveFile 成员函数进行文件的接受
        pDlg->ReceiveFile(recSo);

      return 0;
}
```

这两个函数只需要放在 FileTransfersDlg.cpp 文件中，在该文件中还需要加入两个 CWinThread*类型的全局变量 pThreadListen 和 pThreadSend 用于对线程进行管理。

（五）在主对话框类内组织程序

（1）添加成员变量

使用 ClassWizard 为控件添加成员变量，其对应关系如下所示。

控件 ID	变量类型	变量名
IDC_FILE_NAME	CString	m_strFileName
IDC_FILE_SIZE	CString	m_strFileSize
IDC_PORT	UINT	m_wPort
IDC_IPADDRESS	CProgressCtrl	m_ctrlProgress
IDC_RADIO_SERVER	int	m_nServerType

（2）添加按钮消息的处理函数

添加按钮消息的处理函数，如下所示。

资源 ID	消息类型	函数名称
IDC_RADIO_SERVER	BN_CLICKED	OnRadioServer
IDC_RADIO_CLIENT	BN_CLICKED	OnRadioClient
IDC_BEGIN	BN_CLICKED	OnBegin
IDC_DISCONNECT	BN_CLICKED	OnDisconnect
IDC_SELECT_FILE	BN_CLICKED	OnSelectFile
IDC_STOP_TRANSFERS	BN_CLICKED	OnStopTransfers
IDCANCEL	BN_CLICKED	OnCancel

（3）添加 WM_TIMER 消息控制函数

使用 ClassWizard 为 CFileTransfersDlg 类中加一个 WM_TIMER 消息控制函数。

（4）加入必要的宏

在定义文件 FileTransfersDlg.h 的首部加入如下的宏，以增强程序的可读性。

```
#define PORT 1024                          //文件传输套接字的端口号
#define BLOCKSIZE 1024                     //每次要发送或是接受的文家大小
#define SERVER 0                           //表示当前为服务器端
#define CLIENT 1                           //表示当前为客户端
#define CONNECT_BE_ACCEPT    0x00          //客户端的连接申请被接受
#define CONNECT_BE_REFUSE    0x01          //客户端的连接申请被拒绝
#define DISCONNECT           0x02          //连接被断开
#define REQUEST              0x03          //请求发送文件
#define ACCEPT               0x04          //同意发送文件
#define REFUSE               0x05          //拒绝发送文件
#define CANCEL               0x06          //取消文件的发送
```

（5）自定义消息

自定义一个消息 WM_ACCEPT_TRANSFERS，用于当文件接收方同意接收文件且文件接收线程_ListenThread 已经准备好接收文件是，发送一个信息给文件发送方，说可以开始发送文件。

第一步，我需要在主窗口类 CFileTransfersDlg 的实现文件 FileTransfersDlg.cpp 的首部加入一句：#define WM_ACCEPT_TRANSFERS WM_USER + 100，定义一个宏。

第二步，实现消息处理函数。该函数使用 WPRAM 和 LPARAM 参数并返回 LPESULT，其实现如下：

```
LRESULT CFileTransfersDlg::OnAcceptTransfers(WPARAM wParam, LPARAM lParam)
{
      //告诉对方文件请求被接受且准备好接收
        CMessage* pMsg = new CMessage(ACCEPT);
      m_psockClient->SendMsg(pMsg);
      //设置一个 ID 为 2 的超时几时器
        m_nTimer = SetTimer(2, 5000, NULL);
      return 0;
}
```

第三步，在 CFileTransfersDlg 类定义中的 AFX_MSG 块中说明消息处理函数，详见类的定义。

第四步，在 FileTransfersDlg.cpp 文件的在用户类的消息块中，使用 ON_MESSAGE 宏指令将消息映射到消息处理函数中，加入了如下一句：

```
BEGIN_MESSAGE_MAP(CFileTransfersDlg, CDialog)
      //{{AFX_MSG_MAP(CFileTransfersDlg)
      ..........
      ON_MESSAGE(WM_ACCEPT_TRANSFERS, OnAcceptTransfers)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

（6）添加成员

使用 ClassView 或是手工加入如下的函数及成员变量的定义：

```
public:
      CListenSocket* m_psockServer;
      CClientSocket* m_psockClient;
      CString m_strPath;
      DWORD m_dwFileSize;
      int m_nTimer;
      BOOL m_bIsClient;
```

```
BOOL m_bIsStop, m_bIsWait, m_bIsTransmitting;
void ProcessAccept();
void ProcessReceive(CClientSocket* pSocket);
void SendFile(CSocket &senSo);
void ReceiveFile(CSocket &recSo);
void TransfersFailed();
CString GetError(DWORD error);
int GetLocalHostInfo(CString &strHostName, CString
&strIPAddress);
```

其中 m_psockServer 和 m_psockClient 分别是用于监听连接和传输控制信息的套接字；m_strPath 为要发送的文件的路径；m_dwFileSize 为文件的大小；m_bIsClient 用于标识当前是文件发送方还是接收方；m_bIsStop，m_bIsWait 和 m_bIsTransmitting 分别表示是否要停止文件传输，是否处于等待状态和是否正在传输文件。在定义文件 FileTransfersDlg.h 的首部加入如下的头文件包含：

```
#include "ListenSocket.h"
#include "ClientSocket.h"
```

同时还得在主窗口类的定义上方加入下面的语句：

```
class CListenSocket;
class CClientSocket;
```

这样程序才能让程序正常运行。

（7）修改成员函数代码

首先在 CFileTransfersDlg 类的构造函数中添加如下的初始化代码：

```
m_strFileSize = _T("0 字节");
m_wPort = 9600;
m_nServerType = 0;
m_psockServer = NULL;
m_psockClient = NULL;
m_strPath = _T("");
m_dwFileSize = 0;
m_bIsStop = FALSE;
m_bIsWait = FALSE;
m_bIsTransmitting = FALSE;
```

在 OnInitDialog 函数中加入如下的控件初始化代码：

```
  CString strHostName,strIPAddress;
    if(GetLocalHostInfo(strHostName, strIPAddress))
        return FALSE;
    GetDlgItem(IDC_IPADDRESS)->SetWindowText(strIPAddress);
    GetDlgItem(IDC_IPADDRESS)->EnableWindow(FALSE);
```

在 OnInitDialog 函数调用了一个成员函数 GetLocalHostInfo，用于获取本地的 IP 地址，其代码如下所示：

```
int CFileTransfersDlg::GetLocalHostInfo(CString &strHostName, CString &strIPAddress)
{
    char szHostName[256];
    if(gethostname(szHostName, sizeof(szHostName)))
    {
        strHostName = _T("");
        MessageBox(GetError(GetLastError()), _T("错误"), MB_ICONHAND|MB_OK);
        return -1;
    }
    PHOSTENT hostinfo;
    if((hostinfo = gethostbyname(szHostName)) == NULL)
        return GetLastError();
    LPCSTR ip = inet_ntoa (*(struct in_addr *)*hostinfo->h_addr_list);
    strIPAddress = ip;
    strHostName = szHostName;
    return 0;
}
```

用于获取错误信息的的成员函数 GetError，其代码如下所示：

```
CString CFileTransfersDlg::GetError(DWORD error)
{
    CString strError;
    switch(error)
    {
```

```
        case WSANOTINITIALISED:
              strError="初始化错误";
              break;
        case WSAENOTCONN:
              strError="对方没有启动";
              break;
        case WSAEWOULDBLOCK :
              strError="对方已经关闭";
              break;
        case WSAECONNREFUSED:
              strError="连接的尝试被拒绝";
              break;
        case WSAENOTSOCK:
              strError="在一个非套接字上尝试了一个操作";
              break;
        case WSAEADDRINUSE:
              strError="特定的地址已在使用中";
              break;
        case WSAECONNRESET:
              strError="与主机的连接被关闭";
              break;
        default:
              strError="一般性错误";
        }
        return strError;
}
```

传输启动错误是的处理函数 TransfersFailed，其代码如下：

```
void CFileTransfersDlg::TransfersFailed()
{
        GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
        GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
        GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
```

```
}
```

当 CListenSocket 套接字类接收到连接请求时的函数 ProcessAccept，其代码如下：

```
void CFileTransfersDlg::ProcessAccept()
{
	CClientSocket* pSocket = new CClientSocket(this);
	//将请求接收下来，得到一个新的套接字 pSocket
	  if(m_psockServer->Accept(*pSocket))
	{
		//初始化套接字 pSocket
		    pSocket->Init();
		CMessage* pMsg;
		//如果 m_psockClient 套接字为空，则表示还没有和任何客户端建立连接
		    if(m_psockClient == NULL)
		{
			//向客户端发送一个消息，表示连接被接受
				pMsg = new CMessage(CONNECT_BE_ACCEPT);
			pSocket->SendMsg(pMsg);
			m_psockClient = pSocket;

			GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
		}
		else
		{
			//否则向客户端发一个信息，服务器已经存在连接
				pMsg = new CMessage(CONNECT_BE_REFUSE);
			pSocket->SendMsg(pMsg);
		}
	}
}
```

这一来就能保证服务器端一次只能同一个客户端存在连接。

CClientSocket 套接字类收到信息时的处理函数 ProcessReceive，其代码如下：

```
void CFileTransfersDlg::ProcessReceive(CClientSocket* pSocket)
{
	//获取信息
	  CMessage* pMsg = new CMessage();
	  pSocket->ReceiveMsg(pMsg);

	//当消息类型为连接被接受时执行该 if 语句里面的内容
	  if(pMsg->m_nType == CONNECT_BE_ACCEPT)
	{
		GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
		return;
	}
	  //当消息类型为连接被拒绝时执行该 if 语句里面的内容
	if(pMsg->m_nType == CONNECT_BE_REFUSE)
	{
		MessageBox(_T("服务器已经和另外的客户端建立连接，请等一下再连接。"), _T("错误"), MB_ICONHAND);
		delete m_psockClient;
		m_psockClient = NULL;
		GetDlgItem(IDC_RADIO_SERVER)->EnableWindow(TRUE);
		GetDlgItem(IDC_RADIO_CLIENT)->EnableWindow(TRUE);
		GetDlgItem(IDC_IPADDRESS)->EnableWindow(TRUE);
		GetDlgItem(IDC_PORT)->EnableWindow(TRUE);
		GetDlgItem(IDC_BEGIN)->EnableWindow(TRUE);
		GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
		return ;
	}
	  //当消息类型为连接被断开时执行该 if 语句里面的内容
	if(pMsg->m_nType == DISCONNECT)
	{
		MessageBox(_T("对方已经关闭"), _T("警告"), MB_ICONHAND);
```

```
            if(m_psockClient != NULL)
            {
                  delete m_psockClient;
                  m_psockClient = NULL;
            }
            if(m_nServerType == CLIENT)
            {
                  GetDlgItem(IDC_RADIO_SERVER)->EnableWindow(TRUE);
                  GetDlgItem(IDC_RADIO_CLIENT)->EnableWindow(TRUE);
                  GetDlgItem(IDC_PORT)->EnableWindow(TRUE);
                  GetDlgItem(IDC_BEGIN)->EnableWindow(TRUE);
                  GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
                  GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
                  GetDlgItem(IDC_IPADDRESS)->EnableWindow(TRUE);
            }
            else
            {
                  GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
            }
            return ;
      }
      //当收到传输文件请求时执行该 if 语句里面的内容
        if(pMsg->m_nType == REQUEST)
      {
            m_bIsWait = TRUE;
            m_strFileName = pMsg->m_strFileName;
            m_dwFileSize = pMsg->m_dwFileSize;
            CFileDialog dlg(FALSE, NULL, NULL,
OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT, "所有文件 (*.*)|*.*||", this);
            dlg.m_ofn.lpstrTitle = _T("另存为");
            strcpy(dlg.m_ofn.lpstrFile,
m_strFileName.GetBuffer(m_strFileName.GetLength()));
```

```
		if(dlg.DoModal() == IDOK)
		{
			if(m_bIsWait == FALSE)
			{
				MessageBox(_T("对方已经取消文件发送"), _T("警
告"), MB_ICONEXCLAMATION);
				return ;
			}
			m_bIsClient = FALSE;
			m_strPath = dlg.GetPathName();
			GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
			GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
			GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(TRUE);
			m_strFileSize.Format("%ld 字节", m_dwFileSize);
			GetDlgItem(IDC_FILE_NAME)-
>SetWindowText(dlg.GetFileName());
			GetDlgItem(IDC_FILE_SIZE)-
>SetWindowText(m_strFileSize);
			GetDlgItem(IDC_TRANSFERS_TIP)->SetWindowText(_T("已
收到:"));
			//启动接收文件的线程
				pThreadListen
= ::AfxBeginThread(_ListenThread, this);
			return ;
		}
		if(m_bIsWait == TRUE)
		{
			//告诉对方文件发送请求被拒绝
				CMessage* pMsg = new CMessage(REFUSE);
			m_psockClient->SendMsg(pMsg);
		}
		m_bIsWait = FALSE;
		return ;
	}
```

```
        //当对方同意且准备好接收文件时执行该 if 语句里面的内容
          if(pMsg->m_nType == ACCEPT)
        {
              KillTimer(1);
              m_bIsWait = FALSE;
                  //启动文件发送线程
              pThreadSend = ::AfxBeginThread(_SendThread, this);
              return ;
        }
          //当发送文件请求被拒绝时执行该 if 语句里面的内容
        if(pMsg->m_nType == REFUSE)
        {
              m_bIsWait = FALSE;
              MessageBox(_T("请求被拒绝"), _T("警告"),
MB_ICONEXCLAMATION);
              GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
              GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
              GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
              return ;
        }
        //当对方取消文件传输时执行该 if 语句里面的内容
          if(pMsg->m_nType == CANCEL)
        {
              m_bIsWait = FALSE;
              return ;
        }
        return ;
}
```

该函数主要用于处理文件传输的控制消息。

发送文件函数 SendFile，该函数在发送文件线程被调用，其代码如下：

```
void CFileTransfersDlg::SendFile(CSocket &senSo)
{
```

```
	m_bIsTransmitting = TRUE;
	//打开要发送的文件
	  CFile file;
	if(!file.Open(m_strPath, CFile::modeRead | CFile::typeBinary))
	{
		AfxMessageBox(_T("文件打开失败"));
		GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
		GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
		GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
		senSo.Close();
		return ;
	}
	m_ctrlProgress.SetRange32(0, m_dwFileSize);
	int nSize = 0, nLen = 0;
	DWORD dwCount = 0;
	char buf[BLOCKSIZE] = {0};
	file.Seek(0, CFile::begin);
	//开始传送文件
	  for(;;)
	{
		//每次读取 BLOCKSIZE 大小的文件内容
		    nLen = file.Read(buf, BLOCKSIZE);
		if(nLen == 0)
		      break;
		//发送文件内容
		    nSize = senSo.Send(buf, nLen);
		dwCount += nSize;
		m_ctrlProgress.SetPos(dwCount);
		CString strTransfersSize;
		strTransfersSize.Format("%ld 字节", dwCount);
		GetDlgItem(IDC_RECEIVE_SIZE)-
>SetWindowText(strTransfersSize);
```

```
            //用户是否要停止发送
                if(m_bIsStop)
            {
                  m_bIsStop = FALSE;
                  break;
            }

            if(nSize == SOCKET_ERROR)
                  break;
      }
        //关闭文件
      file.Close();
        //关闭套接字
      senSo.Close();

      if(m_dwFileSize == dwCount)
            AfxMessageBox(_T("文件发送成功"));
      else
            AfxMessageBox(_T("文件发送失败"));
      m_ctrlProgress.SetPos(0);
      GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
      GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
      GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
      m_bIsTransmitting = FALSE;
}
```

文件接收函数 ReceiveFile，该函数在文件接收线程中被调用，其代码如下：

```
void CFileTransfersDlg::ReceiveFile(CSocket &recSo)
{
      //停止等待超时计时器
        KillTimer(2);
```

```
	m_bIsWait = FALSE;
	m_bIsTransmitting = TRUE;
	m_ctrlProgress.SetRange32(0, m_dwFileSize);
	GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
	GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
	GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(TRUE);
	int nSize = 0;
	DWORD dwCount = 0;
	char buf[BLOCKSIZE] = {0};
	//创建一个文件
	  CFile file(m_strPath, CFile::modeCreate|CFile::modeWrite);
	//开始接收文件
	  for(;;)
	{
		//每次接收 BLOCKSIZE 大小的文件内容
		    nSize = recSo.Receive(buf, BLOCKSIZE);
		if(nSize == 0)
		      break;
		//将接收到的文件写到新建的文件中去
		    file.Write(buf, nSize);
		dwCount += nSize;
		m_ctrlProgress.SetPos(dwCount);
		CString strTransfersSize;
		strTransfersSize.Format("%ld 字节", dwCount);
		GetDlgItem(IDC_RECEIVE_SIZE)-
>SetWindowText(strTransfersSize);
		    //用户是否要停止接收
		if(m_bIsStop)
		{
		      m_bIsStop = FALSE;
		      break;
		}
```

```
        }
          //关闭文件
        file.Close();
          //关闭套接字
        recSo.Close();

        if(m_dwFileSize == dwCount)
              AfxMessageBox(_T("文件接收成功"));
        else
              AfxMessageBox(_T("文件接收失败"));
        m_ctrlProgress.SetPos(0);
        GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
        GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
        GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
        m_bIsTransmitting = FALSE;
}
```

修改点击“服务器(&S)”单选按钮的事件响应函数 OnRadioServer，可以使用 WizardBar 定位到函数的实现代码，修改如下：

```
void CFileTransfersDlg::OnRadioServer()
{
        CString strHostName,strIPAddress;
        if(GetLocalHostInfo(strHostName, strIPAddress))
              return ;
        GetDlgItem(IDC_IPADDRESS)->SetWindowText(strIPAddress);
        GetDlgItem(IDC_IPADDRESS)->EnableWindow(FALSE);
        GetDlgItem(IDC_BEGIN)->SetWindowText(_T("启  动(&B)"));
        GetDlgItem(IDC_DISCONNECT)->SetWindowText(_T("关  闭(&D)"));
        GetDlgItem(IDC_TRANSFERS_TIP)->SetWindowText(_T("已发送:"));
}
```

修改点击“客户端(&C)”单选按钮的事件响应函数 OnRadioClient，代码如下：

```
void CFileTransfersDlg::OnRadioClient()
{
```

```
	GetDlgItem(IDC_IPADDRESS)->SetWindowText(_T(""));
	GetDlgItem(IDC_IPADDRESS)->EnableWindow(TRUE);
	GetDlgItem(IDC_BEGIN)->SetWindowText(_T("连　接(&B)"));
	GetDlgItem(IDC_DISCONNECT)->SetWindowText(_T("断　开(&D)"));
}
```

修改点击“启　动(&B)”按钮的事件响应函数 OnBegin，代码如下：

```
void CFileTransfersDlg::OnBegin()
{
	UpdateData(TRUE);

	//当程序作为服务器
	  if(m_nServerType == SERVER)
	{
		//创建服务器套接字
		m_psockServer = new CListenSocket(this);
		if(!m_psockServer->Create(m_wPort))
		{
			delete m_psockServer;
			m_psockServer = NULL;
			MessageBox(GetError(GetLastError()), _T("错误"),
MB_ICONHAND);
			return ;
		}

		//监听客户端连接
		if(!m_psockServer->Listen())
		{
			delete m_psockServer;
			m_psockServer = NULL;
			MessageBox(GetError(GetLastError()), _T("错误"),
MB_ICONHAND);
			return ;
		}
```

```
	}
	else
	{
		//当程序作为为客户端
		if(((CIPAddressCtrl*)GetDlgItem(IDC_IPADDRESS))->IsBlank())
		{
			MessageBox("IP 地址不能为空", "错误", MB_ICONHAND);
			return ;
		}

		//创建客户端套接字
		m_psockClient = new CClientSocket(this);
		if(!m_psockClient->Create())
		{
			delete m_psockClient;
			m_psockClient = NULL;
			MessageBox(GetError(GetLastError()), _T("错误"), MB_ICONHAND);
			return ;
		}
		//与服务器建立连接
		CString strIPAddress;
		GetDlgItem(IDC_IPADDRESS)->GetWindowText(strIPAddress);
		if(!m_psockClient->Connect(strIPAddress, m_wPort))
		{
			delete m_psockClient;
			m_psockClient = NULL;
			MessageBox(GetError(GetLastError()), _T("错误"), MB_ICONHAND);
			return ;
		}
		//初始化套接字
```

```
            m_psockClient->Init();
            GetDlgItem(IDC_IPADDRESS)->EnableWindow(FALSE);
        }
        GetDlgItem(IDC_RADIO_SERVER)->EnableWindow(FALSE);
        GetDlgItem(IDC_RADIO_CLIENT)->EnableWindow(FALSE);
        GetDlgItem(IDC_PORT)->EnableWindow(FALSE);
        GetDlgItem(IDC_BEGIN)->EnableWindow(FALSE);
        GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
}
```

修改点击“关　闭(&D)”按钮的事件响应函数 OnDisconnect，代码如下：

```
void CFileTransfersDlg::OnDisconnect()
{
        UpdateData(TRUE);
        //关闭客户端套接字 m_psockClient
          if(m_psockClient != NULL)
        {
            //告诉对方连接被断开
                CMessage* pMsg = new CMessage(DISCONNECT);
            m_psockClient->SendMsg(pMsg);
            delete m_psockClient;
            m_psockClient = NULL;
        }
          //关闭服务器端套接字 m_psockServer
        if(m_psockServer != NULL)
        {
            delete m_psockServer;
            m_psockServer = NULL;
        }
        GetDlgItem(IDC_RADIO_SERVER)->EnableWindow(TRUE);
        GetDlgItem(IDC_RADIO_CLIENT)->EnableWindow(TRUE);
        GetDlgItem(IDC_PORT)->EnableWindow(TRUE);
        GetDlgItem(IDC_BEGIN)->EnableWindow(TRUE);
```

```
	GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
	GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
	if(m_nServerType == CLIENT)
		GetDlgItem(IDC_IPADDRESS)->EnableWindow(TRUE);
}
```

修改点击“选择文件(&F)”按钮的事件响应函数 OnSelectFile，代码如下：

```
void CFileTransfersDlg::OnSelectFile()
{
	CFileDialog dlg(TRUE, NULL, NULL,
OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT, "所有文件 (*.*)|*.*||", this);
	dlg.m_ofn.lpstrTitle = _T("打开");
	if(dlg.DoModal() == IDOK)
	{
		m_bIsWait = TRUE;
		m_bIsClient = TRUE;
		m_strPath = dlg.GetPathName();
		m_strFileName = dlg.GetFileName();

		//打开文件
		    CFile file(m_strPath, CFile::modeRead);
		//获取文件大小
		    m_dwFileSize = file.GetLength();
		m_strFileSize.Format("%ld 字节", m_dwFileSize);
		//关闭文件
		    file.Close();
		UpdateData(FALSE);
		//发出文件发送请求
		    CMessage* pMsg = new CMessage(REQUEST, m_strFileName,
m_dwFileSize);
		m_psockClient->SendMsg(pMsg);

		GetDlgItem(IDC_DISCONNECT)->EnableWindow(FALSE);
		GetDlgItem(IDC_SELECT_FILE)->EnableWindow(FALSE);
```

```
            GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(TRUE);
            GetDlgItem(IDC_TRANSFERS_TIP)->SetWindowText(_T("已发
送:"));
            //设置 ID 为 1 的等待超时定时器
            m_nTimer = SetTimer(1, 50000, NULL);
      }
}
```

修改点击“停止传输(&T)”按钮的事件响应函数 OnStopTransfers，代码如下：

```
void CFileTransfersDlg::OnStopTransfers()
{
      if(m_bIsWait)
      {
            if(MessageBox(_T("真的要停止等待吗？"), _T("警告"),
MB_ICONEXCLAMATION|MB_YESNO) == IDYES)
            {
                  m_bIsWait = FALSE;
                  if(!m_bIsClient)
                  {
                        //停止 ID 为 2 的计时器
                              if(KillTimer(2))
                        {
                              //结束监听
                                    CSocket sockClient;
                              sockClient.Create();
                              sockClient.Connect(_T("127.0.0.1"),
m_wPort + PORT);
                              sockClient.Close();
                        }
                  }
                  else
                  {
                        //停止 ID 为 1 的计时器
```

```
                        if(KillTimer(1))
                        {
                              //告诉对方发送等待被取消
                                    CMessage* pMsg = new
CMessage(CANCEL);
                              m_psockClient->SendMsg(pMsg);

                              GetDlgItem(IDC_DISCONNECT)-
>EnableWindow(TRUE);
                              GetDlgItem(IDC_SELECT_FILE)-
>EnableWindow(TRUE);
                              GetDlgItem(IDC_STOP_TRANSFERS)-
>EnableWindow(FALSE);
                        }
                  }
            }
            return ;
      }

      if(MessageBox(_T("真的要停止文件传输吗？"), _T("警告"),
MB_ICONEXCLAMATION|MB_YESNO) == IDYES)
      {
            m_bIsStop = TRUE;
            return ;
      }
}
```

修改点击“退出(&Q)”按钮的事件响应函数，代码如下：

```
void CFileTransfersDlg::OnCancel()
{
      if(m_bIsWait)
      {
           MessageBox(_T("等待中，请先停止传送后再退出"), _T("警告"),
MB_ICONEXCLAMATION);
            return ;
```

```
	}
	if(m_bIsTransmitting)
	{
		MessageBox(_T("文件传输中，请先停止传送后再退出"), _T("警告"), MB_ICONEXCLAMATION);
		return ;
	}
	OnDisconnect();
	CDialog::OnCancel();
}
```

修改 WM_TIMER 消息控制函数 OnTimer，代码如下：

```
void CFileTransfersDlg::OnTimer(UINT nIDEvent)
{
	switch(nIDEvent)
	{
	//ID 为 1 的计时器
	  case 1:
		{
			//结束 ID 为 1 的计时器
				KillTimer(1);
			m_bIsWait = FALSE;
				//告诉对方发送等待被取消
			CMessage* pMsg = new CMessage(CANCEL);
			m_psockClient->SendMsg(pMsg);

			MessageBox(_T("等待超时"), _T(" 警告"), MB_ICONEXCLAMATION);
			GetDlgItem(IDC_DISCONNECT)->EnableWindow(TRUE);
			GetDlgItem(IDC_SELECT_FILE)->EnableWindow(TRUE);
			GetDlgItem(IDC_STOP_TRANSFERS)->EnableWindow(FALSE);
			break;
		}
	//ID 为 2 的计时器
```

```
        case 2:
            {
                  //结束 ID 为 2 的计时器
                        KillTimer(2);
                  //结束监听
                        CSocket sockClient;
                  sockClient.Create();
                 sockClient.Connect(_T("127.0.0.1"), m_wPort + PORT);
                  sockClient.Close();
                  break;
            }
      }
      CDialog::OnTimer(nIDEvent);
}
```

至此，所有的代码编写完毕，最后的工作就是要编译连接完成。

5 软件测试

5.1 开发工具

软件采用面向对像的设计方法，考虑到对计算效率的要求，采用 C++编程语言，开发环境为 Windows XP，编程工具为 Microsoft Visual C++ 6.0，采用 C++标准库函数和 MFC 类库。

软件环境：使用计算机系统为 Windows 7。

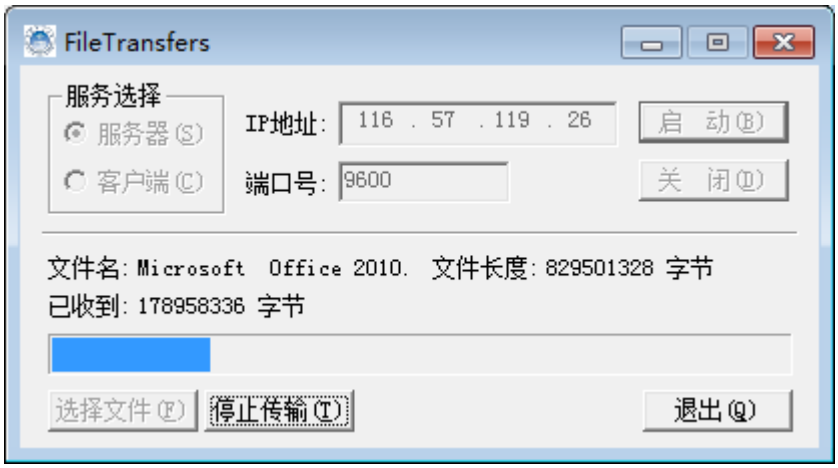
主要的特点是：利用多线程并发处理，模拟了异步传输模式，并通过消息达到了子线程与主线程的通信。

5.2 测试过程

（1）在编程工具 Microsoft Visual C++ 6.0 下完成程序的编译，调试无误后，生成可执行文件，对可执行文件进行测试。

（2）选择两台不同 IP 地址的主机进行测试，打开应用自动获取本机的 IP 地址，如图。

（3）通信双方建立连接，然后选择需要传输的文件，就能进行文件传输，如图。



（4）传输完毕可选择在此传输其他文件，或者断开连接，退出应用。

详细测试过程可参考 1.4 实现效果。

6 实验总结

6.1 实验体会

本次实验的内容是局域网内点对点文件传输的设计与实现，该设计实现局域网内的文件传输功能，包括服务器端程序和客户端程序两部分，用 C++作为编程语言，在 VC6.0 上进行调试、编译、测试等。测试时，在两台不同的主机上进行文件的传输，测试结果显示，文件传输的过程稳定、效果良好。在确定实验内容后，我首先是对对相关专业知识的学习与理解，学到了很多通信及编程的知识，首先是对 TCP 协议和 UDP 协议有了更进一步的认识；其次在编写过程中，通过翻阅书籍学习了 VC++编程和 MFC 的相关内容，拓展了自己的知识面，学到了很多在课堂上无法学到的东西。

6.2 实验展望

本次实验实现了局域网内点对点文件的传输，在局域网内对大文件的传输也体现出良好的性能，大大提高了工作效率，也节省了资源。此文件传输系统经过优化后可以作为聊天软件的一个附加功能，就是把原有的文件传输拓展为带有文件传输的聊天软件。这样可以使通信双方传递有用信息，而不需要借助其他通信工具，集多功能于一体的应用会更有前景。此外，利用 Winsock 进行数据文件的传输，其特点是可控性强，实现灵活方便， 可根据需要将此功能扩展到广域网中， 使此方法的数据传输技术更加完善实用。

参考资料

[1]《网络通信原理（第二版）》冯穗力著

[2]《计算机网络（第五版）》谢希仁著

[3]《Windows 网络编程技术》 北京:机械工业出版社

[4]《Visual C++ 6.0 编程实例精解》 北京希望电子出版社 兰芸编著

[5]《MFC 编程大全》 中国邮电出版社