实验四 TCP文件传输程序

# 4.1 实验目的

本实验应用 Socket 通信的 TCP 模式实现文件发送和接收，.NET 平台中 Networkstream 类提供数据在网络信道上的读写方法。支持 TCP 通信的 Windows 窗体程序不仅涉及到文件、线程、Socket、数据缓冲区和同步对象，它还包含回调函数和自定义消息处理，多个对象关系复杂。Socket 对象实现的 TCP 通信流程包含 Listen、Accept 及 Connect 主要操作，Windows 平台上的多线程与异步并发机制要与 Socket 对象的方法协调配合才能实现网络通信任务。本实验设计的 Windows 窗体程序应用 TCP 模型实现了单连接线程的文件传输的功能。

# 4.2 TCP 工作原理

应用于互联网中的 TCP/IP 是最重要的网络协议，TCP 是基于连接的网络通信协议实现设备之间可靠的数据传输。TCP 通信模式包含服务器和客户机两种角色，服务器先处于监听状态，多个客户端向服务器发连接请求，服务器针对每个连接请求创建一个套接字对象与客户端配对，配对的套接字执行实际数据的通信。服务端管理套接字对象可通过配套线程对象，它实现传输任务的异步执行，将客户端与服务端的线程对象结合 TCP 通信流程模式可由图4-1进行描绘，它展示了 TCP 通信时的工作原理。

网络通信程序具有跨语言、跨平台等特点，比如使用 C# 语言编写的 ftp 客户端程序运行在 Windows 平台上可以访问 Unix 平台上用 C++ 语言编写的 ftp 服务程序，这是因为通信设备中发送和接收的数据都基于字节序列。网络协议规定了字节在网络中的传输规范，是平台无关和语言无关的，因此计算机中任何数据类型，如图片、文字、视频等都可以字节序列在网络传送。

.NET 平台提供 FileStream 流类访问文件中的字节序列，NetworkStream 类访问网络中的字节序列，它们都从 Stream 流派生具有 Read 和 Write 方法。FileStream 流还支持 Seek 方法允许将读写位置移动到文件的任意位置，NetworkStream 流类提供在阻止模式下通过 Stream 套接字发送和接收数据的方法，它不支持对网络数据流的随机访问，在已连接成功的 Socket 对象上才可创建 NetworkStream 对象。

## 4.3 程序介绍

本实验采用 TCP 传输协议，以客户机/服务器网络通讯模式实现对任意类型文件发送和接收，任务包括下面几个功能：

1. 通信双方建立网络连接；

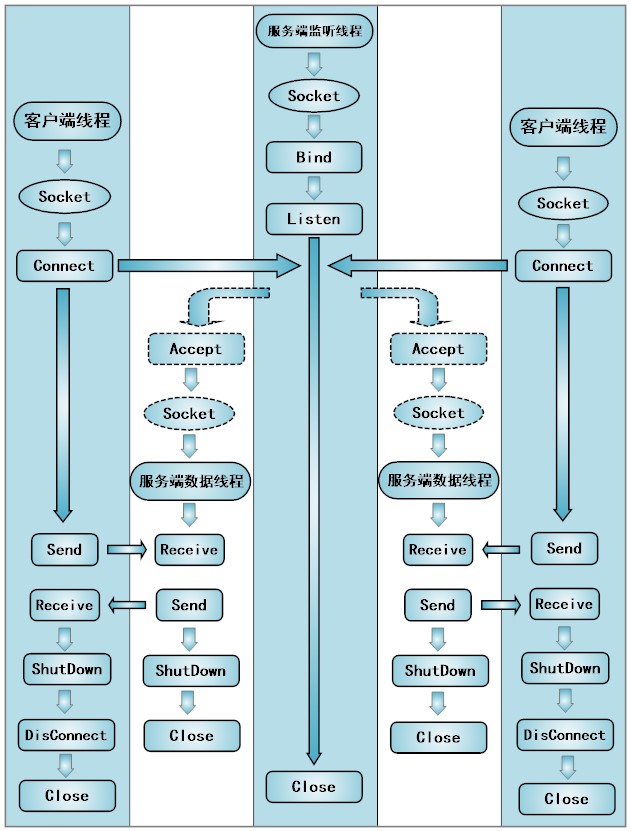


图 4-1 TCP 工作原理

1. 文件流与网络流的读写；
2. 数据传输过程控制与进度显示；

使用 TCP 的网络通信程序需要双方连接配对，服务端程序首先要先执行 Socket 类的 bind 方法完成端口绑定，绑定时要指定网络端口值，而客户端需要指明要连接的服务器的端口值。

接下来执行 Socket 类的 listen 方法使套接字对象处于监听状态，listen 方法中的参数值指明允许的最大连接数。客户端采用 Socket 类的 Connect 方法向服务器发起连接，服务端将发生 Accept 事件，可使用 Socket 的 BeginAccept 方法实现异步操作，该方法需要已定义的回调函数，这个回调函数响应连接请求后被调用，驱动程序创建新的 Socket 对象用于与客户端传输数据。

数据的传输是通过 Read 和 Write 方法实现的，网络硬件设备提供的只是数据的发送和接收服务，而网络设备随着网络吐吞表现会有变化，发送方的 Write 方法与读取方的 Read 方法很难存在严格的一一对应关系，比如发送方多次使用 Write 方法而接收方可能一次 Read 方法可接受完毕，程序逻辑要能够适应这种次数的不对应要求。常规方法是首先将文件大小信息发送给接收者，双方都通过文件长度值控制数据读写完全。文件的长度信息是用长整型数类型表示，在传送前要转化为字节序列，接收方则要对接收到的字节序列还原为长整型数。通过文件长度值控制发送与接收的循环，直到文件传送完成。

网络通信任务属耗时操作宜采用工作线程方式执行，工作线程向窗体发消息的方式实现把当前传输进度通知窗体线程。

## 4.4 实验内容

在本实验中只有一个客户端连接到服务端，客户端打开文件，读取文件内容到字节数组中，将字节数组内容写入到网络流。服务端用于接收数据的Socket对象通过流对象接收数据，并存入文件中。

### 4.4.1 客户端程序

创建窗体应用程序，添加一个文件选择对话框，一个进度条控件，首先添加变量声明部分：

//文件发送客户端，负责与服务器连接，传送文件

public static IntPtr main\_wnd\_handle;

public static IntPtr main\_label2\_handle;

public static String tran\_file\_name;

[DllImport("User32.dll", EntryPoint = "PostMessage")]

private static extern int PostMessage(

IntPtr hWnd, // handle to destination window int Msg, // message

int wParam, // first message parameter

int lParam // second message parameter

);

[DllImport("User32.dll", EntryPoint = "SendMessage")]

private static extern int SendMessage(

IntPtr hWnd, // handle to destination window

int Msg, // message

int wParam, // first message parameter

int lParam // second message parameter

);

//定义消息常数

public const int TRAN\_INFO = 0x500;

public const int TRAN\_SET\_PROGRESS = 0x501;

public const int TRAN\_UPDATE\_PROGRESS = 0x502;

public const int TRAN\_FINISHED = 0x503;

public static String Tr\_info;

工作线程负责将指定文件发送到服务器，它的主要流程是：

1. 设置进度条初值；
2. 连接远程服务器；
3. 发送文件名，文件大小基本信息给服务器；
4. 循环发送数据并实时更新进度值；
5. 设置发送完毕信息；

static void thread\_client\_trans()

{

//1. 设置进度条值

Tr\_info = "开始连接服务器";

PostMessage(main\_wnd\_handle, TRAN\_INFO, 100, 200);

FileInfo tr\_finf = new FileInfo(tran\_file\_name);

//PostMessage(main\_wnd\_handle, TRAN\_SET\_PROGRESS, 100, (int)tr\_finf.Length);

Tr\_info = " 正在传送文件";

SendMessage(main\_wnd\_handle, TRAN\_SET\_PROGRESS, 100, 5000);

//2. 连接远程服务器

//IPHostEntry ipHostInfo = Dns.Resolve("127.0.0.1");

//IPAddress ipAddress = ipHostInfo.AddressList[0];

IPEndPoint remoteEP = new IPEndPoint(IPAddress.Parse("127.0.0.1"),

Int32.Parse("8131")); // Create a TCP/IP socket.

Socket client\_sock = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

try

{ client\_sock.Blocking = true;

client\_sock.Connect(remoteEP);

try{

// Create the NetworkStream for communicating with the remote host.

NetworkStream client\_NetStream = new NetworkStream(client\_sock,

FileAccess.Write,true);

FileInfo tran\_file\_in = new FileInfo(tran\_file\_name);

//数据暂存缓存，开始数据打包

byte[] SendDataBuffer = new byte[1024];

//1 将数组值清空

Array.Clear(SendDataBuffer, 0, 1024);

//2long 整数返回字节长度为 8 的字节数组

byte[] b\_file\_len = BitConverter.GetBytes(tran\_file\_in.Length);

Array.Copy(b\_file\_len, 0, SendDataBuffer, 0, 8);

//3 文件字节数 —4 字节

byte[] b\_filename\_len = BitConverter.GetBytes(tran\_file\_in.Name.Length);

Array.Copy(b\_filename\_len, 0, SendDataBuffer, 8, 4);

//4 文件名长度 —8 字节

byte[] b\_file\_name = Encoding.UTF8.GetBytes(tran\_file\_in.Name);

//文件名字节数量因文件名会有所不同

Array.Copy(b\_file\_name, 0, SendDataBuffer, 8 + 4, b\_file\_name.Length);

//虽然客户端可以确定每次发送多少个字节，但接收端无法确定，因此约定先发送

1024 字节，有浪费

client\_NetStream.Write(SendDataBuffer, 0, 1024);

//使流发送出去

client\_NetStream.Flush();

//通知窗体发送文件的长度

SendMessage(main\_wnd\_handle, TRAN\_SET\_PROGRESS, 100, (int)tran\_file\_in.Length);

int tran\_count = 0;

int file\_read\_count = 0;

//FileStream 可以读取任意类型文件，StreamReader 只能读文本文件

FileStream fs\_file =tran\_file\_in.OpenRead();

do

{ file\_read\_count = fs\_file.Read(SendDataBuffer, 0, 1024);

client\_NetStream.Write(SendDataBuffer, 0, file\_read\_count);

client\_NetStream.Flush();

tran\_count += file\_read\_count;

SendMessage(main\_wnd\_handle, TRAN\_UPDATE\_PROGRESS, 100, tran\_count);

} while (client\_NetStream.CanWrite && fs\_file.Position<fs\_file.Length);

SendMessage(main\_wnd\_handle, TRAN\_FINISHED, 100, 100);

fs\_file.Close();

}catch(SocketException se3)

{

MessageBox.Show(" 客户端异常"+se3.Message);

}

}

catch (SocketException se1)

{MessageBox.Show("SocketException"+se1.Message);}

catch (Exception se2)

{MessageBox.Show(" 客户端异常" + se2.Message);}

Thread.Sleep(1000);

SendMessage(main\_wnd\_handle, TRAN\_FINISHED, 100, 200);

}

重载窗体消息处理函数处理自定义消息，更新窗体控件属性：

protected override void DefWndProc(ref System.Windows.Forms.Message m)

{ switch (m.Msg)

{

case TRAN\_INFO:

label2.Text = Tr\_info; break; case TRAN\_SET\_PROGRESS:

progressBar1.Maximum = (int)m.LParam;

progressBar1.Value = 0;

label2.Text = " 正在传送文件...";

break;

case TRAN\_UPDATE\_PROGRESS:

progressBar1.Value = (int)m.LParam ;

break;

case TRAN\_FINISHED:

label2.Text = " 文件已经传输完成";

progressBar1.Value = progressBar1.Maximum; break;

default:

base.DefWndProc(ref m); break;

}

}

文件发送按钮执行简单的检测：

private void button3\_Click(object sender, EventArgs e)

{//用户发出命令进行连接

//1. 检查具有文件信息，能否进行完整发送

//2. 如果没有文件信息，提示用户进行文件选择

//文件信息齐全则启动线程

//1. 检查具有文件信息，能否进行完整发送

if(!File.Exists(tran\_file\_name))

{

MessageBox.Show(" 你没有选择要传输的文件，不能传送");

return;

}

label2.Text = " 启动连接线程进行服务器连接......";

//2. 开始文件传输线程

ThreadStart workStart = new ThreadStart(thread\_client\_trans);

Thread workThread = new Thread(workStart);

workThread.IsBackground = true; workThread.Start();

}

选择文件对话框：

private void button1\_Click(object sender, EventArgs e)

{

//利用对话框选择要传输的文件

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{ tran\_file\_name = openFileDialog1.FileName;

FileInfo finf = new FileInfo(tran\_file\_name);

label1.Text = finf.Name;

}

}

### 4.4.2 服务端程序

服务端逻辑流程则要复杂些，它执行监听操作，采用套接字的异步方法 BeginAccept，响应连接请求后则启动工作线程接收客户端发来的文件数据，并对窗体控件进行更新。创建窗体程序在窗体上放置进度条和标签控件。下面是参考代码，首先是变量和常量定义：

[DllImport("User32.dll", EntryPoint = "SendMessage")]

private static extern int SendMessage(

IntPtr hWnd, // handle to destination window

int Msg, // message

int wParam, // first message parameter

int lParam // second message parameter 8

);

//定义消息常数

public const int BEGIN\_LISTEN = 0x500;

public const int END\_LISTEN = 0x501;

public const int TRAN\_CLIENT\_ACCEPT = 0x502;

public const int TRAN\_CLIENT\_TRAN = 0x503;

//用于设置传输进度

public const int TRAN\_FILE\_NAMES = 0x504;

public const int TRAN\_SET\_PROGRESS = 0x505;

public const int TRAN\_UPDATE\_PROGRESS = 0x506;

public const int TRAN\_FINISHED = 0x507;

public static ManualResetEvent User\_Terminate\_listen;

public static ArrayList socket\_list;

public static Socket S\_Listen\_sock;

public static Socket S\_client\_sock;

public static String tran\_file\_name;

public class Accep\_Object

{

}

在窗体的 Load 事件中对变量赋初始值：

private void Frm\_s\_Load(object sender, EventArgs e)

{ main\_wnd\_handle = this.Handle;

User\_Terminate\_listen = new ManualResetEvent(false);

socket\_list = new ArrayList();

}

服务器端最重要的部分是实现套接字监听，监听线程执行步骤如下面内容：

1. 获取主机信息；
2. 绑定指定端口；
3. 执行 listen 方法；
4. 使用 BeginAccept 开始异步接收；
5. 线程监听停止事件信号；
6. 检查所有已经连接的客户端，向每个客户端发送 close 命令；
7. 等待客户端关闭...，这部分涉及多个连接的套接字对象，程序实现复杂；
8. 如果所有连接客户端已经关闭，执行 listen 任务的 socket 执行 close 命令；下面是监听线程代码：

static void thread\_listen()

{

IPAddress[] host\_ip = Dns.GetHostAddresses(Dns.GetHostName());

S\_Listen\_sock = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

LingerOption \_lingerOption = new LingerOption(true, 3);

S\_Listen\_sock.SetSocketOption(SocketOptionLevel.Socket,

SocketOptionName.Linger, \_lingerOption);

S\_Listen\_sock.Blocking = false;//设定其为异步

//IPAddress.Parse("127.0.0.1");

//IPEndPoint host\_end = new IPEndPoint(host\_ip[0], 8128);

IPEndPoint host\_end = new IPEndPoint(IPAddress.Parse("127.0.0.1"),

Int32.Parse("8131"));

User\_Terminate\_listen.Reset();

S\_Listen\_sock.Bind(host\_end);//开始绑定

S\_Listen\_sock.Listen(3);//开始监听，listen 的参数用于指定最大连接数

Accep\_Object Ac\_state = new Accep\_Object();

S\_Listen\_sock.BeginAccept( new AsyncCallback(AcceptCallback),

Ac\_state);

SendMessage(main\_wnd\_handle, BEGIN\_LISTEN, 100, 200);

User\_Terminate\_listen.WaitOne();

//关闭所有的子 socket，结束监听

S\_Listen\_sock.Close();

SendMessage(main\_wnd\_handle, END\_LISTEN, 100, 200);

}

采用异步的 Accept 方法用到回调函数，这里是负责接收连接的回调函数代码：

public static void AcceptCallback(IAsyncResult ar)

{

//有新的客户端连接

// Get the socket that handles the client request.

if (S\_Listen\_sock == null)

{ //在此增加此语句有两个原因

//1.—Callback 是在消息队列里面循环调用的，当发生了主监听不存在的时候，

//callback 仍然会被调用。没有真正的客户到来，但是此函数被真实的调用了

S\_client\_sock = S\_Listen\_sock.EndAccept(ar);

MessageBox.Show(" 新客户已经开始连接服务器");

}else

{

//MessageBox.Show(" 新客户已经连接到服务器");

S\_client\_sock = S\_Listen\_sock.EndAccept(ar);

S\_client\_sock.Blocking = true;

//每次新的 Client 到来则启动一个新的线程，利用新的 Socket 与客户交互

ThreadStart clientWorkStart = new ThreadStart(thr\_client\_recv);

Thread clientThread = new Thread(clientWorkStart);

clientThread.IsBackground = true; clientThread.Start();

}

}

在.NET 平台中创建的资源由自动垃圾回收机制回收，网络通信中的 Socket 类的运行非常特殊，它实现 Berkeley 套接字接口，Socket 类的方法是对 WS2\_32.dll 文件的引用，当执行 Socket 类的 Close 方法时执行 WS2\_32.dll 文件中对应方法 closesocket 并对资源执行回收操作，Socket 类调用 Close 方法后其对象被设为空值 NULL，Callback 却实际上会被调用，因此发生ObjectDisposedException 异常，因此将执行下面代码：

MessageBox.Show("listen Socket is null，监听已经停止");

而函数 int closesocket(SOCKET ) 的动作是关闭指定的 socket，并且回收其所有的资源。函数 shutdown(SOCKET s, int how)要求在指定的 socket 上禁止进行由 how 参数代表的操作，但并不对资源进行回收，调用此函数往往表明发送者不再发送数据了，shutdown 之后还可获取socket对象的一些属性值，但不能用此sock对象建立新连接，必须先close，再socket创建新套接字建立新连接。socket.Close 方法调用后对应资源就不复存在,这时的socket对象将被设为null，引用其属性将会引发对象为空的异常。

成功执行 Accept 方法后，服务端新的 Socket 对象就可使用 Read 和 Write 方法进行实际的数据通信内容，包括下步骤：

1. 利用 Accept 方法创建的 client\_socket 创建 NetWorkStream 对象；
2. 接收文件信息并设置进度条；
3. 利用 Networkstream 对象接收所有文件内容，并更新进度条；
4. 设置发送完毕信息；下面是线程代码：

static void thr\_client\_recv()

{//单个接收线程入口

//线程流程

try

{

// Create the NetworkStream for communicating with the remote host.

NetworkStream client\_NetStream = new NetworkStream(S\_client\_sock,

FileAccess.Read, true);

byte[] ReceiveDataBuffer = new byte[1024];

//Array.Clear(ReceiveDataBuffer, 0, 1024);

//1. 获取文件头信息

client\_NetStream.Read(ReceiveDataBuffer, 0, 1024);

//解析基本文件信息

//2. 得到文件长度值

long file\_len = BitConverter.ToInt64(ReceiveDataBuffer, 0);

SendMessage(main\_wnd\_handle, TRAN\_SET\_PROGRESS, 100, (int)file\_len);

//3. 得到文件名长度值

int file\_name\_len = BitConverter.ToInt32(ReceiveDataBuffer, 8);

tran\_file\_name = Encoding.ASCII.GetString(ReceiveDataBuffer,

8 + 4, file\_name\_len);

SendMessage(main\_wnd\_handle, TRAN\_FILE\_NAMES, 100, 200);

string new\_file\_name = Environment.GetFolderPath(

Environment.SpecialFolder.Desktop) + "\\" + tran\_file\_name;

//重复传相同名称文件时，删掉原来接收到的文件

if (File.Exists(new\_file\_name)) File.Delete(new\_file\_name);

FileInfo fi\_file = new FileInfo(new\_file\_name);

FileStream fs\_newfile = fi\_file.OpenWrite();

int tran\_count = 0;

int numberOfBytesRead = 0;

do

{ numberOfBytesRead = client\_NetStream.Read(ReceiveDataBuffer, 0, 1024);

fs\_newfile.Write(ReceiveDataBuffer, 0, numberOfBytesRead);

fs\_newfile.Flush();

tran\_count += numberOfBytesRead;

SendMessage(main\_wnd\_handle, TRAN\_UPDATE\_PROGRESS, 100, tran\_count);

}while(client\_NetStream.DataAvailable && tran\_count < file\_len); SendMessage(main\_wnd\_handle, TRAN\_FINISHED, 100, 100);

fs\_newfile.Close();

}

catch (SocketException Se1)

{MessageBox.Show("SocketException:"+Se1.Message);}

catch (Exception Se2)

{MessageBox.Show(" 服务器端"+Se2.Message);}

}

启动监听线程的参考代码：

private void button1\_Click(object sender, EventArgs e)

{

//启动监听线程

ThreadStart workStart = new ThreadStart(thread\_listen);

Thread workThread = new Thread(workStart);

workThread.IsBackground = true; workThread.Start();

}

对窗体消息处理函数的重载处理自定义消息，用于更新窗体控件：

protected override void DefWndProc(ref System.Windows.Forms.Message m)

{

switch (m.Msg)

{

//接收自定义消息，并显示其参数

case BEGIN\_LISTEN: //m.WParam, m.LParam;

label4.Text = " 正在监听";

break;

case END\_LISTEN: //m.WParam, m.LParam;

label4.Text = " 结束监听";

break;

case TRAN\_CLIENT\_ACCEPT: //m.WParam, m.LParam;

label4.Text = " 新客户到达";

break;

case TRAN\_CLIENT\_TRAN: //m.WParam, m.LParam;

label4.Text = " 正在传输中";

break;

case TRAN\_FILE\_NAMES:

//设置文件名

//m.WParam, m.LParam;

label1.Text = tran\_file\_name;

break;

case TRAN\_SET\_PROGRESS:

//客端传来文件大小信息，开始传输

progressBar1.Maximum = (int)m.LParam; progressBar1.Value = 0;

break;

case TRAN\_UPDATE\_PROGRESS:

//更新文件传输状态

progressBar1.Value = (int)m.LParam;

break;

case TRAN\_FINISHED:

//文件传输完成

progressBar1.Value = progressBar1.Maximum; label4.Text = " 文件传输完成";

break;

default: 13

base.DefWndProc(ref m); break;

}

}

监听线程的结束是通过基本事件对象进行控制的，设置监听停止的代码：

private void button2\_Click(object sender, EventArgs e)

{ //用户按下按钮停止监听

User\_Terminate\_listen.Set();

}

## 4.5 程序说明

使用异步方式的 Accept 函数 BeginAccept，除了回调函数名参数，还有个 Object 类型的参数，它包含请求的状态信息，在程序中没有使用到，但有占位的语法要求，Accep\_Object 类用于定义这样的变量。

## 4.6 网络通讯异常与排错

网络程序会因为网络原因发生异常，例如服务端没有监听客户端进行连接等，.NET 平台的 SocketException 类提供获取错误代码，由于.NET 平台的 Socket 相关类引用 Ws2\_32.dll 文件中的函数，这个文件是 C++ 语言生成的，因此 SocketException 类获取的错误代码来自Winsock2.h 文件中的定义。获取网络通信的异常代码可参考下面的代码片断：

catch (SocketException se1)

{

MessageBox.Show(se1.ErrorCode.ToString());

//如果服务端没有进行监听，客户端会得到值为 10061 的错误码

//在 MSDN 中查找 Winsock Reference 主题能够查找对于错误的定义

//WSAECONNREFUSED 10061

//也可在 Winsock2.h 文件中查找这个错误代码

}

网络的异常情况非常多，异常码能够帮助程序定位异常的发生原因，反馈网络状况，提高程序的适用性。

## 4.7 实验作业

1. 调试并完善程序代码，完成本实验中的程序项目。
2. 本实验中文件名在全英文情况下可运行，请修正使其支持中文文件名。
3. 在服务端不启动情况下尝试进行连接，获取错误代码，并查找错误码代表的意义。
4. 思考如何进行多客户端连接，并终止多个客户连接。