计算机远程控制技术及其实现

李 彦，卢 虎
(1. 空军工程大学 电讯工程学院，陕西 西安 710077;2. 西北工业大学，陕西 西安 710077)

摘 要:介绍了利用 Microsoft 基本类库(MFC)实现计算机远程控制的核心技术及其主要功能的实现方法,给出了 Visual C ++ 6.0 实现的源代码并在 Windows 9x 上调试通过。

关键词:MFC;消息;插口
中图分类号:TN923　　**文献标识码**:A　　**文章编号**:1009 - 3516(2002)03 - 0087 - 04

随着计算机网络的迅速普及,远程控制做为计算机网络的一项关键技术,也越来越受到重视。不仅大量的 Hacker 软件采用此技术,而且越来越多的网络、工控、管理等软件也都强调此项功能。实现远程控制的方法多种多样,本文着重讨论了用 MFC Ssocket 类实现远程控制通信等问题。

1 MFC CSocket 类

远程控制软件属于网络编程的范畴,要进行网络编程,首先需要了解插口(Socket)的有关概念。插口是对通信端点进行抽象并提供发送和接收数据机制的一种网络编程接口,包括数据报插口和流式插口。为了简化基于 Windows 的 Socket(WinSock)网络编程,使用户更专注于应用程序算法的设计,我们使用 Microsoft 基本类库(MFC)提供的 Csocket 类。

CSocket 类为 WinSock API 提供了高级编程接口。CSocket 类可以和 CSocketFile 类、CArchive 类一起工作来处理数据的发送和接收,从而使用户避免手工处理字节顺序、字符串顺序等问题,而且 CSocket 类提供的阻塞调用功能恰好是使用 CArchive 类进行同步数据传输的最基本要求。

2 用 MFC Csocket 类实现远程控制通信

远程控制一般是一个控制端对多个被控端的控制。在通信机制上,属于服务器一客户机工作模式。控制端是客户机,被控端是服务器。程序工作的方式是:控制端向被控端发出连接请求,被控端进行应答,然后双方开始工作。

2.1 控制端通信层的编制

远程控制用三个类实现控制端通信层;分别是:CMsg、CClientSocket 和 CMyclass。下面,结合具体代码来说明这三个类的作用。

1)建立连接。在编写具体程序代码时,设置了两个全局变量。一个是 CString 类型的数组 m_Handle[2],其中数组成员个数代表被控端个数。这里为了简单只设置了两个元素,每个元素代表一个被控端的标识;另一个是指向 CMyClass 类的指针数组 m_Connect[2],每个元素指向一个 CMyClass 对象,代表一个被控端。

在主程序里,当用户发出一个连接被控端的请求后,通过一个有模式对话框,获取被控端的计算机标识,通常是一个 IP 地址或计算机名。然后,主程序把这个标识在 m_Handle 数组中保存起来。接着,主程序为 m_Connect 数组中的一个元素分配资源并将程序的框架窗口的指针传递给 CMyClass 类。

例如,连接第一个被控端,则返回计算机的标识被存储在 m_Handle[0]中,连接代码如下:

收稿日期:2001 - 09 - 10
作者简介:李 彦(1963 -),男,北京市人,副教授,主要从事信息处理研究.

m_Connect[0] = new CMyClass(this);其中,this 指针代表着框架窗体的对象指针。

接着调用 CMyClass 类的 ConnectSocket 成员函数:

m_Connect[0] - > ConnectSocket(m_Handle[0]);其中,m_Handle[0]是被控端的 IP 地址或计算机名。

CMyClass 类的成员函数 ConnectSocket 完成 CClientSocket 对象的创建、连接等一系列工作。代码如下:

```
//建立连接
BOOL CMyClass::ConnectSocket(Cstring strIp)
{ //创建客户插口
    m_pClientSocket = new CClientSocket(this);
    if (! m_pClientSocket - > Create())
    { //错误处理
      delete m_pClientSocket;
      m_pClientSocket = NULL;
      AfxMessageBox("插口创建失败");
      return FALSE;
    }
    //发送连接请求
    while (! m_pClientSocket - > Connect(strIp,8000))
    { //无法连接被控端
      if (AfxMessageBox("无法连接服务器\n 重试?",MB_YESNO) = = IDNO)
      { //错误处理
        delete m_pClientSocket;
        m_pClientSocket = NULL;
        return FALSE;
        }
      }
      //创建 CSocketFile 对象
      m_pFile = new CSocketFile(m_pClientSocket);
      //创建 CArchive 对象
      m_pArchiveIn = new CArchive(m_pFile,CArchive::load);
      m_pArchiveOut = new CArchive(m_pFile,CArchive::store);
      return TRUE;
}
```

可以看到,程序首先为客户插口指针 m_pClientSocket 分配资源,同时把 CMyClass 类的指针传递给 CClientSocket 对象;接着,调用 CClientSocket 类的成员函数 Create()才真正创建插口对象。Create()函数是 CClientSocket 类的基类成员函数,没有为该函数传递参数,就意味着插口的端口号是任意分配的,可以是系统的可用端口号中的任意一个;随后,使用 CClientSocket 类的成员函数 Connect 向被控端发出连接请求,并传递两个参数:被控端 IP 地址 strIP 和被控端连接端口号 8000。

如果被控端连接成功,则开始创建 CSocketFile 对象及输入、输出 CArchive 对象。并将它们与 CClientSocket 对象相关联。到此为止,我们已经和被控端建立起连接,之后控制端就可以利用这条建立起来的连接对被控端进行控制了。

2)数据收发过程。数据的接收是由插口来自动完成的。当数据到达控制端的缓冲区后,系统将自动调用 CClientSocket 类的 OnReceive()回调函数。此 OnReceive()函数并不进行数据的接收工作,而是通过调用 CMyClass 类的成员函数 CMyClass::OnReceive()来完成数据接受。

以上是数据接收的过程,数据发送比较简单,只需在主程序里直接调用 CMyClass 对象的 SendMsg 函数即可。

2.2 被控端通信层的编制

被控端是控制通信层的另一端,所以被控端的通信层大致与控制端相同。除了 CMsg 类、CClientSocket

类和 CMyClass 类以外,被控端增加了一个 CListenSocket 的类,其基类也是 CSocket。CListenSocket 只有一个成员变量,是指向框架窗口的指针 m_pDlg,成员函数主要是 OnAccept 回调函数。因为被控端相当于是通信中的服务器,所以在被控端程序的初始化阶段,需要创建侦听插口。其代码如下:

```
//创建侦听插口对象
    m_pListenSocket = new ClistenSocket(this);
    //开始侦听
    if (m_pListenSocket -> Create(8000))
    { //判断
        if (! m_pListenSocket -> Listen())
        {
            AfxMessageBox("侦听失败");
        }
    }
```

代码首先为侦听插口类分配资源,接着创建一个位于 8000 端口上的插口,然后调用 Listen 成员函数在该端口上进行侦听。这个端口号 8000,也就是控制端进行连接时所用的端口号。

完成以上步骤之后,被控端已经处于侦听状态。这个时候,如果控制端向被控端上的 8000 端口发出连接请求,则 ClistenSocket 类的回调函数 OnAccept()将被系统自动调用,但实际是调用主程序中的同名函数来分别为 CMyClass 类和 CClientSocket 类指针分配资源,而后调用 ClistenSocket 类的 Accept 成员函数来关联 CClientSocket 对象与请求连接的插口,并初始化该对象。至此已成功建立控制端和被控端间的通信,而被控端上数据的接收和发送同控制端上的大致相同,就不再赘述。

3　控制功能模块的实现

建立起控制端和被控端的通信层后,就可以进行远程控制具体功能的实现了。

3.1　控制端向被控端发送消息

控制端主程序中的发送消息的代码如下:

```
//发送文本消息
CTextDlg dlg;
If (dlg. DoModal() = =IDOK)
{ //标示
    dlg. m_Text = _T("1") + dlg. m_Text;
    //发送
    m_Connect[0] -> SendMsg(dlg. m_Text,0);
}
```

在主程序中弹出一个模式对话框,获取用户的文本输入,之后在该文本字符串前加上一个标记"1"。表示此后的数据是一条文本消息。接着,调用 CMyClass 类的成员函数 SendMsg 将数据发送出去。

当被控端收到这个数据后,首先进行解析,当读到第一个字符是"1",便将其后的数据作为一条文本消息显示出来。

3.2　控制端向被控端发送命令

和发送文本消息类似,在主程序弹出一个有模式对话框,用来获取用户的命令行字符串。然后,在命令行字符串前加上一个标志"2",用来表明标志后面的数据是一个命令行字符串。接着把它发送出去。

同接收文本消息类似,被控端收到数据进行解析,当读到标志"2",便把后面的数据作为一条命令行字符串来执行。执行命令行字符串的代码如下:

```
::ShellExecute(NULL,NULL,strTemp,NULL,NULL,SW_SHOWNORMAL);
```

被控端为了编程方便,直接调用了 Win32 API 函数 ShellExecute,前面的两个冒号表示调用 API 函数。ShellExcute 将把 strTemp 中的字符串作为一个命令加以执行。例如控制端发送"calc"字符串给被控端,被控端计算机将运行 Windows 自带的计算器应用程序 calc. exe。

3.3 控制端获取被控端系统信息

控制端发送的代码如下:

```
//标示
CString strTemp = _T("3");
//发送
m_Connect[0] -> SendMsg(strTemp,0);
```

被控端收到数据后进行解析,读到标志"3"后,知道控制端需要知道被控端的系统信息。可以获取的系统信息很多,为了简单可以直接调用 API 函数来获取各种系统信息,然后发送给控制端。需要注意的是被控端调用的各种获取系统信息的 Win32 API 函数,GetComputer Neme 只是其中之一,除此之外,还有如:Get - VersionEx,用来获取操作系统的版本号,CetSystemInfo 用来获取系统的各类信息等。

4 结束语

远程控制实际上就是一些计算机间传输数据的过程,重要的是对两边的程序处理,其功能远不止本文介绍的这些,解决方案也多种多样,比如我们想要隐藏客户端程序就可以通过注册一个服务进程来实现;而我们若要禁止用户的热启动,则可以利用相应的函数屏蔽该功能,然后重新编制一个关闭对话框。总之,完全可以根据实际需要来开发属于自己的远程控制程序。

参考文献:

[1] Jeffery Richter. Windows®高级编程指南[M]. 北京:清华大学出版社,1999.
[2] 候俊杰. 深入浅出 MFC[M]. 武汉:华中理工大学出版社,2001.
[3] 周天明. TCP/IP 协议[M]. 北京:清华大学出版社,1997.

(编辑:门向生)

The Computer Remote Control Technology and Realization

LI Yan, LU Hu
(The Telecommunication Engineering Institute, Air Force Engineering University, Xi'an 710077, China)

Abstract: This paper introduces the core technology about computer remote control and realization of its main functionality by Microsoft® MFC, and also presents the Visual C ++ 6.0 resource codes that have been debugged and passed in the Windows 9x.

Key Words: MFC; message; socket