基于 MFC CSocket 类的通用网络通信 DLL 设计①

李晓京, 文治洪, 胡文东, 张利利, 顼红雨, 马 进

(第四军医大学 航空航天医学教育部重点实验室, 西安 710032)

摘 要: 介绍一种在 VC、Delphi 等多种应用程序开发环境中通用的网络通信动态链接库(DLL)模块的设计实现. 在 visual studio 2010 开发环境下, 采用 c++ MFC 类库的 CSocket 类设计支持 TCP 协议和 UDP 协议的 DLL. 该 DLL 可被多种开发语言平台调用, 能以客户/服务端(C/S)、点对点、广播等多种模式建立局域网内通信机制, 为不同开发环境下设计模块化数据通信应用提供了便捷途径, 可大大提高跨平台网络通信应用程序开发的效率.

关键词: MFC; CSocket; 套接字; DLL; UDP 协议; TCP 协议; 局域网广播

Development of Universal Network Communication DLL Using MFC CSocket Class

LI Xiao-Jing, WEN Zhi-Hong, HU Wen-Dong, ZHANG Li-Li, XU Hong-Yu, MA Jin

(Key Laboratory of Aerospace Medicine of Ministry of Education, Fourth Military Medical University, Xi'an 710032, China)

Abstract: This paper develops a Dynamic Link Library (DLL) module that could be generally utilized by multiple programming environments for the construction of network communication. CSocket class, which is included in Microsoft Foundation Class (MFC), was adopted to develop the DLL that supports TCP and UDP protocol in visual studio 2010. The DLL was developed to construct network communication mechanism in the client/server, peer-to-peer and broadcast modes in a LAN. The realization of universal network communication DLL provides a good way to design the modularized data transfer applications in different programming environments. It can greatly raise the developing efficiency of such application projects.

Key words: MFC;CSocket; socket; DLL; UDP; TCP; LAN broadcast

局域网络通信机制在应用程序间的数据交互方面具等传输速度快、响应时间短、硬件成本低、可靠性高等优势. 尤其在目前 PC 主板普遍集成 100M 网卡的情况下, 应用项目的若干功能模块可方便的在局域网内建立相互间的数据交互渠道, 通过本地端口甚至不用添加任何硬件设备即可实现本地程序间(同一主机不同应用)的数据传输.

一般而言, 各种应用程序开发环境都提供了网络通信应用开发的便捷方式, 使得开发人员可以绕过繁琐的 Winsock API 函数实现网络通信, 如 VC MFC 类库中的 CSocket、CAsyncSocket 类, C++ Builder、Delphi 中的 TCPClient、TCPServer、UDPSocket 等控件 [1-3](为方便起见, 本文中统称组件). 尽管这些组件均建立在底层 Winsock API 的基础上实现, 但是由于不同开发工具底层设计的差异, 只有相同开发环境下使用配对组件创建的程序之间才能建立正常的网络连接, 不同开发环境下生成的应用间却无法正常通信. 如在 VC 环境下利用 CSocket 类开发的客户端应用就无法与 C++ Builder 环境下利用 TCPServer 控件开发的服务端应用建立连接. 如果在一个产品项目中, 需要由不同开发环境偏好的程序员分别完成的不同功能模块程序之间需要实现网络数据交互, 往往就需要采用相同的设计原则, 利用 Winsock API 针对各种开发环境设计出多个适用于不同开发环境的网络通信单元. 这无疑给项目开发成员增添了很大的工作量和复杂度.

为此, 本文以 VC MFC CSocket 类为基础, 采用 DLL 技术[4], 开发出了可在多种开发环境中调用的网络通信 DLL 组件, 解决了上述问题.

① 收稿时间:2012-10-16;收到修改稿时间:2012-11-24

1　CSocket类与TCP、UDP协议

CSocket 类由 MFC 类库的 CAsyncSocket 类派生而来, 继承并进一步实现了其对 WinSock API 的抽象封装, 使开发者不必过多考虑底层细节, 就可以快速建立网络通信框架.

CSocket 类支持流 (SOCK_STREAM) 和报文 (SOCK_DGRAM)两种通信模式[5], 其中流模式基于有连接的 TCP 协议进行数据传输, 报文模式则基于无连接的 UDP 协议实现数据传输. 流模式数据传输稳定可靠, 但速度稍慢; 报文数据则为非可靠传输, 但具有更优异的传输速度(所谓非可靠, 是指数据包的传输路由、能否到达、到达顺序都是不确定的, 但在小规模局域网中, 由于数据传输路由简单, 可靠性还是可以得到保证的)[6,7], 二者各有优劣. 但由于报文模式可以对同一网段内以广播方式实现公用数据的一对多一次性发送, 在局域网络应用中更具优势.

2　网络通信通用DLL模块需求分析

针对通用网络通信应用目标, 该模块需求如下:

1) 支持 TCP、UDP 协议;

2) 可以作为服务端、客户端运行, 可以点对点、广播方式发送接收数据;

3) 提供用于申请/注销套接字、绑定端口等网络连接创建与销毁操作、查询指定套接字工作模式、工作状态、本地\远程连接地址与端口等等必要接口函数, 提供用于连接/中断请求、数据收发的消息响应机制与相应函数, 底层细节封装良好;

4) 可同时被多种开发平台生成的应用程序共享调用.

由于 DLL 具有代码共享、独立更新、强保密性等优点[8], 采用该技术可以很好的满足上述需求.

3　网络通信通用DLL的实现

3.1 VC 项目配置需注意的问题

在 VS 2010 IDE 中选择 MFC DLL 项目并进入配置向导时, 根据目标需求, 在“应用程序设置”中, 选择“带静态链接 MFC 的规则 DLL”项, 这样可以使得 DLL 脱离 MFC 库独立发布, 且能够被所有 Win32 应用调用. 同时“附加功能选项”需选中“Windows 套接字”项, 让 IDE 自动加入 Windows 套接字的初始化代码.

3.2 数据结构与通信类设计

基于前述设计方案, 首先定义一个全局数据结构:

```
struct st_SRH
{
  HWND hwnd;
  CClientSocket *pClientSock;
  CListeningSocket *pListenSock;
  CUDPSocket *pUDPSock[2];
};
```

该结构实例指针将作为 SR 工具箱句柄提供给申请源应用, 其中保存了申请源应用窗体句柄, 并包含了一对 TCP 协议客户端/服务端套接字对象和两个 UDP 协议对象指针.

其次, 基于 MFC 的 CSocket 类派生分别用于 TCP 协议和 UDP 协议的三个套接字类, 其继承关系其继承关系如图 1.

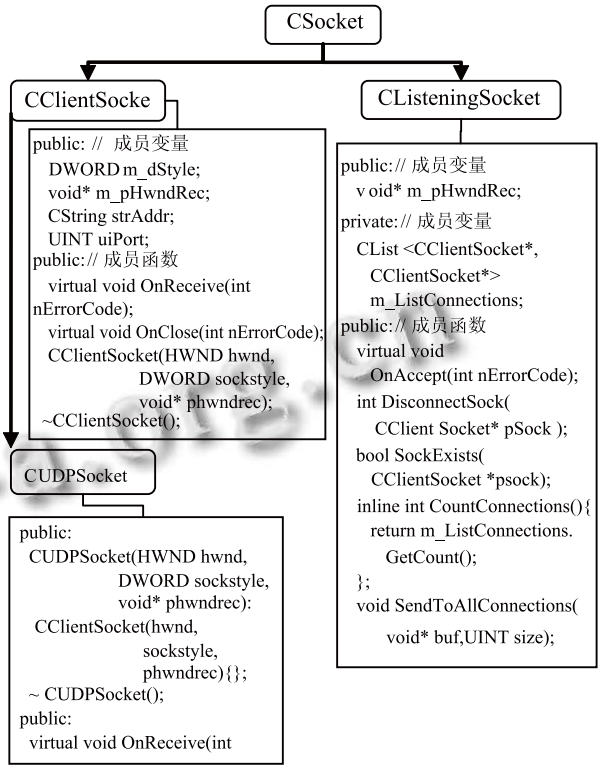

图 1　派生套接字类继承关系

其中, CClientSocket 类定义了五个成员变量, 用于保存套接字类型(分为客户端、服务端子客户端、UDP 点对点、UDP 广播四种)、所属 SR 实例句柄、远程连接地址及端口等信息, 重载数据接收、连接断开响应函数, 在响应函数中向所属应用窗体发送消息提

请接收数据或注销套接字, 并在构造函数中对成员变量完成初始化; CUDPSocket 类仅为分类方便直接由 CClientSocket 原封继承而来; CListeningSocket 定义相对复杂一些, 成员变量除用于保存所属 SR 实例句柄指针外, 还包括对可能的一对多连接进行管理的私有 CList 型子连接列表变量 m_ListConnections; 重载了远程连接请求响应函数, 该函数接受远程连接请求并创建子客户端套接字, 且将其纳入中子连接列表中以便管理, 此外还定义了断开子连接、查询子连接是否存在、子连接个数、通过所有子连接发送数据等成员函数. 相应地, 其析构函数需要完成所有子联接的清理任务, 这里不作细述.

一个或多个客户应用可能向 DLL 申请多个 SR, DLL 内部需要建立 SR 管理机制. 因此, 为 IDE 自动生成的 DLL 类(本文源程序中名为 CSockDllApp, 派生自 CWinApp 类)中添加成员变量:

```
CList<st_SRH*,st_SRH*> m_lstSockRec;
```

该列表变量用于存储、管理创建的 SR 句柄.

为了防止潜在的内存泄漏隐患, 为 DLL 类重载 CWinApp::ExitInstance 虚函数:

```
int CSockDllApp::ExitInstance()
{
    st_SRH *p;
    POSITION pos=m_lstSockRec.GetHeadPosition();
    for (int i=0;i<m_lstSockRec.GetCount();i++){
        p=m_lstSockRec.GetNext(pos);
        SAFEDEL(p->pClientSock);
        SAFEDEL(p->pListenSock);
        SAFEDEL(p->pUDPSock[0]);
        SAFEDEL(p->pUDPSock[1]);
        SAFEDEL(p);
    }
    return CWinApp::ExitInstance();
}
```

在 DLL 的 DLL_PROCESS_DETACH 机制发生作用时[9], 该函数将被调用, 逐一清理客户应用申请但未注销的 SR.

上述数据结构及套接字类被封装 DLL 中, 对客户应用来说处于黑箱之中, 用户将通过 DLL 公开的导出函数、响应预定义消息, 实现 SR 句柄获取、套接字管理并实现网络通信.

客户应用管理套接字资源、接收数据等操作, 建立在响应来自 DLL 中的各种消息并从消息参数中获得必要信息的基础上. 消息定义说明如下:

```
//该消息由 CClientSocket::OnReceive 重载函数向客户窗体
//发出, LPARAM 传递待接收数据套接字指针,
//WPARAM 参数参数保存套接字所属 SR 句柄
#define MSG_ RECIVE WM_USER+1500
//该消息由 CClientSocket::OnClose 重载函数向客户窗体发出,
//参数同上
#define MSG_ CLOSE WM_USER+1501
//该消息由 CListeningSocket::OnAccept 重载函数向客户窗体
//发出, 参数同上
#define MSG_ ACCEPT WM_USER+1502
```

为了便于使用, DLL 中除了定义资源句柄申请、注销等都套接字应用必须用到的常规导出函数, 还为各种套接字分类定义专用导出函数. 其中, 常规导出函数包括:

```
//客户应用申请 SR, 返回一个 SR 工具箱指针作为 SR 句柄
void PASCAL EXPORT RegSR(HWND hwnd,
  void** pphwndrec )
{
  st_SRH *phwndrec=new st_SRH;
  memset((void*)(phwndrec),0,sizeof(st_SRH));
  phwndrec->hwnd=hwnd;
  theApp.m_lstSockRec.AddTail(phwndrec);
  *pphwndrec=phwndrec;
}
//注销指定的 SR 句柄
void PASCAL EXPORT UnregSR(void* phwndrec)
{
  POSITION pos=
    theApp.m_lstSockRec.Find((st_SRH*)phwndrec);
  if(pos==NULL) return;
  st_SRH *p=theApp.m_lstSockRec.GetAt(pos);
  SAFEDEL(p->pClientSock);
  SAFEDEL(p->pListenSock);
  SAFEDEL(p->pUDPSock[0]);
```

```
        SAFEDEL(p->pUDPSock[1]);
        SAFEDEL(phwndrec);
        theApp.m_lstSockRec.RemoveAt(pos);
    }
    //关闭指定 SR 句柄的指定套接字
    bool PASCAL EXPORT CloseSock(void* phwndrec,
      void** psock )
    {
      AFX_MANAGE_STATE(AfxGetStaticModuleStat
e());
        POSITION pos;
        If    ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
          ==NULL) return FALSE;
        st_SRH *p=(st_SRH*)phwndrec;
        CClientSocket* pSock=(CClientSocket*)*psock;
        switch (pSock->m_dStyle) {
          //按预定义的四种套接字类型分别进行处理
          case SOCK_STYLE_TCP_CLIENT: //客户端
            if(pSock==p->pClientSock){
              SAFEDEL(p->pClientSock);
              pSock=p->pClientSock;}
            else return FALSE;
          break;
          case  SOCK_STYLE_TCP_CHILD:    //服务端
子连接
            p->pListenSock->DisconnectSock(pSock);
          break;
          case SOCK_STYLE_UDP_P2P: //UDP 点对点
          case  SOCK_STYLE_UDP_BROADCAST:  //
UDP 广播
            if (pSock==p->pUDPSock[0]){
              SAFEDEL(p->pUDPSock[0]);
              pSock=p->pUDPSock[0];}
            else if (pSock==p->pUDPSock[1]){
              SAFEDEL(p->pUDPSock[1]);
              pSock=p->pUDPSock[1];}
          break;
          default:
          break;
        }
        *psock=pSock;
        return TRUE;
    }
```

```
    //从指定 SR 句柄的指定套接字接受数据，并返回
套接字类型、
    //收到的字符数等必要信息，该函数在客户应用
窗体收到 DLL 中
    //发出的接受数据消息后调用，phwndrec、psock
指针作为
    // WPARAM、LPARAM 参数随消息传递到客户
窗体
    extern  "C"  __declspec(dllexport)  int  PASCAL
EXPORT Receive(
      void* phwndrec, void* psock, DWORD *psockstyle,
      void *buf,UINT bufsize,UINT* size )
    {
      AFX_MANAGE_STATE(AfxGetStaticModule
State());
        POSITION pos;
        if   ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
          ==NULL) return FALSE;
        st_SRH *p=(st_SRH*)phwndrec;
        CClientSocket* pSock=(CClientSocket*)psock;
        if (pSock==NULL)return 0;
        if   (pSock->m_dStyle==SOCK_STYLE_UDP_
BROADCAST||
          pSock->m_dStyle==SOCK_STYLE_UDP_P
2P)
        {
          CString strAddr;
          *size=pSock->ReceiveFrom(
                buf,bufsize,strAddr,pSock->uiPort);
        }
        else *size=pSock->Receive(buf,bufsize);
        *psockstyle=pSock->m_dStyle;
        return 1;
    }
```

专用函数，以较为复杂的服务端套接字为例说明如下:

```
    //启动指定 SR 句柄的服务端
    bool  PASCAL  EXPORT  StartServer(void*
phwndrec,UINT port )
    {
      AFX_MANAGE_STATE(AfxGetStaticModuleSt
ate());
        POSITION pos;
```

```
    if    ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
      ==NULL){ return FALSE;}
    st_SRH *p=(st_SRH*)phwndrec;
    p->pListenSock=new CListeningSocket(p->hwnd,
p);
    if (!p->pListenSock->Create(port))return false;
    if (!p->pListenSock->Listen())return false;
    return TRUE;
  }
  //关闭指定 SR 句柄的服务端
  bool PASCAL EXPORT EndServer(void*
phwndrec){
    AFX_MANAGE_STATE(AfxGetStaticModuleSt
ate());
    POSITION pos;
    if    ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
      ==NULL)return FALSE;
    st_SRH *p=(st_SRH*)phwndrec;
    SAFEDEL(p->pListenSock);
    return TRUE;
  }
  //经指定 SR 句柄的指定客户端子连接发送数据
  int PASCAL EXPORT Server_Send(
  void* phwndrec, void* psock,void *buf,UINT
size ){
   AFX_MANAGE_STATE(AfxGetStaticModuleSta
te());
    POSITION pos;
    if    ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
      ==NULL)return FALSE;
    st_SRH *p=(st_SRH*)phwndrec;
    CClientSocket* pSock=(CClientSocket*)psock;
    if (!p->pListenSock->SockExists(pSock))return 0;
    pSock->Send(buf,size);
    return 1;
  }
  //查询指定 SR 句柄客户端子连接数
  int PASCAL EXPORT Server_CountConnections(
    void* phwndrec )
  {
    AFX_MANAGE_STATE(AfxGetStaticModuleSt
ate());
    POSITION pos;
    if    ((pos=theApp.m_lstSockRec.Find((st_SRH*)
phwndrec))
    ==NULL) {
      return FALSE;
    }
    st_SRH *p=(st_SRH*)phwndrec;
    return p->pListenSock->CountConnections();
  }
```

其他诸如状态查询等函数及客户端、UDP 套接字专用导出函数较为简单, 略去代码. UDP 套接字相对特殊, 其使用无连接的报文数据协议收发, 又分为广播、点对点两种模式, 因此在 DLL 中的 StartUDP 导出函数中需要一些不同的初始化参数及调用设置函数[10], 其它与客户端套接字基本一致.

至此, 编译完成的 DLL 已可被各类 Win32 应用调用.

4 网络通信DLL应用实例及讨论

以一对一服务\客户端通信应用为例, 无论采取动态或静态链接, 在客户应用中添加加 DLL 预定义消息、引入导出函数(在客户应用中为导入函数), 申请 SR 句柄、启动套接字, 按图 3 所示消息响应机制适时调用相应函数, 即可实现网络通信.

由图 2 可见, 无论是服务端、客户端应用, 都只需要从 DLL 导入少量函数, 并在特定消息响应函数中完成对应操作, 即可建立连接、进行通信. 一对多连接服务端稍复杂一些, 客户应用必须根据一定的握手协议, 建立起所有服务端子连接的跟踪信息列表(即掌握各个连接远端对象身份等信息, 以便确定各种数据的发送目标), 其它与一对一连接无异.

UDP 由于是无连接协议, 尤其在广播模式下, 数据来源多端, 需要客户应用自行构建通信双端身份确认机制并遴选有用数据, 通常将数据源及目的身份信息包含在每一帧数据包中即可; 此外, 由于 UDP 协议的特殊性, 当使用广播模式时, 每次 Send 不超过 512 字节[11], 否则需要分包发送.

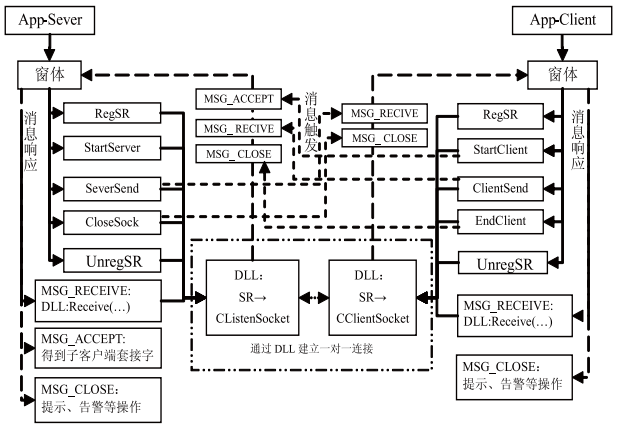

图 2　套接字 DLL 一对一连接应用示例

基于 MFC CSocket 类的通用网络 DLL 具有如下工作特点:

1) 客户应用可申请的 SR 数量理论上仅受限于内存大小及端口数量限制, 且面向对象的设计大大简化了用户管理套接字的复杂度, 比如通过注销函数就可完成 SRH 所管理的各类套接字工具的销毁, 用户仅需自行建立 SR 提供的各种套接字句柄与远程应用之间的关系映射, 即可方便的实现多对多、一对多、点对点、广播形式的通信;

2) 导出函数设计针对处于不同网络通信地位的应用需求作了详细分类, 客户应用只要调用若干功能函数并对个别消息正确响应即可达到应用目的, 如作为客户端的应用只需导入三个通用函数(RegSR、UnregSR、Recieve)、三个专用函数(StartClient、ClientSend、EndClient)、实现两个消息响应函数(MSG_RECEIVE、MSG_CLOSE), 即可建立连接、实现数据收发, 十分简便.

3) 每个 SR 中包括四个套接字工具, 但仅在工具启动时动态创建, 申请一个 SR 即可拥有服务端、客户端、UDP 终端三种工具, 既可满足客户应用的调试、发布等不同需求, 又合理地利用了内存资源.

4) DLL 内建的 SR 管理机制对所有动态生成的 SR 进行自动管理, 即使客户应用忘记注销申请到的资源, 在 DLL 的 DLL_PROCESS_DETACH 机制发生作用时, 也会自动注销所有 SR 及相关资源, 有效避免了内存泄漏等问题.

5) 依靠 DLL 技术的固有特性, 可以建立独立于开发平台语言类型的客户应用模块间的网络通信管道, 该 DLL 的应用可大大简化具有不同开发语言偏好的项目开发组成员在网络通信方面的工作.

5　结论

本文设计的网络通信 DLL 在 Visualstudio 2010 环境下编译生成, 在某型飞行模拟训练设备软件系统中得到应用, 实现了在 VC 6.0、C++ Builder 6.0、Delphi 7.0 三种不同开发环境创建的应用模块间建立本地及局域网内的通信体系, 工作良好, 表明 DLL 模块达到了预期的设计目标.

随着信息技术的飞速发展, 网络通信技术及相关设计方法也随之进步. 然而不少开发人员对于旧的开发环境产生了一定的依赖性, 因而无法得到技术进步所带来的优点. 同时, 受限于不同开发环境底层实现细节的区别, 无法快速实现网络数据在各自产品间的传输. 利用 DLL 技术, 既可在客户应用不做修改的情况下, 通过 DLL 的改进实现技术升级, 也能够破除不同应用间的“沟通”障碍, 无疑为上述问题的解决提供了一条便捷有效的途径.

参考文献

1 博嘉科技.Visual C++6.0 网络编程实作教程[M].希望电子出版社,2001.

2 耿宏运,陈战林,赵宗福,钟显宏等.Delphi 6 组件大全[M].电子工业出版社.2002,7.

3 清宏计算机工作室.C++ Builder 编程技巧(网络与数据库篇)[M].机械工业出版社,2001,1.

4 蔡一兵,赵红宇.用 DLL 连接 Delphi、VC++编制的不同程序.计算机系统应用.1999,8:59–60.

5 邓雪莲,张红霞.基于局域网的通讯程序在 VC 6.0 中的设计与实现.桂林航天工业高等专科学校学报.2005,37(1):95–98.

6 熊华胜,边信黔.VC++6.O 环境下实现基于 UDP 协议的异步广播通信.应用科技.2004,31(2):44–46.

7 Microsoft Visualstudio 2010 文档:Windows Sockets:Datagram Sockets.

8 唐忠莉,孙志卓. Windows 程序设计中的 DLL 技术研究.福建电脑.2007,7:114–115.

9 王西武,阎梅,张殿富.Windows 环境下动态链接库(DLL)程序设计.现代电子技术.2004,183(16):32–33,36.

10 卓红艳,陈进,张家如.VC 环境下基于 UDP 协议的分布式实时监测系统的实现.软件开发与应用.2008,27(12):85–86.

11 Microsoft Visualstudio 2010 文档:sendto Function(Windows).