基于 Windows 的多进程远程 DDE 技术

胡锦敏　彭楚武　袁小坊　傅恪　（长沙湖南大学电气系　410082）

摘要：本文设计了把串行通信服务和 DDE 服务合二为一的服务器进程，各 DDE 客户进程可以通过服务器进程进行本地动态数据交换(DDE)，同时可共享通信服务实现与远程的 DDE 客户进程进行动态数据交换。

关键词：动态数据交换(DDE)　串行通信　Windows 客户/服务器

1. 引言

Windows 是一个多任务环境，它允许多个进程基于消息的驱动下“同时”运行。动态数据交换(DDE)是 Windows 非常有用的技术之一，它使得进程之间可共享信息，进行动态数据交换。Windows 提供了与设备无关的，对用户透明的开发和操作平台，几乎所有的硬件资源都在 Windows 的有效管理之下，用户程序对端口的访问是通过调用 API(应用编程接口)实现的，对于“同时”运行的多个进程，若它们都要访问通信端口则可能使通信混乱，可以设计一个通信服务器专门管理通信并向客户进程提供服务，利用客户/服务器(Client/Server)的软件设计思想实现这一点，使通信软硬件资源很好地得以共享。同时此服务器可作为本地客户进程通信的中介使得本地 DDE 是多对多的。

2. 动态数据交换(DDE)

Windows 的动态数据交换是 Windows 进程之间进行通信的重要手段，它是以共享全局(Global)内存来实现数据交换的，传统应用程序的 DDE 实现是采用在应用程序插入处理各种 DDE 消息(如 WM-DDE-INITIATE, WM-DDE-POKE 等)来实现的，许多著名的程序如 Word, Excel 等都是采用这种方式的。显然这种方式要求应用程序实现 DDE 的协议细节，也就是程序要直接响应、处理和发送 DDE 消息，这样的工作量就很大，当多个服务和话题同时存在于一个服务器时则更显复杂。能够把 DDE 协议的实现封装起来使它对应用程序相对透明正是 DDEML(动态数据管理库)所做到的。DDEML 的核心是 DDEML.DLL，它提供了一些接口函数和事务处理函数使得用户可方便地实现动态数据交换，用户在 DDE 回调函数里处理各种 DDE 事务而不用关心协议细节。

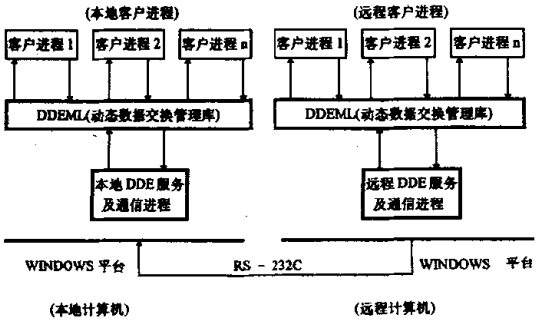
DDEML 中的 DDE 协议有三级，分别为服务(Service)、话题(Topic)和项目(Item)，只有具有相同服务和话题的进程才能进行 DDE 连接。在本文中因为所有需要通信服务的程序都要和服务器建立 DDE 连接，因此它们必须具有相同的服务和话题。这样就会出现多个进程(所有和服务器建立会话的进程)都会收到 DDE 消息，也就是对服务器发出的 DDE 事务消息，所有客户进程的 DDE 回调函数均被调用，这对于响应服务器的广播消息是非常有用的，但是对于非广播消息，则应该只有一个进程处理这些事务消息，而其他的进程应该忽略这些事务。在这里，我们巧妙地使用项目(Item)这一协议来区分不同的进程，后面将详细讨论到这一点。

3. Windows 串行通信技术

前面提到，在 Windows 环境下对通信端口的访问是通过调用 API 实现的，与串行通信有关的几个主要 API 是：SetCommState(…)设置通信口状态，GetCommState(…)获取通信口状态，此两个函数都是对一个叫 DCB(设备控制块)的结构进行操作，DCB 结构定义了通信口的许多参数；OpenComm(…)用来打开指定的通信口并设置接收队列和发送队列的大小，CloseComm(…)关闭一个打开的通信口；ReadComm(…)从接收队列区中读取数据，WriteComm(…)把数据写入发送队列中，FlushComm(…)清除接收或发送队列中的数据；SetCommEventMask(…)设置允许发生的通信事件 GetCommEventMask(…)清除事件记录标志重新允许通信事件的发生；EnableCommNotification(…)设定是否允许 WM-COMMNOTIFY 消息的发生，这些消息有 CN-EVENT，通信事件消息，通过用 SetCommEventMask 设置；CN-RECEIVE，接收消息，当接收队列接收到的字节数达到设置的 cbWriteNotify 时产生；CN-TRANSMIT，发送消息，当发送队列的字节数降到设置的 cbOutQueue 时产生。如果不想产生这些消息则把这两个参数设置为 -1 即可。

4.远程 DDE 的设计与实现

(1)远程 DDE 的体系结构



(2)串行通信的设计与实现

在这里用户数据都是封装在用户数据包 UDP 中传送的,这样当进行串行通信时,用户层的通信协议也封装于 UDP 中,通信数据以 UDP 为单位进行,每次发送接收数据长度就是一个包长 sizeof(UDP),因此这里调用以下的函数设置通信事件的允许:

EnableCommNotification (idComDev, hWnd, sizeof (UDP),1);

这个函数表明当通信接收队列收到的字节数为一个包长时,允许 CN-RECEIVE 消息产生,当发送队列所剩的字节数小于 1 时,也就是发完一个包时允许产生 CN-TRANSMIT 消息。在这里不使用 SetCommEventMask (…)函数设置通信事件的原因有两个,一是不需要别的通信事件发生,二是对因采用三线制的 RS - 232C 串行通信方式省掉了一些信号线,一些事件不能正确产生。对 CN-RECEIVE 的响应在后面给出,对 CN-TRASMIT 的响应只是清除发送队列。

因为每次发送和接收的字节数都是一个长包,故对于串行口的读写操作,可以直接读入一个包:ReadComm (idComDev,(void far *)&rcvudp, sizeof(UDP))和写一个数据包到串行通信口的发送队列中:WriteComm(idComDev, &sendudp, sizeof(UDP))。无论是在读完通信口或是在发送完数据时,最好都使用 FlushComm()函数分别清除接收队列和发送队列。

用户对通信口的初始化通过 Windows 中的对话框进行交互,用户通过选择按钮来选择端口和波特率等,程序根据用户的选择来填充 DCB 然后初试化通信口:

dcb.BaudRate = BaudChoice(波特率选择) dcb.Parity = NOPARITY(无奇偶校验)

dcb.ByteSize = 8(八位数据) dcb.StopBits = ONESTOPBIT(一位停止位),填好 dcb 以后即可调用 SetCommState(&dcb)进行通信口初始化,在初始化前先调用 idComDev = OpenComm (port, sizeof (UDP), sizeof (UDP))函数打开指定端口,此函数还设置了接收和发送队列的大小并获取相应端口(port)的操作柄 idComDev。

(3)用户数据包及多 DDE 客户进程共享服务器的实现

前面提到,要使多个客户进程能连接到同一个服务器上,那么它们的服务与话题都必须相同,这样要区分不同的客户进程只能用项目或在用户数据中加以区分,后者必须在各个客户进程都读取了数据以后才能辨别是否为自己的数据然后决定取舍,这显然浪费时间;而采用前者,客户进程只在发现服务器项目和自己的项目相同时才会读取来自服务器的数据。

这里采用一个叫 UDP(User Data Packet,用户数据包)的结构来描述用户数据信息,其定义如下:

```
typedef struct{
  char idSourceClient[16]; //源客户进程标识
  char idTargetClient[16]; //目的户进程标识
  BYTE idServer; //服务器标识 ,0:本地,1:远程
  BYTE UserServiceNo; //用户定义的服务号
  int PacketNo; //数据包号
  BOOL NextPacket; //有无后续包标志
  int DataLenth; //数据长度
  BYTE Data[DATALEN]; // 数据区
 }UDP;
```

这个结构主要考虑进行远程动态数据交换时的协议实现而定义的。下面详细描述一次远程动态数据交换过程及其实现:

当客户进程和服务器进程都用 DdeInitalize 函数初始化同一服务和话题的会话(Conversation)后,客户进程调用 DdeConnect 函数连接服务器,并用 DdeClientTransaction 函数发送数据链路建立请求(XTYP-ADVSTART),当服务器收到请求后即可做某种标志进行记录请求并响应。在此以后,若远地客户进程通过远地服务器向本地客户进程发送数据,则在本地服务器收到数据后,并用以下程序实现项目(Item)的动态创建,通过调用 DdePostAdvise(…)函数产生 XTYP-ADCREQ 消息,使服务器 DDE 回调函数被调用,进行 DDE 事务处理。

……

```
    case CN-RECEIVE: //收到数据
    ReadComm(idComDev, (void far *) &rcvudp, sizeof
(UDP));
    //从串行口接收队列读入一个用户数据包
    if(HotLinkCount>0) //如果存在数据链路
    {
    DdeFreeStringHandle(idInst, hszItem); //释放前一
个项目操作柄
    hszItem = DdeCreateStringHandle(idInst, (char *)
rcvudp.idTargetClient, CP-WINANSI);
    //以目标客户进程标识(这里是目标进程名,由发送
者填好)创建新的项目
    DdePostAdvise(idInst, hszTopic, hszItem);
    //请求 DDEML 向服务器发 XTYP-ADVREQ 消息
    }
```

下面的程序段是服务器的 DDE 回调函数的实现，DDEML 通过回调这个函数来处理各种 DDE 事务：

```
HDDEDATA EXPENTRY DDECallback ( WORD wType,
WORD wFmt,
        HCONV hConv, HSZ hsz1, HSZ hsz2, HDDEDATA
        hData,
        DWORD dwData1, DWORD dwData2 )
    {
      switch ( wType )
      {
        case XTYP-CONNECT: //连接请求
          if ( hsz2 = = hszService ) return ( (HDDEDATA)
TRUE );
            else return ( (HDDEDATA) FALSE );
        case XTYP-ADVREQ : //向客户提供新的数据
          hData = DdeCreateDataHandle ( idInst, &rcvudp,
              sizeof (UDP), 0L, hszItem, wFmt, 0 );
          //创建数据操作柄,相当于获取数据块的全局指针
          if ( hData ! = NULL ) return ( hData ); else re-
turn ( NULL );
        case XTYP-ADVSTART : //开始建立数据链路
          Hotlinkcount + + ; //数据链路计数器加 1
          return (HDDEDATA)1;
        case XTYP-ADVSTOP : //有进程请求拆除数据链路
          Hotlinkcount - - ; //数据链路计数器减 1
          break;
      case XTYP-REQUEST: …//客户请求数据
      case XTYP-EXECUTE: …//客户向服务器发执行命令
      case XTYP-POKE: //客户向服务器发数据
        if ( hsz1 = = hszTopic )
        { deGetData ( hData, (LPBYTE) &sendudp, sizeof
(UDP), 0L);
        //读入数据放到发送数据包中
          if(hData! = NULL)
          { if (sendudp.DataLenth! = 0 )
            if(sendudp.idServer = = 0) //发给本地客户
            { DdeFreeStringHandle(idInst, hszItem);
            //释放前一个项目操作柄
            hszItem = DdeCreateStringHandle(idInst, (char *)
                sendudp.idTargetClient, CP-WINANSI);
            //以目标客户进程标识创建新的项目
            hData = DdeCreateDataHandle ( idInst, &sendudp,
                sizeof (UDP), 0L, hszItem, wFmt, 0 );
            return ( hData );
          }else//发给远地服务器
                WriteComm ( idComDev, &sendudp, sizeof
(UDP));
            //把数据包写入串行通信口的发送队列
            return ( (HDDEDATA) DDE-FACK );
            }
          }else return ( (HDDEDATA) NULL );
          }break;
        case XTYP-DISCONNECT:
          hConv = NULL; break; //拆除连接结束会话
        }//End Switch
      return ( (HDDEDATA) NULL );
    }//End CallBack Function
```

程序中的 HotLinkCount 用来记录已建立的链路数，当服务器中收到 XTYP-ADVSTRT 时，说明有一个客户进程要与服务器建立数据链路，此时 HotLinkCount 加 1，当服务器收到 XTYP-STOP 时，说明客户进程要拆除链路，此时 HotLinkCount 减 1，当其变为 0 时，说明已不存在数据链路，服务器将不在调用 DdePostAdvise 函数，也就不会再向客户进程传送数据。

DDE 客户的 DDE 回调函数如下所示：

```
HDDEDATA EXPENTRY DDECallback ( WORD wType,
WORD wFmt, HCONV hConv, HSZ hsz1, HSZ hsz2, HDDE-
DATA hData, DWORD dwData1, DWORD dwData2 )
    {
      switch ( wType )
```

```
        |
        case XTYP-DISCONNECT: //拆除连接,结束会话
          hConv = NULL; return ( (HDDEDATA) NULL
);
        case XTYP-ERROR: break;
        case XTYP-ADVDATA : //服务器有新的数据提供
        if(HotLinkFlag = = FALSE) break;//如果没有建立
数据链路则退出
        if( hConv ! = NULL && hsz1 = = hszTopic)
          if(hsz2 = = hszPrivateItem||hsz2 = = hszItem)
            //是本进程私有的项目或广播项目
          | DdeGetData ( hData, (LPBYTE)&rcvudp, sizeof
(UDP), 0L );
          //读出数据放到用户数据包中供处理
          return ( (HDDEDATA) DDE-FACK );
          |
        return ( (HDDEDATA) NULL );
      case XTYP-XACT-COMPLETE:
        // compare transaction identifier, indicate transaction
complete
        break;
      | //End of Switch
    | //End of CallBackFunction
```

程序中的 HotLinkFlag 用来标记本进程是否和服务器建立了数据链路,当程序调用 DdeClientTransaction(NULL, 0, hConv, hszItem, CF-TEXT, XTYP-ADVSTART | XTYPF-ACKREQ, 1000, &dwResult)和 DdeClientTransaction(NULL, 0, hConv, hszPrivateItem, CF-TEXT, XTYP-ADVSTART | XTYPF-ACKREQ, 1000, &dwResult)时,HotLinkFlag 变为 TRUE,前一个函数为请求建立广播链路,后者为请求建立进程的私有链路;当进程调用 DdeClientTransaction(NULL, 0, hConv, hszItem, CF-TEXT, XTYP-ADVSTOP , 1000, &dwResult)和 DdeClientTransaction(NULL, 0, hConv, hszPrivateItem, CF-TEXT, XTYP-ADVSTOP , 1000, &dwResult)时 HotLinkFlag 为 FALSE,这两个函数用来释放已建立的数据链路。函数中的广播主题和私有主题的句柄用以下两个函数分别实现 hszItem = DdeCreateStringHandle (idInst, "BroadCast", CP-WINANSI)当进程要广播自己的数据时,发送方只要对 UDP 中 idTargetClient 项赋予"BroadCast",那么在服务器收到此数据包时自然会动态创建广播项目,使得所有客户进程均可获取广播数据。

hszPrivateItem = DdeCreateStringHandle (idInst, (LPSTR) szAppName, CP-WINANSI) 使用了进程名(szAppName)来作为私有项目来保证其私有性。

5. 结论

本文设计的多进程远程 DDE 实现了本地和远程多个 DDE 客户进程进行一对多或多对多的动态数据交换,所有客户进程均可方便地使用串行通信功能,它们只需向通信服务器提交事务或响应服务器的数据而无须管理通信,这一点充分体现了 Client/Server 的思想。用户通过对 UDP 的 UserServiceNo 进行定义可以使进程间支持多种用户服务。如果通信服务器的通信服务程序改为网络通信,则可使所有运行在网络计算机上的 DDE 客户进程都可进行动态数据交换,这样可以实现事务的分布处理和动态信息的全局共享,此时给每台计算机赋予不同的 idServer,即可区分不同的 DDE 服务器,从而区分不同的客户进程。本文的远程 DDE 设计与实现有明显的现实意义。

参考文献

[1] Richard Wilton. Microsoft Windows 软件开发环境与技术. 北京:清华大学出版社,1993.4.

[2] 李元通,等. Borland C++ 3.1 Windows API 快速参考. 北京:北京航空航天大学 出版社,1994.6.

[3] 何立起. Borland C++. 北京:人民邮电出版社,1994.6.

(本稿件收到日期:1997 年 5 月)