利用 Windows API 函数实现串行通信的方法

李晓磊　刘长有　（山东工业大学自动化系　250061）

摘要：本文介绍了一种利用调用 Windows API 函数来实现串行通信的方法，并介绍了两种简便实用的通信进程管理机制。

关键词：串行通信　Windows API　定时器　事件驱动

一、引言

在工业控制系统中，经常遇到工业控制计算机与工业 PLC 或各种智能仪表、智能调节器之间的通信问题，它们一般采用 RS－232/422/485 的串行通信方式，这种通信方式一般连接形式比较简单，造价比较低廉，且传输距离可达 1.2Km 左右，因此，在中小规模的工控系统中得到了广泛的采用。

在早期的工控系统中，常采用 C/C＋＋或汇编语言通过工控机串行口的中断调用来实现，这种实现方法只能是基于 DOS 操作系统。随着 Windows 操作系统在工控系统中被日益广泛的采用，如何在 Windows 系统中实现串行通信成了广大工控软件开发者所关心的问题。下面，将介绍一下利用调用 Windows API(16 位)函数来实现串行通信的方法。

二、串行通信的有关函数及数据结构

由于 Windows 操作系统支持多任务操作，要求应用程序不要独占系统资源，那些对硬件设备直接进行中断或其他功能调用的操作都有可能造成系统的崩溃或死机，因此，在设计串行通信的程序时，我们采用了调用 Windows 提供的应用程序接口(API)的方法。

Windows 系统的 API 中为串行通信提供了一系列的函数，在编程时不必考虑 UART 的地址以及 IRQ 号等，只需要通知 Windows 要打开端口的名字(如 COM1、COM2 等)和要求的波特率、校验方式、握手信号等信息，Windows 就可以负责对端口进行初始化和设置、发送、接收等操作。表 1 中列出了 Windows API(16 位)中常用的通信函数：

表 1　常用 Windows 串行通信 API 函数

OpenComm()	打开一个端口
SetCommState()	配置一个端口
GetCommState()	获得端口的当前设置
BuildCommDCB()	构造一个 DCB 结构
ReadComm()	读取串行口的内容
WriteComm()	把数据送入串行口
GetCommError()	获得通信错误和状态信息
CloseComm()	关闭一个串行口

在 Windows 中定义了一个很重要的结构 DCB(Device Control Block)，它包含了串行通信的各种参数：

```
typedef struct tagDCB
{
BYTE Id;    //设备的识别号
UINT BaudRate;    //波特率
BYTE ByteSize;    //字长(4~8 位)
BYTE Parity;    //校验方式，定义为 EVENPARITY, MARKPARITY, NOPARITY, ODDPARITY, SPACEPARITY
BYTE StopBits;    //停止位位数，定义为 ONESTOPBIT, ONE5STOPBITS, TWOSTOPBITS
UINT Cts Timeout;    //等待 CTS 信号的超时时间
UINT DsrTimeout;    //等待 DSR 信号的超时时间
UINT fParity;    //是否进行奇偶校验
UINT fDtrDisable;    //是否关闭 DTR (data－terminal－ready) 信号
UINT fRtsDisable;    //是否关闭 RTS(request－to－send)信号
... ...
```

```
    char EvtChar;   //定义用来发送事件信号的字符
    UINT fChEvt;   //如果收到 EvtChar 定义的字符,则向主程序发送事件信号
    ... ...
} DCB;
```

通过填充该 DCB 结构,可以实现对串行口的设置和确定通信的方式。

三、串行通信的进程管理

在工控系统中,一台主控计算机可同时挂接数台智能仪表和工业 PLC,而主控计算机的串行口数目一般可扩充为 4～9 个,因此,通常必须由多台仪表共用一个串行口,各仪表通过设置不同的机号来相互区分,而在同一时间内一个串行口只能与其中的一台仪表进行通信,这就需要有一个合理的通信进程管理机制。

1. 定时通信进程管理

对于此类远距离的串行通信,一般没有硬件的握手信号,当主控计算机给仪表发送出一帧命令信息后,与该串行口相连的智能仪表都将接收到该命令信息帧,根据命令信息帧中的机号信息,将确定其中一台与主控计算机通信,该台仪表翻译出命令的内容后,执行该命令,进行相应的数据处理后形成响应数据帧回送给主控计算机。在信息帧传送的过程中,该串行口不得再被占用,即主控计算机接收到仪表的响应信息帧后才能再次发送命令信息帧。

针对以上仪表串行通信的特点,我们可以利用 Windows 下的定时器定时向仪表网络循环发送命令和接收仪表的响应,而不必一直监测串行口的状态,这样就可以腾出时间执行其他任务。

在窗口的回调函数中可以这样处理:

```
case ID-STARTCOMM;   //开始进行串行通信
int ComDev;
DCB dcb;
ComDev = OpenComm(ComNo, 512, 512);   //打开串行口
if(ComDev<0){   //打开串行口错误,发送出错消息
    SendMessage(hWnd, WM-COMMAND, COM-ERROR, 0L);
    break;}
GetCommState(ComDev, &dcb);   //获取串行口的信息
//填充 DCB 结构
dcb.BaudRate = user-BaudRate;   //设波特率
dcb.ByteSize = user-Byte;   //设字长
dcb.StopBits = user-StopBits;   //设停止位数
dcb.Parity = user-Parity;   //设奇偶校验方式
dcb.fRtsDisable = 1;   //不使用 RTS 信号
dcb.fDtrDisable = 1;   //不使用 DTR 信号
dcb.CtsTimeout = 0;   //不等待 CTS 信号
dcb.DsrTimeout = 0;   //不等待 DSR 信号
//设置串行口状态
SetCommState(&dcb);
... ...
//组装 1 号设备的命令信息帧并将其赋给 lpBufl 发送出去
WriteComm(ComDev, lpBufl, strlen(lpBufl)
Sending = TRUE;   //设发送数据标志
... ...
break;
case ID-ENDCOMM;   //结束串行通信
... ...
CloseComm(ComDev);   //关闭串行口
... ...
break;
case WM-TIMER;
  switch(wParam){
    case COMM-TIMER;
        if(Sending)
        {                    //读取串行口内容
    ReadComm(ComDev, lpBuf2, 512);   //翻译响应信息帧的内容并处理
        ... ...
    Sending = FALSE;}   //清除发送数据标志
        //设备号加 1 并组装命令信息帧后赋给 lpBufl
        ... ...
        //发送下一设备的命令信息帧
        WriteComm(ComDev, lpBufl, strlen(lpBufl)
        Sending = TRUE;   //设发送数据标志
      break;}
```

本通信进程的管理方法比较容易实现,但定时器的定时时间需要适当的选择,一般不应少于智能仪表所能响应的最短时间;由于在 Windows 操作系统中,WM-TIMER 消息是优先级别较低的一类消息,因此,在实际运行过程中它的到来时早时迟,并非严格按照规定的定时时间,有可能造成通信过程中的时间冲突,有必要在程

序内加入数据发送标志，以避免通信错误的发生。

2. 事件驱动通信进程管理

在主控计算机发出命令信号后，如果一直监测串行口的状态以等待仪表的响应，那就什么事也做不成，白白浪费 CPU 的时间，如果每隔一段时间监测一次串行口状态，则有可能过早或过晚，有没有更好的解决方法呢，那就是事件驱动通信方式。事件驱动通信是在通信过程中有事件发生时(如收到一个字符、奇偶校验错误等)，向应用程序发送一定的消息以给予通知。这样，在我们的串行通信程序中，当主控计算机发出命令信号后，可以去做其他事，仪表的响应信号到来时，Windows 操作系统将发送消息通知我们，串行通信程序这时再处理相应的通信进程，因此，这是一种有效的工作方式，可以大大提高系统运行的效率。

在 Windows 的 API 中，有两个函数用于设置通信事件：

·EnableCommNotification()函数

该函数用于通知接收到字符串或字符串已被发送，其调用格式如下：

```
BOOL EnableCommNotification(
    int ComDev,    //通信设备号
    HWND hWnd,     //要接收信息的窗口句柄
    int cbWriteNotify,    //希望接收的字符数量
    int cbOutQueue)    //输出队列中的最少字符量
```

·SetCommEventMask()函数

如果想在其他通信事件发生时收到相应的消息，可以将上面的函数 EnableCommNotification()和函数 SetCommEventMask()放在一起使用。SetCommEventMask()函数的调用格式如下：

```
WORK FAR * SetCommEventMask(
    int ComDev,    //通信设备号
    int nEvtMask)    //事件类别字
```

nEvtMask 所定义的事件类型如下：

EV-BREAK　收到断开信号
EV-CTS　CTS 改变状态
EV-DSR　DSR 改变状态
EV-ERR　出现帧、过冲或奇偶校验错误
EV-RING　接收到响铃标志
EV-RSLD　载波检测改变状态
EV-RXCHAR 收到一个字符
EV-RXFLAG 在 DCB 结构中定义的 EvtChar 字符到达
EV-TXEMPT 所有发送队列中的字符已发送

针对仪表通信数据信息帧的格式，其最后一个字符大都是以 CR(回车)作为结束符，因此，我们可以以回车符的到达作为一个事件，其设置方法如下：

·在对串行口进行初始化时，其 DCB 结构中需设置字符事件允许并填充相应的字符：

```
dcb.fChEvt = 1; //允许字符到来事件的触发
dcb.EvtChar = '/r'; //触发事件的字符为回车符
```

·设置通信事件屏蔽字，并允许操作系统在事件发生时，发送消息通知接收信息的窗口：

```
setCommEventMask(COmDev, EV-RXFLAG);
EnableCommNotification(ComDev, hWnd, -1, -1);
```

·在接收事件窗口的回调函数中需对发送来的事件消息进行如下处理：

首先，判断一下 lParam 参数中有无 CN-EVENT 特征字，然后调用 GetCommEventMask()函数将事件触发标志复位，表明我们已经收到消息，接下来就可以从串行口的输入队列中读取信息了。

```
switch(Message){
  case WM-COMMNOTIFY:
    if(CN-EVENT&LOWORD(lParam)!  = CN-'EVENT)
      return(FALSE);
    GetCommEventMask(comdev, EV-RXFLAG);
    ReadComm(comdev, lpBuf, 512);
  break;
  }
```

四、结束语

以上方法可以在 Borland C++ 或 Visual C++ 环境下很顺利的实现，从实际运行效果来看，只要程序处理得当，在各种版本的 Windows 3.x 和 Windows 95 中都能可靠的长时间运行。需要说明的是，以上介绍的是 16 位的 Windows API 调用，要想开发完全 32 位的 Windows 95/NT 应用程序，以上介绍不适用，请参考 WIN32 API 的调用，本文暂不介绍。

参考文献

[1] 张国峰 编著，Windows 应用程序设计—原理、方法和技巧，电子工业出版社，1994.
[2] Peter W. Gofton 著，王仲文等译，精通串行通信，电子工业出版社，1995.
[3] 魏彬等编译，Windows 3.0 软件的开发指南(三)—库函数及数据结构，清华大学出版社，1991.

(来稿时间：1998 年 5 月)