[用Windows API 编写串口通讯程序](http://blog.csdn.net/bingcaihuang/article/details/5668474)

标签： [windows](http://www.csdn.net/tag/windows)[api](http://www.csdn.net/tag/api)[通讯](http://www.csdn.net/tag/%e9%80%9a%e8%ae%af)[mfc](http://www.csdn.net/tag/mfc)[null](http://www.csdn.net/tag/null)[microsoft](http://www.csdn.net/tag/microsoft)

2010-06-13 10:05 5312人阅读 [评论](http://blog.csdn.net/bingcaihuang/article/details/5668474#comments)(4) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/bingcaihuang/article/details/5668474#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

VC++（19） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg windows（5） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

最近在做一个PC机上和ARM机串口通讯的程序。

    实际上，我并没有在VC上编写过串口程序。记得大一下学期的课程实践上倒是在DOS环境下做个简单的串口通讯，可是就是因为太简单了，而且是DOS那种独占式的进程，所以现在要搬到VC和MFC界面应用程序环境中，难度还是有的，我一时没有头绪。

    我首先当然想到用ActiveX控件了。曾听说过Microsoft曾做过一个ActiveX控件，用来简化在MFC中进行的串口编程。找了点资料，又去图书馆找了本书，试了两天，结果以失败告终。网上说，用那个MSComm控件进行串口编程是最简单的，可是我仍然没有成功，可见我有多愚笨！也不知道哪儿出现问题了，总之，由于时间紧迫，我不得不选择其他方案。MSComm的使用，我想等这个任务完成之后，我会回过头来再看看的，到时候再写篇文章来向大家说明。

不用MSComm控件，那看起来只能是使用Windows API了，因为MFC貌似没有什么类封装了串口API函数的。

    用Windows API 编写串口程序本身是有巨大优点的，因为控制能力会更强，效率也会更高，而且对于那些纯绿色软件追求者来说，没有ActiveX控件比什么都重要——呵呵，我也是这么认为。

    API编写串口，过程一般是这样的：

    1、  创建串口句柄，用CreateFile；

    2、  对串口的参数进行设置，其中比较重要的是波特率（BaudRate），数据宽度（BytesBits），奇偶校验（Parity），停止位（StopBits），当然，重要的还有端口号（Port）；

    3、  然后对串口进行相应的读写操作，这时候用到ReadFile和WriteFile函数；

    4、  读写结束后，要关闭串口句柄，用CloseFile；

下面依次大致讲讲个步骤的过程：

    第一步，从字面上去理解，大家也可以发现CreateFile实际上表明Windows是把串口当作一个文件来处理的，所以它也有文件那样的缓冲区、句柄、读写错误等，不同的是，这个文件名字只有固定的几个（一般为四个），而且始终存在（EXSITING），而且在调用CreateFile的时候请注意它的参数。CreateFile函数原型如下：

    HANDLE CreateFile(LPCTSTR lpFileName,

                      DWORD dwDesiredAccess,

                      DWORD dwShareMode,

                      LPSECURITY\_ATTRIBUTES lpSecurityAttributes,

                      DWORD dwCreationDisposition,

                      DWORD dwFlagsAndAttributes,

                      HANDLE hTemplateFile );

lpFileName是你需要创建的端口号，默认情况下是COM1；dwDesiredAccess是表明你想让你创建的串口以何种方式存在于你的应用程序中，因为串口通常是可读可写的，所以这里必须设置为GENERIC\_READ|GENERIC\_WRITE；dwShareMode是用来设置串口共享属性的，因为串口属于临界资源，当然不能共享，所以这里也必须设置为0；lpSecurityAttributes是设置安全模式，一般采用默认的安全模式就可以了，选择NULL；dwCreationDisposition是设置是否打开新的“文件”（上面说过了，Windows是把串口等端口当作文件来处理的），因为串口属于硬件端口，当然不能随便重复创建，所以这里必须告诉Windows，每次创建的时候必须使用已经存在的串口，所以这里设置OPEN\_EXSITING；dwFlagsAndAttributes，这个参数可以设置的值比较多，大家若需要深入了解可以查找MSDN，这里因为我们接下去要做的是异步通讯，所以需要设置FILE\_FLAG\_OVERLAPPED；最后一个参数hTemplateFile是指定模板文件，串口没有模板，选择NULL；

     所以最后我们设置的CreateFile函数如下：

            m\_hCom=CreateFile(m\_sPort,

                              GENERIC\_READ|GENERIC\_WRITE,

                              0,

                              NULL,

                              OPEN\_EXISTING,

                              FILE\_FLAG\_OVERLAPPED,

                              NULL);

在创建完串口后，最后进行句柄测试：

        if(m\_hCom==INVALID\_HANDLE\_VALUE)

          {

                 AfxMessageBox("打开串口失败!");

                 return;

           }

    上面说到了异步，那什么是异步呢？异步是相对同步这个概念而言的。异步，就是说，在进行串口读写操作时，不用等到I/O操作完成后函数才返回，也就是说，异步可以更快得响应用户操作；同步，相反，响应的I/O操作必须完成后函数才返回，否则阻塞线程。对于一些很简单的通讯程序来说，可以选择同步，这样可以省去很多错误检查，但是对于复杂一点的应用程序，异步是最佳选择；

第二步，设置串口，并创建串口线程。串口有很多的属性，上面也已经介绍了一些最重要的参数。这里不得不介绍一个重量级的数据结构DCB：

typedef struct \_DCB { // dcb

    DWORD DCBlength;           //DCB结构体大小

    DWORD BaudRate;           //波特率

    DWORD fBinary: 1;    //是否是二进制，一般设置为TRUE

    DWORD fParity: 1;//是否进行奇偶校验，我做的是ARM嵌入式，所以FALSE

    DWORD fOutxCtsFlow:1; //CTS线上的硬件握手

    DWORD fOutxDsrFlow:1;  //DSR线上的硬件握手

    DWORD fDtrControl:2;  //DTR控制

    DWORD fDsrSensitivity:1;

    DWORD fTXContinueOnXoff:1;

    DWORD fOutX: 1;      //是否使用XON/XOFF协议

    DWORD fInX: 1;        //是否使用XON/XOFF协议

    DWORD fErrorChar: 1;   //发送错误协议

    DWORD fNull: 1;

    DWORD fRtsControl:2;

    DWORD fAbortOnError:1;

    DWORD fDummy2:17;

    WORD wReserved;

    WORD XonLim;    //设置在XON字符发送之前inbuf中允许的最少字节数

    WORD XoffLim;   //在发送XOFF字符之前outbuf中允许的最多字节数

    BYTE ByteSize;  //数据宽度，一般为8，有时候为7

    BYTE Parity;    //奇偶校验

    BYTE StopBits;   //停止位数

    char XonChar;   //设置表示XON字符的字符,一般是采用0x11这个数值

    char XoffChar; //设置表示XOFF字符的字符,一般是采用0x13这个数值

    char ErrorChar;

    char EofChar;

    char EvtChar;

    WORD wReserved1;

} DCB;

     大家不要被这个结构体“强大”的身躯所吓倒，我这里只是为了向大家展示一下DCB的所有内部数据成员，其实我们真正在串口编程中用到的数据成员没有几个。

用DCB进行串口设置时，应先调用API’函数GetCommState，来获得串口的设置信息：

           GetCommState(m\_hCom, &dcb)；

然后在需要设置的地方对dcb进行设置，然后再末尾调用

           SetCommState（m\_hCom，&dcb）

就可以了，还是比较方便的。然后调用SetCommMask,用来指定程序接收特定的串口事件,调用SetupComm函数,设置串口缓冲区大小:

           SetCommMask(m\_hCom, EV\_RXCHAR);

           //EV\_RXCHAR表示当有字符在inbuf中时产生这个事件

           SetupComm(m\_hCom, MAXBLOCK, MAXBLOCK);

    还有，串口因为是I/O操作，可能会产生错误，这时候需要设置超时限制，以避免阻塞现象。设置超时设置需要一个结构体COMMTIMEOUTS：

typedef struct \_COMMTIMEOUTS {

    DWORD ReadIntervalTimeout; //两个字符之间的超时设置

    DWORD ReadTotalTimeoutMultiplier; //读操作时总的超时系数

    DWORD ReadTotalTimeoutConstant; //读操作时总的超时常数

    DWORD WriteTotalTimeoutMultiplier; //写操作时总的超时系数

    DWORD WriteTotalTimeoutConstant; //写操作时总的超时常数

} COMMTIMEOUTS,\*LPCOMMTIMEOUTS;

我的设置如下：

COMMTIMEOUTS timeouts;

       timeouts.ReadIntervalTimeout=MAXDWORD;

       timeouts.ReadTotalTimeoutConstant=0;

       timeouts.ReadTotalTimeoutMultiplier=0;

       timeouts.WriteTotalTimeoutConstant=50;

       timeouts.WriteTotalTimeoutMultiplier=2000;

       SetCommTimeouts(m\_hCom, &timeouts);

     这里将ReadIntervalTimeout设置为最大字节数，.ReadTotalTimeoutConstant和ReadTotalTimeoutMultiplier都设置为0，表示不设置读操作超时，也就是说读操作瞬间完成，不进行等待。

接下去是一步比较关键的操作，建立工作者线程，用来监听串口消息，如果发现inbuf中有接收到的字符，及时通知相应处理函数进行处理。

调用MFC全局函数AfxBeginThread建立线程。好的线程应该短小精悍，所以，我在这个线程里面其实什么事也不做，只是起到通知别的函数的作用。

m\_pThread=AfxBeginThread(CommProc,

                         this,

                         THREAD\_PRIORITY\_NORMAL,

                         0,

                         CREATE\_SUSPENDED,// 挂起线程

                         NULL);

m\_pThread就是指向我新创建的线程的指针。

线程函数如下，有点长，但是已经是最简单的线程了：

//串口线程

UINT CommProc(LPVOID lParam)

{

       COMSTAT commstat;//这个结构体主要是用来获取端口信息的

       DWORD dwError;

       DWORD dwMask;

       DWORD dwLength;

       OVERLAPPED overlapped;

//OVERLAPPED结构体用来设置I/O异步，具体可以参见MSDN

       memset(&overlapped, 0, sizeof(OVERLAPPED));

       //初始化OVERLAPPED对象

       overlapped.hEvent=CreateEvent(NULL, TRUE, FALSE, NULL);

       //创建CEvent对象

       CUartDlg\* dlg=(CUartDlg\*)lParam;

       if(dlg->m\_hCom==NULL)

       {

              AfxMessageBox("串口句柄为空!");

              return -1;

       }

       while(dlg->m\_bConnected)

       {

              ClearCommError(dlg->m\_hCom, &dwError, &commstat);

              if(commstat.cbInQue)

       //如果串口inbuf中有接收到的字符就执行下面的操作

              {

                     WaitForSingleObject(dlg->m\_hPostMsgEvent, INFINITE);

                     //无线等待。。。

                     ResetEvent(dlg->m\_hPostMsgEvent);

                     //设置CEvent对象为无信号状态

                     ::PostMessage(dlg->m\_hWnd, WM\_COMMSG, EV\_RXCHAR, 0);

                     //发送特定信息，用来通知特定函数进行处理

                     continue;

              }

              if(!WaitCommEvent(dlg->m\_hCom, &dwMask, &overlapped))

              {

                     if(GetLastError()==ERROR\_IO\_PENDING)

//如果操作被挂起，也就说正在读取或这在写，则进行下面的操作

                            GetOverlappedResult(dlg->m\_hCom,

                                &overlapped, &dwLength, TRUE);

                            //无限等待这个I/O操作的完成

                     else

                     {

                            CloseHandle(overlapped.hEvent);

                            return (UINT)-1;

                     }

              }

       }

       CloseHandle(overlapped.hEvent);

       return 0;

}

    因为是多线程，所以，要注意的是对临界资源的访问问题，也就是说互斥问题，避免死锁现象的发生。所以，在这个线程中多次使用了事件对象CEvent，通过它来标志串口有没有被占据，和标志是否正在进行读取串口（串口无法同时进行读写操作）。具体的大家可以看我的代码注释 : )

    这是我对这个线程发出的消息进行处理的函数：

void CUartDlg::OnComMsg(WPARAM wParam, LPARAM lParam)

{

       char buf[MAXBLOCK/4];

       CString str;

       int nLength;

       int nStartChar, nEndChar;

       if(!m\_bConnected ||      (wParam & EV\_RXCHAR)!=EV\_RXCHAR)

 // 是否是EV\_RXCHAR事件?

       {

              SetEvent(m\_hPostMsgEvent);

              // 允许发送下一个线程读取消息

              return;

       }

       nLength=ReadComm(buf,100);

       buf[nLength]='/0';

       if(nLength)

       {

    //IDC\_EDIT\_EDIT是我在一个对话框上一个CEdit控件的ID号，大家可设置成

    //自己的控件ID号

              GetDlgItem(IDC\_EDIT\_EDIT)->SetFocus();

              CString str(buf);

              m\_strMessage+=str;

              UpdateData(FALSE);

              CEdit\* pEdit=(CEdit\*)GetDlgItem(IDC\_EDIT\_EDIT);

              pEdit->GetSel(nStartChar, nEndChar);

              pEdit->SetSel(nStartChar-2, nEndChar-2);

       }

       SetEvent(m\_hPostMsgEvent); // 允许发送下一个线程读取消息

}

    第三步，已经建立好了工作者线程，那么接下去我们就可以进行串口的读写操作了。

DWORD CUartDlg::ReadComm(char \*buf, DWORD dwLength)

{

       COMSTAT comstat;

       DWORD dwError;

       DWORD length;

       DWORD dwByteReaded;

       ClearCommError(m\_hCom, &dwError, &comstat);

       length=min(comstat.cbInQue, dwLength);

       if(!ReadFile(m\_hCom, buf, length, &dwByteReaded, &m\_osRead))

              return 0;

       return dwByteReaded;

}

这是读串口函数；

DWORD CUartDlg::WriteComm(char \*buf, DWORD dwLength)

{

       BOOL fState=FALSE;

       DWORD length=0;

       COMSTAT ComStat;

       DWORD dwErrorFlags;

 //ClearCommError是用来清除Comm中的错误，从而可以在下面的代码通过

 //GetLastError抓取错误

       ClearCommError(m\_hCom,&dwErrorFlags,&ComStat);

       fState=WriteFile(m\_hCom,buf,dwLength,&length,&m\_osWrite);

       if(!fState)

       {

              if(GetLastError()==ERROR\_IO\_PENDING)

              {

                    SetEvent(m\_osWrite.hEvent);

                     while(!GetOverlappedResult(m\_hCom,&m\_osWrite,&length,TRUE))// 等待

                     {

                            if(GetLastError()==ERROR\_IO\_INCOMPLETE)

                                   continue;

                     }

              }

              else

                     length=0;

       }

       return length;

}

    这是写串口函数。这两个函数其实本质是一样的，操作过程也近似，大家可以参考着写。

    第四步，好了，现在一个串口程序大致上已经完工了，呵呵，是不是挺繁琐？确实，用windows API函数进行硬件层次的编程都是比较繁琐的。还有一点，就是在结束程序的时候，千万不要忘了关闭串口的句柄，否则容易造成内存泄露的问题！

void CUartDlg::OnClose()

{

       // TODO: Add your message handler code here and/or call default

       if(m\_bConnected)

       {

              m\_bConnected=FALSE;

              SetEvent(m\_hPostMsgEvent);

              SetCommMask(m\_hCom, 0);

              WaitForSingleObject(m\_pThread->m\_hThread, INFINITE);

              m\_pThread=NULL;

              CloseHandle(m\_hCom);

       }

       CDialog::OnClose();

}

    总结：串口通讯应用非常广泛，特别是在硬件设计领域，更是没有串口不行。但是VC爱好者中懂串口编程的不多。我想，还是大家比较喜欢上层的东西吧。

在编写这个程序的时候，大体上已经写的差不多了，可以读取ARM机串口发送过来的字符，并显示出来，但是就是不能发送通过串口发送命令给ARM机。我在网上看了好多信息，也查阅了很多相关书籍，仍然没有找到答案。最后只能自己埋头一句代码一句代码得找，找了两天，还是没有找到，人差不多已经到了崩溃的边缘了，最后突然灵感发现，将错误锁定在dcb参数设定上。原来，我在dcb硬件握手参数上设置错了，那个fOutxCtsFlow参数应该设置成FALSE，否则串口读操作将一直处于阻塞状态。当时我差不多想亲吻每一个人，呵呵，程序员常常是带有一点病态的 : ) 希望大家以后不要犯我这样的错误。

    还有，就是因为基于对话框（Dialog-based）的应用程序是不能接收WM\_CHAR这个消息的，所以我在进行写串口操作时，重载了PreTranslateMessage这个函数，然后在这个函数内部对消息进行分检处理，起到了很好的效果。大家可以试一试。

因为一大早的，寝室里居然因为线路改造停电，所以只能依靠笔记本里可怜的剩余电池来写这篇文章，时间仓促，肯定有很多不足的地方，大家可以向我提出来，我一定认真回答改正，谢谢！