·电路与控制·

Windows 2000环境下的串口通信

刘 洋,孙 硕

(东北电子技术研究所,辽宁 锦州 121000)

摘 要:对windows 2000环境下串口通信的开发方法进行了阐述,对每种方法的编程实现进行了介绍,比较了各自的优缺点,并根据具体使用要求,选择了合适的开发方法。对串口通信特别是串口数据处理以及串口通信模块封装进行了研究。介绍了环形缓冲区数据处理方法和动态链接库(DLL)封装串口通信模块的方法,并通过Visual C++编写的代码实现进行了验证。

关键词:串口通信;环形缓冲区;动态链接库

中图分类号:TP311 **文献标识码**:A **文章编号**:1673-1255(2011)03-0065-06

Serial Port Communication in Windows 2000 Environment

LIU Yang, SUN Shuo

(*Northeast Research Institute of Electronics Technology*, *Jinzhou* 121000, *China*)

Abstract: In windows 2000 environment, the method of serial port communication development is summarized. Every method to program is introduced, and their advantages and disadvantages are compared. For application requirements, the appropriate development methods are selected. The serial port data processing and serial port module packaging are studied. The method of ring buffer data processing and the method that the serial port communication module is packaged using dynamic-link library (DLL) are introduced, and these methods are validated by the code of Visual C++.

Key words: serial port communication; ring buffer; dynamic-link library

随着计算机硬件的普及使用,串口通信广泛应用在数据采集、工业控制、系统仿真以及军用平台设备之间的数据交换等领域。计算机操作系统的发展,尤其windows 2000环境的成熟稳定,使基于windows 2000环境下的串口控制变得更加便捷。用户可以根据使用需求,选择成熟的控件或使用Windows API(application programming interface 应用编程接口)自定义串口操作。

1 Windows 2000环境下的串口通信

Windows 2000环境下,对串口通信的操作一般不直接对硬件控制,而是通过操作系统提供的设备驱动程序来进行数据传递,通常采用的开发方法有以下几种[1]:

(1) Windows API通信函数方法:虽然Windows 2000环境下可视化的操作界面更加方便直观,但是仍有很多人习惯直接使用32位API函数编写程序。Windows API函数以文件的方式对串口进行操作。适用于对串口操作有更详细的需求,可以全面掌握串口操作的每个环节的技术状态,但由于技术复杂,专业化程度高,使用较困难。

(2) MSComm控件[2]:使用Microsoft Visual系列开发语言进行程序开发时,MSComm控件是一个不错的选择。MSComm控件是微软开发的专用通信控件,封装了串口的所有功能,使用方便。实际应用时,只需要按使用要求对属性进行设置,即可

收稿日期:2011-04-27

作者简介:刘洋(1978-),男,辽宁锦州人,学士,工程师,研究方向为电子工程.

快速开发串口通信程序，缺点是控件带有资源，只能添加在有视窗或对话框的资源上进行使用。

（3）第三方串口通信类[3]：使用Microsoft Visual C++开发语言可以使用诸如：CSerial类、CSerialPort类等许多第三方免费串口通信类。CSerial类是MuMega Technologies公司提供的一个免费的第三方的VC++类；CSerialPort类是由Remon Spekreijse提供的免费串口类。它们的共同特点是：串口函数丰富，全面涵盖各种操作，可以方便地直接使用或根据需求进行定制开发实现串口通信。

（4）串口硬件设备开发包：很多情况下，选用的串口通信扩展卡都会附带软件开发包（接口函数），可以在此基础上，调用现有的函数，编写程序对串口进行操作。此种方法适用于简单串口控制，但对串口控制只局限在厂家提供的接口函数，无法按使用需求更深入开发。

实际使用中可以根据对串口通信控制的要求，结合对技术掌握的熟练程度，选择适合的方法。

2 串口通信处理研究

串口通信处理，一般包括以下步骤：（1）创建串口；（2）初始化配置串口；（3）串口接收（或发送）处理；（4）关闭串口。

选择任何开发方法时，所进行的基本操作都遵循以上步骤，但对于不同的方法，具体的操作又有很大的区别。

2.1 使用Windows API的开发

对于使用Windows API的开发，创建串口使用CreateFile函数实现，函数的声明如下[4]：

```
HANDLE CreateFile（LPCTSTR lpFileName，        //文件名指针
DWORD dwDesiredAccess，      //操作模式
DWORD dwShareMode，        //共享模式
LPSECURITY ATTRIBUTES lpSecurity Attributes，  //安全属性指针
DWORD  dwCteationDistribution，      //文件建立方式
DWORD  dwFlagsAndAttributes，        //文件属性
HANDLE hTemplateFile）              //模板文件名
```

其中，打开串口时，lpFileName为“COM1”、“COM2”等；dwCreationDistribution必须为OPEN_EXISTING；dwDesiredAccess为GENERIC_READ|GENERIC_WRITE（可读写操作）；使用异步读写时dwFlagsAndAttributes必须为FILE_FLAG_OVERLAPPED；其他参数为0（NULL）。函数返回串口的句柄。

对串口初始化设置使用SetCommState函数，函数的声明如下：

```
BOOL SetCommState（HANDLE hFile，     //串口句柄，通过CreateFile返回值得到
LPDCB lpDCB）；  //指向DCB（device-control block）结构
```

DCB结构存储串口的配置信息，可以通过GetCommState函数获得，并将DCB结构中参数如：DWORD BaudRate //波特率、BYTE ByteSize //数据位、BYTE Parity //奇偶校验等进行设置。

串口初始化设置后，就可以使用ReadFile函数进行接收（读）操作，使用WriteFile进行发送（写）操作。函数的声明如下[2]：

```
BOOL  ReadFile（HANDLE hFile，  //串口句柄，通过CreateFile返回值得到
LPCVOID lpBuffer，  //指向发送缓冲区
DWORD nNumberOfBytesToRead，  //指明要从串口读取的字节数
LPDWORD lpNumberOfBytesRead，  //指明实际从串口设备读取的字节数
LPOVERLAPPED  lpOverlapped）；  //OVERLAPPED结构
```

如果hFile为FILE_FLAG_OVERLAPPED时，lpOverlapped参数不能为NULL，而应该是一个有效的OVERLAPPED结构；不需要此结构时，设置为NULL。OVERLAPPED异步I/O重叠结构，包括了异步输入输出（I/O）操作的相关信息。对于串口读写，当同步执行时，函数直到操作完成后才返回，这意味着同步执行时其他处理将可能被阻塞，从而导致效率下降；在重叠执行时，即使操作还未完成，调用的函数也会立即返回，费时的I/O操作在后台进行，从而提高了效率。对于异步操作及OVERLAPPED结构，限于篇幅，不做赘述。

使用完毕后，要使用CloseHandle函数关闭串口。使用Windows API进行串口操作的基本函数如上，但实际使用时，还要建立串口通信事件CreateEvent以使在串口有数据时获知并处理，还要对缓冲区进行处理，对错误的判断处理等大量的操作。使用Windows API的优点就是数据的整个处理过

程都清清楚楚的由自己定制编写处理,缺点就是要熟悉大量的函数和参数,技术难度大,工作量大。

2.2　使用MSComm控件的开发

使用MScomm控件的不同于基本步骤的地方有[2]:(1)在工程中插入Microsoft Communications Control控件;(2)添加控件ID的控制变量(或对象);(3)初始化串口,设置MSComm控件的属性;(4)添加串口事件的消息处理函数OnComm,在函数中根据需要编写数据处理代码;(5)编写串口发送等其他代码;(6)关闭串口。

其中(1)相当于创建串口,不过当资源添加后,系统自动生成函数,不用人工参与,这也是控件区别于Windows API函数的地方。接着(2)、(3)就是初始化配置串口,(4)是接收处理,(5)是发送处理,这些只要调用控件的函数就可以很容易的实现。串口的消息处理函数OnComm只要是在MSComm的成员函数SetRThreshold(x)函数中设置变量x,对于接收缓冲区有x个及以上的字符时,自动引发OnComm事件,不用像Windows API那样,所有的事件都要自己定义实现。MSComm使用便利,对于简单的串口收发小程序开发非常方便,但它的控件必须插入在有资源(如:窗口、对话框等)的工程中,而且一个控件只能控制一个串口。

2.3　使用第三方串口通信类的开发

对于第三方串口通信类,如CSerialPort类,步骤如下[2]:(1)在工程中通过菜单Project->Add to Project->Files…,添加类文件SerialPort.h和SerialPort.cpp,并在工程头文件中声明对串口类的引用:# include "SerialPort.h";(2)使用InitPort函数设置串口的通信参数,初始化串口;(3)启动串口通信监控线程函数StartMonitoring,定义串口通信监控线程;(4)添加消息响应函数声明,定义消息响应函数,并对接收到的数据进行处理;(5)定义串口发送函数WriteToPort和其他功能函数;(6)使用ClosePort关闭串口函数,关闭串口,释放串口资源。

CSerialPort类是基于多线程的,不依靠工程资源,使用方便,而且提供了一些API函数,方便根据使用要求进行扩展定制。常用函数和参数简明易懂,便于使用。CSerialPort类只支持线连接(非MODEM)的串口编程操作,而MSComm支持对MODEM的操作,所以在对MODEM的串口编程操作时,不能使用CSerialPort类。

2.4　使用串口硬件设备开发包的开发

在购买了串口硬件产品的时候,可以使用随产品配备的硬件设备开发包。开发包使用时,需要在工程头文件中添加对开发包的调用声明,然后要具体使用要求,定义参数调用函数,实现串口通信控制。相对来说,优点是单一简单,缺点是受制于有限的开发函数,不能进行定制开发其他功能。

3　控制系统中串口通信软件的开发实践

3.1　串口数据处理

在工业控制等实际使用环境中,通常要求单台计算机同时监控多个外部设备状态,对多个串口数据单独进行采集处理,并在时间精度要求范围内,向对应的设备返回控制命令;通讯的技术协议要求串口通信数据包括:标志头(或标志尾)、透明符等数据,串口的数据长度为定长和不定长的情况等。

其特点为协议单一固定,串口控制功能需求相对简单,具体开发需求如下:

(1)不需要通过编写繁琐的Windows API函数进行处理;

(2)没有工程资源而且串口数目多不便于MSComm控件的添加使用;

(3)从模块化的角度出发,考虑串口程序可以多次复用。

因此在该类应用中,采用Microsoft Visual C++ 6.0开发软件,设计了多个独立的分设备串口通信处理模块(简称:分设备模块)。分设备模块使用DLL封装独立的通用串口控制函数类和专用的分设备数据处理类,其中通用串口控制函数类通过添加CSerialPort类函数,实现了创建串口、配置打开串口、发送数据以及关闭串口等功能,专用的分设备数据处理类将接收和发送的数据按协议要求进行进一步处理。分设备模块通过发送消息和接口函数与主程序进行交互,实现串口的控制功能。关系组成见图1所示。

采用分设备模块的思想,可以使串口处理模块独立于主程序进行测试使用,同时具有可复用的特点,

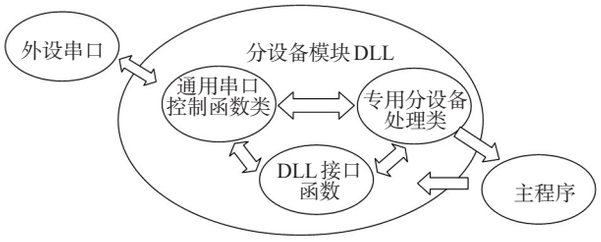

图1 分设备模块关系组成框图

在设计开发相同的外部设备时,直接进行使用或在此基础上根据不同的接口协议进行简单的修改即可使用。分设备模块一方面提高了代码的继承性,减小了资源的浪费;另一方面,不断的测试使用也完善了分设备模块的功能,提高了软件的质量。

串口通信程序,一般由三部分构成:前端人机交互部分、中间数据处理部分及后台串口操作部分。要满足多串口处理实时性及数据量要求,多线程并行处理是很好的选择。多线程的串口通信程序应包括主线程和辅助线程两部分。主线程用来进行人机交互的操作和协调辅助线程运行;辅助线程主要是每个设备的串口监控线程,负责实时监视串口状态,并对串口数据进行处理,对于预定事件向主线程发送消息,请求主线程处理。文中介绍的串口通信程序在分设备模块的串口接收处理部分应用多线程机制,建立独立的串口监控处理线程,通过添加CSerialPort类的基本函数,同时对多个外部设备的串口进行控制,对串口数据按照协议进行提取,然后传递给分类处理,并通过消息函数PostThreadMessage通知主线程(或主程序),分类处理的数据通过消息函数的参数直接(或通过指针参数)传递。分设备模块串口接收处理的流程如图2所示。

串口接收处理中主要部分为提取数据,也就是环形缓冲区处理。环形缓冲区区别于串口缓冲区,为监控线程单独创建的临时缓冲区,每次使用时将串口缓冲区的数据全部拷贝过来。环形缓冲区的特点是,数据顺序写入,当超过缓冲区长度时,数据继续从缓冲区起点写入,并覆盖原有数据。

环形缓冲区使用读、写2个事先定义的指针变量及每次拷贝数据时创建的静态临时读指针变量。初始化时,所有变量为初值0。当从串口缓冲区拷贝数据到环形缓冲区时,将写指针变量增加到实际数据长度,读指针变量赋值给临时读指针变量。进行数据处理时,根据协议规定的透明符、起始符(或结束符),将临时读指针变量指向的数据与

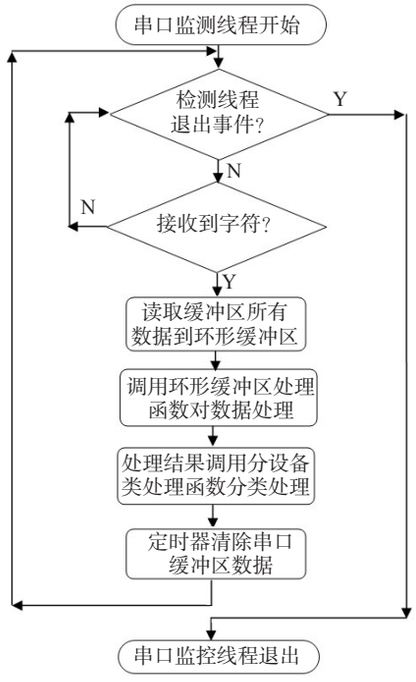

图2 分设备模块串口接收处理的流程

透明符等特殊数据进行比较,查看数据中是否有满足要求的完整的数据帧,依次,临时读指针不断增加,当临时读指针数据超过环形缓冲区长度时进行取余处理,从头开始循环,直至发现一帧完整数据。如果发现完整的一帧数据,则跳出比较,进行数据处理。数据处理时,根据临时读指针的位置,读指针不断追赶临时读指针,并将指向的数据进行协议比较(是否为透明符等特殊数据)后,数据拷贝到处理后数据缓冲区,直至读指针位置等于临时读指针位置,数据提取完毕。串口缓冲区提取数据的同时,根据外部设备使用要求,还可以设置定时器,在数据提取完毕后对串口缓冲区进行清零操作,减少缓冲区误码数据的影响,提高串口数据处理速度。线程里的定时器可以使用timeSetEvent函数定义CALLBACK函数实现。

环形缓冲区数据处理程序如下(此程序针对有透明符和结束符的情况):

```
BOOL RevDateProcess::Pick_Package()
{
 static int Frame_len;//定义得到的数据帧长度
 static int tReadPoint;//定义临时读指针变量
 static BOOL find=FALSE;//定义找到一帧数据标志
 static unsigned char TempBuffer[MAXBUFFERLEN];//
定义临时缓冲区大小
 tReadPoint=ReadPoint;//读指针等于临时读指针
```

```
do{
while(tReadPoint!=WritePoint)
{
if(m_pWorkBuf[tReadPoint]= =TRANSFLAG)//临时读指针指向数据为透明符的情况
if((tReadPoint+1)%MAXBUFFERLEN= =WritePoint)//比较下一帧数据是否是结尾
tReadPoint=(tReadPoint+1)%MAXBUFFERLEN;
else
tReadPoint=(tReadPoint+2)%MAXBUFFERLEN;
else if(m_pWorkBuf[tReadPoint]= =ENDFLAG) //临时读指针指向数据为结束符的情况
{
if((tReadPoint+1)%MAXBUFFERLEN!=WritePoint)//结束符下一帧数据是否是结尾
{
tReadPoint=(tReadPoint+1)%MAXBUFFERLEN;//继续循环
find=TRUE;//找到一帧完整数据
break;
}
else
tReadPoint=(tReadPoint+1)%MAXBUFFERLEN;//继续循环
}
else tReadPoint=(tReadPoint+1)%MAXBUFFERLEN;
}//找到一帧完整的数据或已结束或断帧
Frame_len=0;
if(find)
{
while(ReadPoint!=tReadPoint)
{
……//同读环形缓冲区的过程,进行取数据操作
}//end while
if(Frame_len>0)
{
memcpy(m_AfterProcBuf,TempBuffer,Frame_len);
Process_Package(Frame_len);//进行数据分类处理
}
find=FALSE;
}
}while(tReadPoint!=WritePoint);
return FALSE;
}
```

以上是针对有透明符和结束符情况的代码实现，对于其他协议要求的串口数据，可以根据具体情况，在条件判断语句的位置进行相应更改，其串口环形缓冲区的处理方法原理是一致的。

数据提取完毕后，通过调用分设备模块里的分设备数据处理类进一步处理，将处理后的结果通过PostThreadMessage函数通知主线程直接应用于显示或其他处理。这样的处理可以将分设备数据由各自的线程单独处理，避免了多个分设备数据同时由主线程处理时的资源抢占。

3.2　动态链接库(DLL)封装串口分设备模块

分设备模块要想独立使用，就需要对其进行封装。动态链接库(DLL，Dynamic-Link Library)封装是一种很好的方法。

DLL也是一种可执行文件，它不能像普通的EXE文件直接运行，也不接收消息。他是一些独立的文件，其中包含能被应用程序或其他DLL调用完成一定作业的函数。只有在其他模块呼叫动态链接库中的函数时，它才发挥作用。

动态链接库作为软件系统的重要组成元素，为通用代码的重用提供了基础，也为多人协作开发软件提供了可能。使用动态链接库具有以下优点[5]：

(1) 实现数据、代码的共享，节省系统空间。使用DLL提供了一种共享数据、代码和资源(图标、位图、字符串和对话框等)的方便途径。并且，由于多个应用程序可以共享同一个DLL中的函数，因此，使用DLL可以显著节省磁盘空间。另外，多个应用程序还可以同时共享DLL在内存中的同一份拷贝，这样就有效地节省了应用程序所占用的内存资源，减少了频繁的内存交换，提高了应用程序的执行效率；

(2) 便于程序的升级处理。由于DLL是独立于可执行文件的，因此，如果需要向DLL中增加新的函数或是增强现有函数的功能，只要原有函数的参数和返回值等属性不变，那么，所有使用该DLL的原有应用程序都可以在升级后的DLL的支持下运行，而不需要重新编译。这就极大地方便了应用程序的升级和售后支持。

(3) 便于建立多语言的应用程序。可以把多语言应用程序中所使用的与语言相关的函数做到DLL中，只要不同语言的应用程序所调用的函数都具有相同的接口，就可以通过简单地更换DLL来实现多语言支持。

使用DLL也有不足，最典型的是应用程序在运行时必须有相应的DLL文件支持。另外，使用DLL也增大了程序运行的开销。

综合DLL的优缺点，考虑到程序的升级处理和具体使用要求，采用DLL的方法对分设备模块进行封装。每个分设备模块都封装一个独立的DLL，包括三部分：(1)在DLL中定义主程序调用的接口函数，包括：线程启动函数、线程终止函数、控制命令函数和发送数据函数等；(2)DLL包括独立的通用串口控制函数类，对设备串口数据进行基本的读写和按照协议提取处理；(3)包括专用的分设备数据处理类，对数据进一步处理成满足显示和使用的数据，并发送消息，传递给主程序直接使用。

动态链接库的连接调用。在创建DLL时，使用模块定义(.DEF)文件的方法导出DLL函数。语句如下：

```
; PjDll.def : Declares the module parameters for the DLL.
LIBRARY      "PjDll"
DESCRIPTION  'PjDll Windows Dynamic Link Library'
EXPORTS
  ; Explicit exports can go here
    Load_PjDll
  PjBomQuCmd
  PjStartCmd
  Unload_PjDll
  PjCheckCmd
```

链接主程序到DLL有2种方法：隐式链接和显式链接。隐式链接又称静态加载。使用隐式链接的应用程序，操作系统会在加载应用程序的同时加载应用程序所使用的DLL。显示链接又称为动态加载。使用动态加载的应用程序必须在代码中明确加载所使用的DLL，并使用指针调用DLL中的导出函数，在使用完毕之后，应用程序必须卸载所使用的DLL。因为串口监控要求程序在启动后即开始工作，所以链接选择采用隐式链接的方式。在头文件进行声明：

```
#pragma comment(lib,"PjDll.lib")
_declspec(dllimport)PjCheckCmd(BYTE DeviceName);
_declspec (dllimport) Load_PjDll (CWnd* pParent,
DWORD mThread, sSerialInterface* pSerialPar );
……
```

在应用程序启动时，调用Load_PjDll函数启动DLL中的串口监控线程，开始工作。

在DLL中数据处理完毕后，通过PostThreadMessage函数通知主线程，传递数据。此时应用程序中的要有一个专门的数据处理线程，接收DLL的消息，进行处理。消息定义如下：

```
头文件.h
 afx_msg LRESULT OnPjData (WPARAM wParam,
LPARAM lParam);
源文件.cpp
 ON_THREAD_MESSAGE (PJ_DATA_ARRIVE, OnPj-
Data)
LRESULT CMsgProcess::OnPjData(WPARAM wParam,
LPARAM lParam)
{
 C**App  *pApp =(C**App*)AfxGetApp();
 while(pApp->m_pMainWnd==NULL);
 ::PostMessage (pApp->m_pFormView->m_hWnd,
PJ_DATA_UPDATE, wParam, lParam);
 return 0;
}
```

通过以上设置，接收DLL中的数据，并传递给视窗类用于显示等处理。

应用程序退出前，调用卸载函数清空串口，释放资源，结束监控线程。

4 结束语

通过对Windows 2000特定环境下，串口通信的深入研究，给出了环形缓冲区处理串口数据的方法，并考虑软件重用的思想，采用了DLL的形式封装了串口通信处理模块。文中的研究成果已应用于某工程系统中，串口通信数据稳定，满足实时性要求。多个设备在DLL分设备模块的基础上根据不同协议进行简单修改即可完成对各自设备的串口数据监控处理，提高了开发的效率和程序的可靠性。

参考文献

[1] 龙勇，袁静，黄先祥. 基于VC6的分布式仿真系统多线程串行通信的实现[J]. 计算机工程与应用，2006(6)：141-144.

[2] Charles Petzold. Windows程序设计[M]. 5版. 北京：北京大学出版社，1999:1-9.

[3] 龚建伟，熊光明. Visual C++/Turbo C 串口通信编程实践[M]. 2版. 北京：电子工业出版社，2007：27-130.

[4] 范文庆，周彬彬，安靖. 精通Windows API 函数、接口、编程实例[M]. 北京：人民邮政出版社，2009:20-35.

[5] Jeffrey Richter. Windows 核心编程[M]. 北京：机械工业出版社，2008:325-363.