


原创 Flywithdawn 于 2020-09-11 15:29:47 发布 3813 收藏 29 版权

分类专栏： 串口通信 文章标签： windows 串口通信 api

 串口通信 专栏收录该内容

0 订阅 3 篇文章 订阅专栏

Windows API 串口编程详解

文章目录

- (一) Windows API串口通信编程概述
- (二) Windows串口通信相关API函数
- 2.1打开和关闭串口
- 1.打开串口
- CreateFile () 函数声明如下：
- 2.关闭串口
- CloseHandle函数声明如下：
- 2.2串口配置和串口属性
- 1.串口配置
- GetCommState () 函数声明如下：
- BuildCommDCB函数， 函数声明如下：
- 2.缓冲区控制
- PurgeComm () 函数的声明如下：
- 2.3读写串口
- 1.读串口操作
- ReadFile()函数声明如下：
- 2.写串口操作
- WriteFile()函数声明如下：
- 3. 异步I/O操作
- GetOverlappedResult()函数可声明为：
- 4.超时设置
- 超时结构定义如下：
- GetCommTimeouts()函数。该函数声明如下：
- SetCommTimeouts()函数。该函数声明如下：
- 5.通信状态和通信错误
- ClearCommError()函数的声明如下：
- COMSTAT结构声明如下：
- 2.4通信事件
- 1.通信事件
- 2.操作通信事件
- SetCommMask函数的声明如下：
- etCommMask()函数声明如下：
- 3.监听通信事件
- WaitCommEvent()函数声明如下：

- (一) Windows API 串口通信 编程概述
- 1.打开 串口
- 2.配置串口
- 3.读写串口
- 4.关闭串口

Windows环境下的串口编程。Windows封装了Windows的通信机制，这种方式称为通信API，Windows程序可以利用Windows通信API进行

分类专栏

| | | |
|---|-------------|-----|
|  | QT | 12篇 |
|  | C++ | 7篇 |
|  | 计算机网络 | 1篇 |
|  | 进程通信 | 1篇 |
|  | 前端 | 2篇 |
|  | 框架 | 1篇 |
|  | 读个小故事就弄懂 | 1篇 |
|  | 测试 | 1篇 |
|  | git | 1篇 |
|  | C++Primer入门 | |
|  | 驱动 | 9篇 |
|  | 串口通信 | 3篇 |

编程，不用对硬件直接进行操作。这种体系被称为Windows开放式服务体系。不需要许多协议层的交互式、非实时的通信可以采用Win32通信API来实现。Win32通信API把串口操作（以及并口等）和文件操作统一起来了，使用类似的操作来实现。

（二）Windows串口通信相关API函数

2.1打开和关闭串口

1.打开串口

在32位的Windows系统中，串口和其它设备通信时作为文件处理的。串口的打开、关闭、读取和写入所用的函数与操作文件的函数完全一致。

通信回话以调用CreateFile（）开始。CreatFile（）为读访问、写访问或读写访问“打开”串口。

CreateFile（）[函数声明](#)如下：

```
HANDLE CreateFile
(
    LPCTSTR lpszName, // 串口名
    DWORD fdwAccess, // 指定串口访问类型 读/写
    DWORD fdwShareMode, // 设置共享属性
    LPSECURITY_ATTRIBUTES lpsa, // 设置安全性结构
    DWORD fdwCreate, // 指定如果CreatFile（）正在被已有的文件调用时应采取的动作。因为串口总是存在，fdwCreate必须设置成
    OPEN_EXISTING。
    DWORD fdwAttrsAndFlags, // 描述了端口的各种属性
    HANDLE hTemplateFile // 指向模板的句柄
)
```

lpszName：指定要打开的串口逻辑名，用字符串表示，如“COM1”和“COM2”分别表示串口1和串口2。

fdwAccess:用来指定串口访问的类型。与文件一样，串口也是可以被打开以供读取、写入或者两者皆有。

GENERIC_READ位读取访问打开端口，GENERIC_WRITE位写访问打开端口。

这两个常数定义如：const GENERIC_READ=0x80000000h;

const GENERIC_WRITE=0x40000000h;

用户可以用逻辑操作将这两个标识符连接起来，为读/写访问权限打开端口。因为大部分串口通信都是双向的，因此常常在设置中将这两个标识符连接起来使用。如：

fdwAccess=GENERIC_READ|GENERIC_WRITE;

fdwShareMode

指定该端口的共享属性。该参数是为那些由许多应用程序共享的文件提供的。对于不能共享的串口，它必须设置为0.这就是文件与通信设备之间的主要差异之一。如果当前程序调用了CreateFile（），另一个应用程序已经打开了串口，该函数就会返回错误代码。因为两个应用程序不能共享一个端口。然而，同一个应用程序的多个线程可以共享由CreateFile（）返回的端口句柄，并且根据安全属性设置，该句柄可以被打开端口的应用程序的子程序所继承。

lpsa

引用安全性属性结构（SECURITY_ATTRIBUTES），该结构定义了一些属性，例如通信句柄如何被打开端口的应用程序的子程序所继承。将该参数设置为NULL将为该端口分配缺省的安全性属性。子应用程序所继承的缺省属性是该端口不能被继承的。

安全属性结构声明如下：

```
typedef struct _SECURITY_ATTRIBUTES
{
    DWORD nLength;
    LPVOID lpSecurityDescriptor;
    BOOL bInheritHandle;
} SECURITY_ATTRIBUTES;
```

SECURITY_ATTRIBUTES结构成员nLength指明该结构的长度，lpSecurityDescriptor指向一个安全描述字符，bInheritHandle表明句柄是否能被继承。

fdwCreate

指定如果CreatFile（）正在被已有的文件调用时应采取的动作。因为串口总是存在，fdwCreate必须设置成OPEN_EXISTING。该标志告诉Windows不用企图创建新端口，而是打开已经存在的端口。

OPEN_EXISTING常数被定义为：

const OPEN_EXISTING =3;

fdwAttrsAndFlags

描述了端口的各种属性。对于文件来说，有可能具有很多属性，但对于串口，唯一有意义的设置是FILE_FLAG_OVERLAPPED。当创建时指定该设置，端口IO可以在后台进行（后台IO也叫异步IO）。FILE_FLAG_OVERLAPPED常数定义如下：

```
const FILE_FLAG_OVERLAPPED=0x40000000h
```

hTemplateFile

指向模板的句柄，当端口处于打开状态时，不使用改参数，因而必须设置成0。

实例：

调用CreatFile函数打开COM1串口操作的例子：

```
HANDLE hCom;
DWORD dwError;
hCom=CreateFile
(
    "COM1", // 文件名
    GENERIC_READ | GENERIC_WRITE, // 允许读和写
    0, // 独占方式
    NULL,
    OPEN_EXISTING, // 打开而不是创建
    FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, // 重叠方式
    NULL
);

if(hCom == INVALID_HANDLE_VALUE)
{
    dwError=GetLastError(); // 处理错误
}
```

一旦串口处于打开状态，就可以分配一个发送缓冲区和接收缓冲区，并且通过调用SetupComm（）实现其它初始化工作。也可以不调用SetupComm（）函数，Windows系统也会分配缺省的发送和接收缓冲区，并且初始化端口。但为了保证缓冲区的大小与实际需要的一致，最好还是调用该函数。SetupComm（）函数声明如下：

```
BOOL SetupComm
(
    HANDLE hFile, // 通信设备句柄
    DWORD dwInQueue, // 输入缓冲区大小
    DWORD dwOutQueue // 输出缓冲区大小
);
```

SetupComm（）函数中各项含义说明如下：

hFile

由CreatFile()返回的指向已打开端口的句柄。

dwInQueue和dwOutQueue

接收缓冲区的大小和发送缓冲区的大小。这两个定义并非实际的缓冲区的大小，指定的大小仅仅是“推荐的”大小，而Windows可以分配任意大小的缓冲区。Windows设备驱动程序可以获得这两个数据，并不直接分配大小，而使用来优化性能和避免缓冲区超限。

注意：当使用CreatFile（）函数打开串口时：为实现调制解调器的排他性访问，共享标识必须设为零；创建标识必须设为

OPEN_EXISTING;模板句柄必须置为空。

2.关闭串口

关闭串口比打开串口简单得多，只需要调用CloseHandle（）函数关闭由CreatHandle（）函数返回得句柄即可。

CloseHandle函数声明如下：

```
BOOL CloseHandle
(
    HANDLE hObject // 需关闭的设备句柄
);
```

使用串口时一般要关闭它，如果忘记关闭串口，串口就会始终处于打开状态，其它应用程序就不能打开并使用串口了。

2.2串口配置和串口属性

CreatFile () 打开串口后，系统将根据上次打开串口时设置的值来初始化串口，可以集成上次打开操作后的数值，包括设备控制块（DCB）和超时控制结构（COMMTIMEOUTS）。如果是首次打开串口，Windows操作系统就会使用缺省的配置。

1.串口配置

使用GetCommState () 获取串口得当前配置，使用SetCommState () 重新分配串口资源的各个参数。

GetCommState () 函数声明如下：

```
BOOL GetCommState
(
    HANDLE hFile, // 通信设备句柄
    LPDCB lpDCB // 指向device-control block structure的指针
);
```

参数说明：

hFile

由CreatFile () 函数返回的指向已打开串口的句柄。

lpDCB

一个非常重要的结构——设备控制块DCB（Device Control Block）

DCB结构的主要参数说明如下：

·DCBLength: 一字节为单位指定的DCB结构的大小。

·Baudrate: 用于指定串口设备通信的数据传输速率，它可以是实际的数据传输速率数值，也可以是下列数据之一： CBR_110, CBR_19200, CBR_300, CBR_38400, CBR_600, CBR_56000, CBR_1200, CBR_57600, CBR_2400, CBR_115200, CBR_4800, CBR_12800, CBR_9600, CBR_25600, CBR_14400。

·fBinary: 指定是否允许二进制。Win32API不支持非二进制传输，因此这个参数必须设置为TRUE，如果设置为FALSE则不能正常工作。

·fParity: 指定是否允许奇偶校验，如果这个参数设置为TRUE，则执行奇偶校验并报告错误信息。

·fOutxCtsFlow: 指定CTS是否用于检测发送流控制。当该成员为TRUE，而CTS为OFF时，发送将被挂起，直到CTS置ON。

·fOutxDsrFlow: 指定DSR是否用于检测发送流控制，当该成员为TRUE，而DSR为OFF时，发送将被挂起，直到DSR置ON。

·fDtrControl: 指定DTR流量控制，可以是表1中的任一值。

·fDsrSensitivity: 指定通信驱动程序对DTR信号线是否敏感，如果该位置设为TRUE时，DSR信号为OFF，接收的任何字节将被忽略。

·fTXContinueOnXoff: 指定当接收缓冲区已满，并且驱动程序已经发送出XoffChar字符时发送是否停止。当该成员为TRUE时，在接收缓冲区内接收到了缓冲区已满的字节XoffLim，并且驱动程序已经发送出XoffChar字符终止接收字节之后，发送继续进行。该成员为FALSE时，接收缓冲区接收到代表缓冲区已空的字节XonLim，并且驱动程序已经发送出恢复发送的XonChar字符后，发送可以继续进行。

·fOutX: 该成员为TRUE时，接收到XoffChar之后停止发送，接收到XonChar之后发送将重新开始。

·fInX: 该成员为TRUE时，接收缓冲区内接收到代表缓冲区满的字节XoffLim之后，XoffChar发送出去，接收缓冲区接收到代表缓冲区已空的字节XonLim之后，XonChar发送出去。

·fErrorChar: 当该成员为TRUE，并且fParity为TRUE时，就会用ErrorChar成员指定的字符来代替奇偶校验错误的接收字符。

·fNull: 指明是否丢弃接收到的NULL(ASCII 0)字符，该成员为TRUE时，接收时去掉空（零值）字节；反之则不丢弃。

·fRtsControl: 指定 RTS 流量控制，可以取表2中的值。0值和DTR_CONTROL_HANDSHAKE等价。

·fAbortOnError: 如果发送错误，指定是否可以终止读、写操作。如果该位为TRUE，当发生错误时，驱动程序以出错状态终止所有的读写操作。只有当应用程序调用ClearCommError()函数处理后，串口才能接收随后的通信操作。

·fDummy2: 保留的位，没有使用。

·wReserved: 没有使用，必须为零。

·XonLim: 指定在XOFF字符发送之前接收到缓冲区中可允许的最小字节数。

·XoffLim: 指定在XOFF字符发送之前缓冲区中可允许的最小可用字节数

·ByteSize: 指定端口当前使用的数据位数。

·Parity: 指定端口当前使用的奇偶校验方法。它的可能值如表3所示。

·StopBits: 指定串口当前使用的停止位数，可能值如表4所示

·XonChar: 指明发送和接收的XON字符值，它表明允许继续传输。

·XoffChar: 指明发送和接收的XOFF字符值，它表示暂停数据传输。

·ErrorChar: 本字符用来代替接收到的奇偶校验发生错误的字符。

·EofChar: 用来表示数据的结束。

·EvtChar: 事件字符。当接收到此字符的时候, 会产生一个事件。

·wReserved1: 保留的位, 没有使用。

如果GetLastError () 函数也是Win32API函数, 它的声明如下:

DWORD GetLastError (VOID) ;

如果应用程序只需要修改一部分配置的时候, 可以通过GetCommState () 函数获得当前的DCB结构, 然后更改DCB结构中的参数, 调用SetCommState () 函数配置修改过的DCB来配置端口。SetCommState () 函数声明如下:

```
BOOL SetCommState  
(  
HANDLE hFile, // 已打开的串口的句柄  
LPDCB lpDCB // 指向DCB结构的指针  
);
```

如果函数调用成功, 则返回值不为零; 若函数调用失败, 则返回值为零。出错时可以调用GetLastError()函数获得进一步的出错信息。

SetCommState()函数调用的DCB结构中的XonChar等价于XoffChar成员, 则SetCommState()函数会调用失败。

DCB最经常改变的参数是数据传输速率、奇偶校验的方法以及数据位和停止位数。Windows为改变这些设置提供了

BuildCommDCB函数, 函数声明如下:

```
BOOL BuildCommDCB  
(  
LPCTSTR lpDef, // 设置的字符串  
LPDCB lpDCB // 指向DCB结构的指针  
);
```

BuildCommDCB()参数包含新设置的字符串和一个DCB结构的参数, 该设置将提供给DCB结构。

baud=1200 parity=N data=8 stop=1

这条语句将数据传输速率设置为1200bits/s, 关闭奇偶校验, 数据位数设为8, 停止位数设为1。与在DOS或Windows NT/2000系统中一样, 该字符串不包括串口的名称, 实际上这个函数并不改变端口的设置, 因此没有必要标识该串口, 当然这个串口必须是有效的串口。新的设置只是简单地拷贝到已提供好的DCB结构中, 要使新设置生效, 还必须调用SetCommState()函数。BuildCommDCB()支持老的和新的各种版本的Mode命令, 缺省情况下, BuildCommDCB()函数禁止XON/XOFF和硬件流的控制。如果使用硬件流控制, 则必须设置DCB结构的各个成员的值。如果这个函数调用成功, 则返回值不为零。如果想得到进一步的错误信息, 可以调用GetLastError()函数来获取。

2.缓冲区控制

Win32通信API除了提供SetuoComm () 函数实现初始化的缓冲区控制外, 还提供了PurgeComm () 函数和FlushFileBuffers () 函数来进行缓冲区操作。

PurgeComm () 函数的声明如下:

```
BOOL PurgeComm  
(  
HANDLE hFile, // 返回的句柄  
DWORD dwFlags // 执行的动作  
);
```

参数hFile指向由CreateFile函数返回的句柄, 可以调用GetLastError()函数获得进一步的错误信息。dwFlags表示执行的动作, 这个参数可以是表5中的任一个。

由上面的叙述可以看出, PurgeComm()函数可以在读写操作的同时, 清空缓冲区。当应用程序在读写操作时调用PurgeComm()函数, 不能保证缓冲区内的所有字符都被发送。如果要保证缓冲区的所有字符都被发送, 应该调用FlushFileBuffer()函数。该函数只受流量控制的支配, 不受超时控制的支配, 它在所有的写操作完成后才返回。

FlushFileBuffers()的函数声明如下:

```
BOOL FlushFileBuffers  
(  
HANDLE hFile // 函数打开的句柄  
);
```

参数hFile指向由CreateFile函数打开的句柄, 如果该函数调用成功, 则返回值不为零; 若函数调用失败, 则返回值为零。出错时可以调用GetLastError()函数获得进一步的出错信息

2.3读写串口

利用Win32通信API读写串口时，既可以同步执行，也可以重叠（异步）执行。在同步执行时，函数直到操作完成后才返回。这意味着在同步执行时线程会被阻塞，从而导致效率降低。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，这样线程就可以做其它工作。例如，线程可以在不同的句柄上同时执行I/O操作，甚至可以在同一句柄上同时进行读写操作。“重叠”一词的含义就在于此。

1.读串口操作

程序可以使用Win32API ReadFile()函数或者ReadFileEx()函数从串口中读取数据。ReadFile()函数对同步或异步操作都支持，而

ReadFileEx()只支持异步操作。这两个函数都受到函数是否异步操作、超时操作等有关参数的影响和限定。

ReadFile()函数声明如下：

```
BOOL ReadFile(
    (
        HANDLE hFile, // 指向标识的句柄
        LPVOID lpBuffer, // 指向一个缓冲区
        DWORD nNumberOfBytesToRead, // 读取的字节数
        LPDWORD lpNumberOfBytesRead, // 指向调用该函数读出的字节数
        LPOVERLAPPED lpOverlapped // 一个OVERLAPPED的结构
    );
```

其中主要参数介绍如下：

hFile：指向标识的句柄。对串口来说，就是由CreateFile函数返回的句柄。该句柄必须拥有GENERIC_READ的权限。

lpBuffer：指向一个缓冲区，该缓冲区主要用来存放从串口设备中读取的数据。

nNumberOfBytesToRead：指定要从串口设备读取的字节数。

lpNumberOfBytesRead：指向调用该函数读出的字节数。ReadFile()在读操作前，首先将其设置为0。Windows NT/2000中当lpOverlapped

没有设置时，lpNumberOfBytesRead必须设置。当lpOverlapped设置时，lpNumberOfBytesRead可以不设置。这是可以调用

GetOverlappedResult()函数获取实际的读取数值。Windows 9x中这个参数一定要设置。

lpOverlapped：是一个OVERLAPPED的结构，该结构将在后面介绍。如果hFile以FILE_FLAG_OVERLAPPED方式常见，则需要此结构；否则，不需要此结构。

需要注意的是如果该函数因为超时而返回，那么返回值是TRUE。参数lpOverlapped 在操作时应该指向一个OVERLAPPED的结构，如果该参数为NULL，那么函数将进行同步操作，而不管句柄是否是由 FILE_FLAG_OVERLAPPED 标志建立的。当ReadFile返回FALSE时，不一定就是操作失败，线程应该调用GetLastError函数分析返回的结果。例如，在重叠操作时如果操作还未完成函数返回，那么函数就返回FALSE，而且GetLastError函数返回ERROR_IO_PENDING。

2.写串口操作

可以使用Win32API函数WriteFile() 或者WriteFileEx()向串口中写数据。WriteFile()函数对同步或异步操作都支持，而WriteFileEx()只支持异步操作。这两个函数都受到函数是否异步操作、超时操作等有关参数的影响和限定。

WriteFile()函数声明如下：

```
BOOL WriteFile(
    (
        HANDLE hFile, // 指向标识的句柄
        LPCVOID lpBuffer, // 指向一个缓冲区
        DWORD nNumberOfBytesToWrite, // 指定要向串口设备写入的字节数
        LPDWORD lpNumberOfBytesWritten, // 指向调用该函数已写入的字节数
        LPOVERLAPPED lpOverlapped // 一个OVERLAPPED的结构
    );
```

hFile：指向标识的句柄。对串口来说，就是由CreateFile函数返回的句柄。该句柄必须拥有GENERIC_WRITE的权限。

lpBuffer：指向一个缓冲区，该缓冲区主要用来存放待写入串口设备的数据。

nNumberOfBytesToWrite：指定要向串口设备写入的字节数。

lpNumberOfBytesWritten：指向调用该函数已写入的字节数。WriteFile()在写操作前，首先将其设置为0。

lpOverlapped：是一个OVERLAPPED的结构。如果hFile以FILE_FLAG_OVERLAPPED方式常见，则需要此结构；否则，不需要此结构。

如果函数调用成功，则返回值不为零；若函数调用失败，则返回值为零。调用GetLastError()函数可以获得进一步的出错信息。

3. 异步I/O操作

异步（重叠）I/O操作是指应用程序可以在后台读或者写数据，而在前台做其它事情，例如，用程序可以在开始时对10000个数据进行读或写操作，然后返回执行其它的操作：在读写完成后，Windows就会产生一个信号，应用程序得到这个信号，便可以进行其它的读写操作。

要使用OVERLAPPED的结构， CreateFile()函数的dwFlagsAndAttributes参数必须设为FILE_FLAG_OVERLAPPED标识，读写串口函数必须指定OVERLAPPED结构。异步IO操作在Windows中使用广泛。

OVERLAPPED结构类型声明如下：

```
typedef struct _OVERLAPPED
{ // 0
    DWORD Internal;
    DWORD InternalHigh;
    DWORD Offset;
    DWORD OffsetHigh;
    HANDLE hEvent;
} OVERLAPPED;
```

Internal：操作系统保留，指出一个和系统相关的状态。当GetOverlappedResult()函数返回时，如果将扩展信息设置为ERROR_IO_PENDING，该参数有效。

InternalHigh：操作系统保留，指出发送或接收的数据长度，当GetOverlappedResult()函数返回值不为0时，该参数有效。

Offset和OffsetHigh：指明文件传送的开始位置和字节偏移量的高位字。当进行端口操作时，这两个参数被忽略。

hEvent：指定一个IO操作完成后触发的事件（信号）。在调用读写函数进行IO操作之前，必须设置该参数。

在设置了异步IO操作后，IO操作和函数返回有以下两种情况：

1 函数返回时IO操作已经完成：此时结果好像是同步执行的，但实际上这是异步执行的结果。

2 函数返回时IO操作还没完成：此时一方面，函数返回值为零，并且GetLastError()函数返回ERROR_IO_PENDING；另一方面，系统把OVERLAPPED中的信号事件设为无信号状态。当IO操作完成时，系统要把它设置为信号状态。

异步IO操作可以由GetOverlappedResult()函数来获取结果，也可以使用Windows信号函数来处理。

GetOverlappedResult()函数可声明为：

```
BOOL GetOverlappedResult
(
    HANDLE hFile,
    LPOVERLAPPED lpOverlapped,
    LPDWORD lpNumberOfBytesTransferred,
    BOOL bWait
);
```

hFile：标识通信句柄，它应该和开始调用重叠结构的ReadFile、WriteFile、WaitCommEvent函数的参数一致。

lpOverlapped：在启动异步操作时指定的OVERLAPPED结构。

lpNumberOfBytesTransferred：指向一个长整型变量，该变量接收有一个读或写操作实际传递的字节数

bWait：指定函数是否等待挂起的异步操作完成。如果该参数设为1，则该函数知道操作完成后才返回。如果该参数被设为0，同时处于被挂起状态，则该函数返回为0，并且GetLastError函数返回ERROR_IO_INCOMPLETE。

如果该函数调用成功，则返回值不为零；若函数调用失败，则返回值为零。调用GetLastError()函数可以获得进一步的出错信息。

Windows也使用等待函数来检查事件对象的当前状态或等待Windows状态信号，在WaitForSingleObject()函数，WaitForSingleObjectEx()函数，以及WaitForMultipleObject()，WaitForMultipleObjectsEx()函数中指定OVERLAPPED结构中的hEvent，即可获取函数返回事件。

4. 超时设置

超时结构直接影响读和写的操作行为。当事先设定的超时时间间隔消逝时，ReadFile()、ReadFileEx()、WriteFile()和WriteFileEx()操作仍未结束，那么超时设置将无条件结束读写操作，而不管是否已读出或已写入指定数量的字符。

在读或写操作期间发生的超时将不按错误处理，即读或写操作返回指定成功的值。对于同步读或写操作，实际传输的字节数由ReadFile()和Write()函数报告。对于异步操作，则有OVERLAPPED结构来获取。

超时结构定义如下：

```
typedef struct _COMMTIMEOUTS
{
    DWORD ReadIntervalTimeout;
    DWORD ReadTotalTimeoutMultiplier;
    DWORD ReadTotalTimeoutConstant;
    DWORD WriteTotalTimeoutMultiplier;
    DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```


ReadIntervalTimeout: 以ms为单位指定通信线路上两个字符到达之间的最大时间间隔。在ReadFile()操作期间, 从接收到第一个字符时开始计时。如果任意两个字符到达之间的时间间隔超过这个最大值, 则ReadFile()操作完成, 并返回缓冲数据。如果被置为0, 则表示不使用间隔超时。

ReadTotalTimeoutMultiplier: 以ms为单位指定一个系数, 该系数用来计算读操作的总超时时间。

以ms为单位指定一个系数, 该系数用来计算写操作的总超时时间。

以ms为单位指定一个常数, 该常数也用来计算写操作的总超时时间。

超时有两种类型。第一种类型叫区间超时 (interval timeout) , 它仅适应于从端口读取数据。它指定在读取两个字符之间要经历多长时间。接收一个字符时, Windows就启动一个内部计时器。在下一个字符到达之前, 如果定时器超过了区间超时设定时间, 读函数就会放弃。第二种类型的超时叫做总超时 (total timeout) , 它适于读和写端口。当读或写特定字节数需要的总时间超过某一阈值时, 该超时即被触发。

Windows使用下面的式子计算总超时时间:

ReadTotalTimeout=(ReadTotalTimeoutMultiplierbytes_to_read)+ ReadTotalTimeoutConstant;

WriteTotalTimeout=(WriteTotalTimeoutMultiplierbytes_to_write)+ WriteTotalTimeoutConstant;

该方程使总超时时间成为灵活的工具。总超时时间不是固定值, 而是根据读或写的字节数而“漂浮不定”。应用程序通过设置系数为0而只是用常数, 和通过设置常数为0而只使用于系数。如果系数和常数都为0, 则没有总超时时间。

因此每个读操作的总超时时间等于ReadTotalTimeoutMultiplier参数值乘以读操作要读取的字节数再加上ReadTotalTimeoutConstant参数值的和。如果将ReadTotalTimeoutMultiplier和ReadTotalTimeoutConstant都设置为0, 则表示读操作不使用总超时时间。每个读间隔超时参数ReadIntervalTimeout被设置为MAXDWORK, 而且两个读总超时参数都被设置为0, 那么标识只要一读完接收缓冲区而不管得到什么字符就完成读操作, 即使它是空的。当接收中有间隔时, 间隔超时将迫使读操作返回。因此使用间隔超时的进程可以设置一个非常短的间隔超时参数, 这样它可以实现对一个或一些字符的小的、孤立的数据作出反应。

每个写操作的WriteTotalTimeoutConstant等于WriteTotalTimeoutMultiplier成员值乘以写操作要写的字节数, 再加上

WriteTotalTimeoutConstant参数值的和。如果WriteTotalTimeoutMultiplier和WriteTotalTimeoutConstant参数值都设置为0则表示写操作不使用WriteTotalTimeoutConstant。

在传输某种流量控制被阻塞时和调用SetCommBreak()函数把字符挂起, 写操作的超时可能有用。如果所有的读超时参数都为0, 即没有使用读超时, 那么读操作知道读完要求的字节数或发生错误时位置。同样, 如果所有的写超时参数都为0, 那么写操作知道要求的字节数或发生错误时为止。当打开通信资源时, 超时参数将根据上次设备被打开时所设置的值设置。如果资源从未打开或调用SetComm函数, 那么所有的超时参数都设置为0。

如果欲获得当前超时参数, 应用程序可以调用

GetCommTimeouts()函数。该函数声明如下:

BOOL GetCommTimeouts

(

HANDLE hFile, //标识通信设备, CreateFile()函数返回该句柄;

LPCOMMTIMEOUTS lpCommTimeouts //指向一个CommTIMEOUTS结构, 返回超时信息。

);

如果要设置或改变原来的超时参数, 应用程序可以调用

SetCommTimeouts()函数。该函数声明如下:

BOOL SetCommTimeouts

(

HANDLE hFile,

LPCOMMTIMEOUTS lpCommTimeouts

);

5.通信状态和通信错误

如果在串口通信中发生错误, 如发生中断, 奇偶错误等, I/O操作将会终止。如果程序要进一步执行I/O操作, 必须调用ClearCommError()函数。ClearCommError()函数有两个作用: 第一个作用是清除错误条件; 第二个作用是确定串口通信状态。

ClearCommError()函数的声明如下:

BOOL ClearCommError

(

HANDLE hFile, //标识通信设备, CreateFile()函数返回该句柄。

LPDWORD lpErrors, //指向用一个指明错误类型的掩码填充的32位变量。该参数可以是表6中各值的组合。

LPCOMSTAT lpStat //指向一个COMSTAT结构, 该结构接收设备的状态信息。如果lpStat参数不设置, 则没有设备状态信息被返回。

);

在同步操作时，可以调用ClearCommError()函数来确定串口的接收缓冲区处于等待状态的字节数，而后可以使用ReadFile()或者WriteFile()函数一次读写完。

COMSTAT结构存放有关通信设备的当前信息。该结构内容由ClearCommError()函数填写。

COMSTAT结构声明如下：

```
typedef struct _COMSTAT
(
    DWORD fCtsHold: 1;//指明是否等待CRS信号，如果为1，则发送等待。
    DWORD fDsrHold: 1;//指明是否等到DRS信号，如果为1，则发送等待。
    DWORD fRlsdHold: 1;//指明是否等待RLSD信号，如果为1，则发送等待。
    DWORD fXoffSent: 1;//指明收到XOFF字符后发送是否等待。如果为1，则发送等待。如果把XOFF字符发送给一系统时，该系统就把下一个字符当成XON，而不管实际字符是什么，此时发送将停止。
    DWORD fEof: 1;//EOF字符送出。
    DWORD fTxim: 1;//指明字符是否正等待被发送，如果为1，则字符正等待被发送。
    DWORD fReserved: 25;//系统保留
    DWORD cbInQue;//指明串行设备接收到的字节数。并不是指ReadFile操作要求读的字节数。
    DWORD cbOutQue;//指明发送缓冲区尚未发送的字节数。如果进行不重叠写操作时值为0。
} COMSTAT,*LPCOMSTAT;
```

2.4通信事件

1.通信事件

Windows可以利用GetCommMask()函数和 SetCommMask函数来控制表7所示的通信事件。

2.操作通信事件

应用程序可以利用SetCommMask()函数简历事件掩模来监视指定通信资源上的事件。

SetCommMask函数的声明如下：

```
BOOL SetCommMask
(
    HANDLE hFile,
    DWORD dwEvtMask//事件掩模，标识将被监视的通信事件。如果该参数设置为0，则表示禁止所有事件。如果不为0，则可以是表7中各种事件的组合。
);
```

如果想获取特定通信资源的当前事件掩模，可以使用GetCommMask()函数。G

etCommMask()函数声明如下：

```
BOOL GetCommMask
(
    HANDLE hFile,
    LPDWORD lpEvtMask//事件掩模，标识将被监视的通信事件，一个32位变量，可以是表7中各种事件的组合。
);
```

3.监听通信事件

在用SetCommMask()指定了有用的事件后，应用程序就调用WaitCommEvent()函数来等待其中一个事件发生。WaitCommEvent()函数既可以同步使用，也可以异步使用。

WaitCommEvent()函数声明如下：

```
BOOL WaitCommEvent
(
    HANDLE hFile,
    LPDWORD lpEvtMask,
    LPOVERLAPPED lpOverlapped,
);
```

hFile：标识通信设备，CreateFile()函数返回该句柄。

dwEvtMask: 指向一个32位变量, 接收事件掩模, 标识所发生的通信事件属于何种类型。可以是表7中各种事件的组合。

lpOverlapped: 指向一个OVERLAPPED结构, 如果打开hFile表示的通信设备时, 指定FILE_FLAG_OVERLAPPED标志, 则该参数被忽略。如果不需要异步操作, 则这个参数不用设置。

如果lpOverlapped参数不设置或打开hFile标识的通信设备是未指定FILE_FLAG_OVERLAPPED标志, 则知道发生了指定时间或出错时, WaitCommEvent()函数才返回。如果lpOverlapped参数指向一个OVERLAPPED结构, 并且打开hFile标识的通信设备时指定了FILE_FLAG_OVERLAPPED标志, 则WaitCommEvent()函数以异步操作实现。这种情况下, OVERLAPPED结构中必须含有一个人工复位事件的句柄。和所有异步函数一样, 如果异步操作不能立即实现, 则该函数返回0, 并且GetLastError()函数返回ERROR_IO_PENDING, 以指示该操作正在后台进行。此时WaitCommEvent()函数返回之前, 系统将OVERLAPPED结构中的hEvent参数设置为无信号状态;当发生了指定时间或出错时, 系统将其设置为有信号状态。调用程序可使用等待函数确定事件对象的状态, 然后使用GetOverlappedResult()函数确定WaitCommEvent()函数的操作结束。GetOverlappedResult()函数报告该操作成功或者失败, 并且lpEvtMask()函数所指向的变量被设置以指示所发生的事件。

如果一个进程在WaitCommEvent()函数操作进行期间使用SetCommMask()函数将立即返回, 并且由EvtMask参数指向的变量被设置。

注意: WaitCommEvent()只检测发生在等待开始后的事件。例如, 如果指定EV_RXCHAR事件, 则只有当收到函数字符并将字符放进接收缓冲区后才能满足等待条件。WaitCommEvent()调用时已在接收缓冲区中的字符不符合等待条件。监视CTS、DSR等信号状态改变的事件时, WaitCommEvent()函数只报告信号的变动, 但不报告当前的信号状态, 如果要查询这些信号状态, 进程可以调用GetCommModemstatus()函数。

```
#include "windows.h"
```

```
#include  
#include <TCHAR.H>  
#include <string.h>
```

```
using namespace std;
```

```
HANDLE hCom; //全局变量, 串口句柄
```

```
int serial_open(LPCWSTR COMx, int BaudRate) {
```

```
1  hCom = CreateFile(COMx, //COM1口  
2  GENERIC_READ | GENERIC_WRITE, //允许读和写  
3  0, //独占方式  
4  NULL,  
5  OPEN_EXISTING, //打开而不是创建  
6  0, //重叠方式FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED (同步方式设置为0)  
7  NULL);  
8  if (hCom == INVALID_HANDLE_VALUE)  
9  {  
10     printf("打开COM失败!\n");  
11     return FALSE;  
12 }  
13 SetupComm(hCom, 1024, 1024); //输入缓冲区和输出缓冲区的大小都是1024  
14  
15     //设定读写超时  
16     /*COMMTIMEOUTS TimeOuts;  
17     TimeOuts.ReadIntervalTimeout=1000;  
18     TimeOuts.ReadTotalTimeoutMultiplier=500;  
19     TimeOuts.ReadTotalTimeoutConstant=5000; //设定写超时  
20     TimeOuts.WriteTotalTimeoutMultiplier=500;  
21     TimeOuts.WriteTotalTimeoutConstant = 2000;  
22     SetCommTimeouts(hCom, &TimeOuts); //设置超时  
23     * /  
24 DCB dcb;  
25 GetCommState(hCom, &dcb);  
26 dcb.BaudRate = BaudRate; //设置波特率为BaudRate  
27 dcb.ByteSize = 8; //每个字节有8位  
28 dcb.Parity = NOPARITY; //无奇偶校验位  
29 dcb.StopBits = ONESTOPBIT; //一个停止位  
30 SetCommState(hCom, &dcb); //设置参数到hCom
```

```
31 PurgeComm(hCom, PURGE_TXCLEAR | PURGE_RXCLEAR); //清空缓存区 //PURGE_TXABORT 中断所有写操作并立即返回，即使写操作
32 //PURGE_RXABORT 中断所有读操作并立即返回，即使读操作还没有完成。
33 //PURGE_TXCLEAR 清除输出缓冲区
34 //PURGE_RXCLEAR 清除输入缓冲区
35 return TRUE;
```

```
}
int serial_write(char lpOutBuffer[]) //同步写串口
{
    DWORD dwBytesWrite = sizeof(lpOutBuffer);
    COMSTAT ComStat;
    DWORD dwErrorFlags;
    BOOL bWriteStat;
    ClearCommError(hCom, &dwErrorFlags, &ComStat);
    bWriteStat = WriteFile(hCom, lpOutBuffer, dwBytesWrite, &dwBytesWrite, NULL);
    if (!bWriteStat)
    {
        printf("写串口失败!\n");
        return FALSE;
    }
    PurgeComm(hCom, PURGE_TXABORT | PURGE_RXABORT | PURGE_TXCLEAR | PURGE_RXCLEAR);
    return TRUE;
}
void Serial_close(void) //关闭串口
{
    CloseHandle(hCom);
}
int main()
{
    serial_open(_T("COM1"), 4800); //打开COM1，波特率为4800
    serial_write("a"); //发送字符a
    Serial_close(); //关闭串口
    system("pause");
    return 0;
}
```

文档：Windows串口API.note
链接：<http://note.youdao.com/noteshare?id=f64e65104ecd3a31c3ba50192ee2a117&sub=81ECCBF3741447E086DD9EC7DED713F4>

windows下串口初步编程（多线程+windows串口） rabbitjerry的专栏 6067
环境 win10操作系统 编程环境：Eclipse、Cygwin GCC编译器 串口环境：串口调试助手v2.1、虚拟串口助手（Virtual Serial Port Driver 7.2） 过程 1. 在E...

串口通信（windows C++）全代码 02-21
串行通信口简称串口。美国电子工业协会EIA按电气标准及协议来分，包括RS-232C、RS-422、RS-485、USB等。RS-232、RS-422与RS-485标准只对...

评论 1 您还未登录，请先 登录 后发表或查看评论

Windows串口编程_welshon的博客_windows 串口编程 5-8
Windows串口编程 1.串行接口是采用串行通信方式的扩展接口.将数据一位一位地顺序传送。 2.基本概念 (1) RS232电平 and TTL电平 电平信号是用信号线电...

Windows下 串口编程 代码_残酷中进击的博客_windows串口... 6-23
粗略的对windows下串口的编程写了常用的封装函数(包含打开 关闭 初始化 读写),测试过的 HANDLECUART::uart_open(constchar*uart_str){ HANDLE hC...

Windows API串口编程详解 qq_33800853的博客 1134



Flywithdawn
码龄2年 暂无认证

55 9万+ 4万+ 4万+ 等级
原创 周排名 总排名 访问

743 206 60 11 85
积分 粉丝 获赞 评论 收藏



私信

关注

2020年 34篇



Flywithdawn

关注



5



1



29



专栏目录