Article ID: 1007-1202(2000)03-0323-06

Serial Communications Programming in Windows95 with API Functions*

Wang Cheng-hu, Yang Zi-jie†, Wen Bi-yang, Cheng Feng
College of Electronic Information, Wuhan University, Wuhan 430072, China

Abstract This paper introduces the usage of these functions. It presents the technique of how to program with them to perform the serial communications between a computer and other digital devices by the serial ports in Windows95. The related sample codes of the applications of this technique written in C++ programming language are also given. This technique is applied successfully to receive the data of the radio noise from a frequency monitor.

Key words Windows95; serial communications; API function; C++
CLC number TP 311.1

Generally, there is one or two standard serial port, abbreviated to COM1 or COM2, on IBM-PC and its compatible computers. In many application systems, the master computer needs transfer data and commands with the slaves' by serial ports.

In DOS environment, designers can operate the physical address of the serial port directly and the serial communications can be performed by using the software interruption provided by DOS or BIOS.

In Windows environment, the serial port, being a part of the system resources, is managed by the device driver (Comm.drv) and the direct access to it is not allowed as in DOS. Windows3.1 provides special serial communications API functions to set, write and read the serial port. In Windows95, a 32-bit-OS, the serial port and the other communications devices are all taken as files. The Win32 API functions of operating a serial port (including opening, reading, writing and closing) are the same as that of operating a common file, except that the values of a few parameters are different. Special functions are also provided to perform some useful functions to meet different needs. So these Win32 API functions are more convenient, efficient and generally applied. Based on the practically programming experience, it is introduced how to perform the serial communications by using Win32 API functions.

1 Related API Functions

Now, several API functions associated with the serial communications will be introduced. Others not introduced in detail could be referred to Microsoft Developper Network (MSDN).

1.1 Opening a serial port

The CreateFile function opens a serial port and returns a handle that can be used to access it.

```
HANDLE CreateFile(
  LPCTSTR lpCommName,                  //pointer to name of the port
  DWORD dwDesiredAccess,               //access (read-write) mode
  DWORD dwShareMode,                   //share mode
  LPSECURITY_ATTRIBUTES lpSA,          //pointer to security attributes, NULL is usual
  DWORD dwCreationDistribution,        //how to create
```

* Received date: 2000-01-15 † To whom correspondence should be addressed. Tel: 86-27-87684365
Foundation item: Supported by the 863 High Technology Project of China (863-818-01-02)
Biography: Wang Cheng-hu (1975-), male, Master candidate, research direction: ocean remote sensing.

```
    DWORD dwFlagsAndAttributes,          //port attributes
    HANDLE hTemplateFile                 //handle to file with attributes to copy
);
```

The usage of some parameters should be noted. The serial port can not be shared and it exists objectively, so dwShareMode must be 0 and dwCreationDistribution could be OPEN_EXISTING only. Besides NULL, FILE_FLAG_OVERLAPPED is the alternative of dwFlagsAndAttributes. And the hTemplateFile parameter must be NULL in Windows95.

If the function fails, the return value is INVALID_HANDLE_VALUE. If it succeeds, the return value is an open handle to the port which is represented by hComm in this paper and the default input and output buffers are given. In order to use the memory efficiently, the SetupComm function should be called to set the size of the buffers.

Here is a sample.

```
HANDLE hComm;
hComm= CreateFile("COM1", GENERIC_READ| GENERIC_WRITE, 0, NULL,
                  OPEN_EXISTING, NULL, NULL); //open COM1
if (hComm== INVALID_HANDLE_VALUE)
    {… }      //deal with the error
SetupComm(hComm, 1024, 512); //set input buffer to 1024 bytes and output buffer to 512 bytes
```

1.2 Setting a serial port

1) COMMPROP structure

The COMMPROP structure is used by the GetCommProperties function to return allowable settings about a serial port, such as baud rate, data bits, parity and so on.

2) DCB structure

The DCB structure defines the current control settings for a serial port. Before changing the settings of a serial port, new settings must be checked on whether they are allowed in the COMMPROP structure or not. Only the allowable settings can be written into the DCB structure by using the GetCommState and SetCommState functions.

The following sample code illustrates setting baud rate to 9600bps.

```
COMMPROP commprop;
DCB dcb;
GetCommProperties(hComm, &commprop);
GetCommState(hComm, &dcb);
if (commprop.dwSettableBaud& Baud_9600) //check on before setting
    dcb.dwBaudRate= 9600;
SetCommState(hComm, &dcb);
```

3) Setting Time-out

In the serial communications, it is a key problem how to deal with the unpredictable accidents like sudden interruptions during data transfer etc, which might cause the I/O operation pending or jammed. In order to avoid these situations, the COMMTIMEOUTS structure is used in the SetCommTimeouts and GetCommTimeouts functions to set and query the time-out parameters for a serial port which determine how long it is before ReadFile or WriteFile operation is abandoned.

```
typedef struct_COMMTIMEOUTS{
    DWORD ReadIntervalTimeout;          //the maximum time allowed to elapse between the
                                        //arrival of two characters
    DWORD ReadTotalTimeoutMultiplier;   //the multiplier used to calculate total time-out for read
    DWORD ReadTotalTimeoutConstant;     //the constant used to calculate total time-out for read
    DWORD WriteTotalTimeoutMultiplier;  //the multiplier used to calculate total time-out for write
    DWORD WriteTotalTimeoutConstant;    //the constant used to calculate total time-out for write
```

}COMMTIMEOUTS,* LPCOMMTIMEOUTS;

There are two kinds of time-out whose unit is millisecond. One is interval time-out which is applied in read operations only. During a ReadFile operation, if the interval between the arrival of any two sequent characters exceeds this amount, the operation is completed and any buffered data is returned. A value of zero indicates that the interval timeouts are not used. A value of MAXDWORD specifies that the read operation is to return immediately with the characters that have already been received, even if no characters have been received.

The other is total time-out. If the time of reading or writing a certain number of characters exceeds this amount, the read or write operation is completed. A value of zero indicates that the total timeouts are not used. The equation of calculating the total time-out period is shown below.

Total time-out= Read/WriteTotalTimeoutMultiplier* the number of bytes to read/write+
Read/WriteTotalTimeoutConstant

Here is a sample.

```
COMMTIMEOUTS timeouts;
GetCommTimeouts(hComm, &timeouts);
timeouts.ReadIntervalTimeout= 5;
SetCommTimeouts(hComm, &timeouts);
```

Certainly, checking on the COMMPROP structure is necessary before setting.

1.3 Reading a serial port

The ReadFile function reads data from a serial port.

```
BOOL ReadFile(
  HANDLE hComm,                    //handle of the serial port to read
  LPVOID lpBuffer,                 //address of buffer that receives data
  DWORD nNumberOfBytesToRead,      //number of bytes to read
  LPDWORD lpNumberOfBytesRead,     //address of number of bytes read
  LPOVERLAPPED lpOverlapped        //address of structure for overlapped read
);
```

If this function succeeds, the return value is nonzero. And it should be noted that the actual number of bytes read may be less than that of need, because of all kinds of possible accidents in the communications.

1.4 Writing a serial port

The WriteFile function writes data to a serial port.

```
BOOL WriteFile(
  HANDLE hComm,                    //handle of the serial port to write
  LPVOID lpBuffer,                 //pointer to data to write
  DWORD nNumberOfBytesToWrite,     //number of bytes to write
  LPDWORD lpNumberOfBytesWritten,  //address of number of bytes written
  LPOVERLAPPED lpOverlapped        //address of structure for overlapped write
);
```

The usage of this function is similar to that of the ReadFile function.

1.5 Closing a serial port

The serial port is not a sharable device. If it is opened by an application program, it can not be accessed by others until it is closed. So the CloseHandle function is required to close an open handle before the program ends.

```
BOOL CloseHandle(HANDLE hComm);
```

2 Serial Communications Modes

There are two modes of the serial communications, i. e. query mode and event-driven mode. Each of them has its own specialties and users can choose either according to their actual needs.

2.1 Query mode

The query mode is the simplest method of receiving data from a serial port. A thread continuously queries a serial port. Read them if there are data in the input buffer or go on querying if not. It is easy to program and to be understood, but it occupies so much CPU time that its efficiency is very low.

A sample of query mode is below.

```
COMMTIMEOUTS timeouts;
COMMPROP commprop;
...
DWORD ReadThread(LPDWORD lpdwParam)
{BYTE inbuff[100];                  //data buffer
 DWORD nBytesRead;                  //number of bytes read
 //check on whether interval time-out setting is allowed
 if (! (commprop.dwPropCapabilities & PCF_INTTIMEOUTS))
    return 1L;      //error: can not set interval times-outs
 //set the interval time-out
 memset(&timeouts, 0, sizeof(COMMTIMEOUTS));
 timeouts.ReadIntervalTimeout= MAXDWORD;
 SetCommTimeouts(hComm, &timeouts);
 while(bReading)
 {if (! ReadFile(hComm, inbuff, 100, &nBytesRead, NULL))
    {... }      //deal with the error
  else
    {... }      //deal with the data read
 } //end while
 PurgeComm(hComm, PURGE_RXCLEAR);       //clear the input buffer
 return 0L;
} //end thread
```

In this program, bReading determines whether the read thread is performed or not and is set by the control thread. When it is TRUE, the loop is active and the ReadFile function is called. When FALSE, the loop and the thread end.

2.2 Event-driven mode

The event-driven mode in Windows95 is similar to the interruption mode in DOS. Using this mode, the thread controls the I/O operations by monitoring a set of events which are listed (in Table 1).

Table 1 Event masks and their meanings

Event mask	meaning
EV_BREAK	A break was detected on input
EV_CTS	The CTS (clear-to-send) signal changed state
EV_DSR	The DSR (data-set-ready) signal changed state
EV_ERR	A line-status error occurred
EV_RING	A ring indicator was detected
EV_RLSD	The RLSD (receive-line-signal-detect) signal changed state
EV_RXCHAR	A character was received and placed in the input buffer
EV_RXFLAG	The event character was received and placed in the input buffer
EV_TXEMPTY	The last character in the output buffer was sent

When programming, the SetCommMask and GetCommMask functions are used to set and query the current events of the serial port. Then the WaitCommEvent function is called to monitor the specified events.

```
BOOL WaitCommEvent(
  HANDLE hComm,                    //handle of the serial port
  LPDWORD lpdwEventMask,           //address of variable for event that occurred
  LPOVERLAPPED lpOverlapped        //address of overlapped structure
);
```

If this function succeeds, the return value is not zero and the mask of the event that occurred is return in the second parameter—— lpdwEventMask.

When the arrival of data is infrequent or irregular, using the event-driven mode, the thread need not continuously query the serial port so that the valuable CPU time is saved and the efficiency is increased.

The following sample code illustrates how to receive data by this mode. It is the read thread part of the environment radio noise detecting software that is used to receive the data of the noise to process.

```
COMMTIMEOUTS timeouts;
COMMPORP commporp;
…
DWORD ReadThread(LPDWROD lpdwParam)
{BYTE* inbuff;
 DWORD nBytesRead, dwEvent, dwError;
 COMSTAT comstat;        //define a COMSTAT structure
 if (! (commprop.dwPropCapabilities& PCF_TOTALTIMEOUTS))
     return 1L;          //error: can not set total time-out
 //set the total time-out
 memset(&timeouts, 0, sizeof(COMMTIMEOUTS));
 timeouts.ReadTotalTimeoutMultiplier= 5;
 timeouts.ReadTotalTimeoutConstant= 50;
 SetCommTimeouts(hComm, &timeouts);
 SetCommMask(hComm, EV_RXCHAR);        //set the event to EV_RXCHAR
 while(1)
 { if(WaitCommEvent(hComm, &dwEvent, NULL))        //monitor the event
     // an event occurred
    //delay a period of time to receive all the bytes of one detection delay();
    {//make sure the number of bytes in inpnt buffer by COMSTAT structure
     ClearCommError(hComm, &dwError, &comstat);
     if(dwEvent& EV_RXCHAR)        //check on whether the event is EV_RXCHAR
         //if it is, read the data in the input buffer
         {inbuff= new BYTE[comstat.cbInQue]; //set the size of the data buffer
          //read operation
          if(! ReadFile(hComm,inbuff, comstat.cbInQue, &nBytesRead,NULL))
               {… }        //deal with the error
          else if (n BytesRead)
              {… }        //deal with the data read
          delete inbuff; //clear the data buffer
         }
     else
         {… } //deal with other events
    } //end if
```

```
    } //end while
    PurgeComm(hComm, PURGE_RXCLEAR);        //clear the input buffer
    return 0L;
} //end thread
```

In this sample, after setting the event to EV_RXCHAR, WaitCommEvent will return whenever a character is received and placed in the input buffer. Then check on the value of dwEventMask. If it is EV_RXCHAR, call ReadFile to read all the data in the input buffer. If not, an error event occurred and it is required to be dealed with.

3 Result

Using the technique introduced in this paper, it is easy to deal with the serial communications programming in Windows95 according to the followingsteps: open a serial port and obtain the handle, set its parameters to the working conditions as needed, read or write the data and close it finally. The compiled code of the program is small so that the performing efficiency and reliability are both high. In addition, this technique has great practicability. The program based on the event-driven mode has been already sucessfully applied in the enviroment radio noise detecting software of the high frequency ground wave radar system. It can instantly receive the real-time information of the radio noise sent by the frequency monitor. And this technique is able to be applied in other application systems also.

Referrences

[1] Berry John Thomas. *The Waite Group's C++ Programming*. Beijing: Xueyuan Press, 1994.
[2] Mirho Charles A, Terrisse Andre. *Windows95 Communications Programming*. Beijing: Tsinghua Press, 1997.

Abstract [**In** *Wuhan Daxue Xuebao* (*Ziran Kexue Ban*), 2000, **45**(3): 373~375]

The Serial Communication Based on Multithreading Technique of Windows

CHEN Shu-zhen, SHI Bo
(College of Electronic Information, Wuhan University, Wuhan 430072, China)

Abstract Present a kind of method which is used to communicate between serial serial port and peripheral equipment dynamicly and real-time using multithreading technique based on the basic principle of communication and multitasking mechanism in the circumstance of Windows. This method resolves the question of Real-time answering in the serial communication validly, reduces losing rate of data and improves reliability of system. This article presents a general method used in the serial communication which is practical.

Key words multithreading; serial communication; real-time query