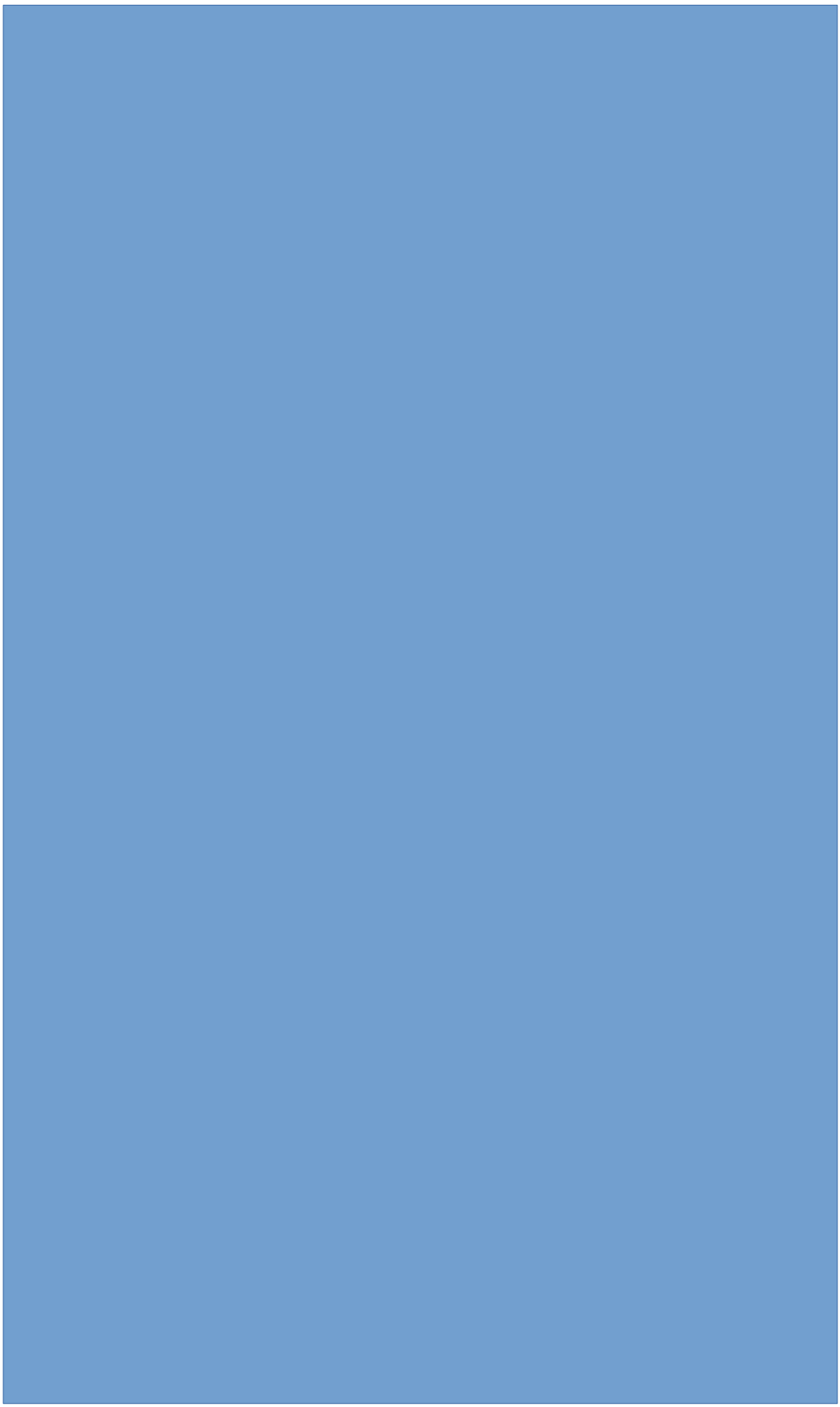
# 9 장 윈도 95 의 통신 함수

통신 자원에 대한 사전적인 정의는 "양방향 비동기 자료의 흐름(bidirectional asynchronous data stream)을 제공하는 논리적 물리적 장치"이다. 직렬 포트, 병렬 포트, 팩스, 모뎀 같은 것들이 바로 통신 자원의 예이다. 다음의 설명은 꼭 직렬 포트에만 해당되는 것이 아니기 때문에 통신 포트나 직렬 포트라는 용어 대신 통신 자원이란 용어를 사용한다. 응용프로그램이 통신 자원에 접근할 수 있도록 하기 위해 각 통신 자원에 대해서는 라이브러리나 드라이버로 구성된 서비스 제공자(service provider)가 있다.

# 윈도 3.1 과는 어떻게 다른가?

윈도 95 에서 통신 자원을 이용하는 프로그램을 작성하고자 하는 사람은 시작부터 약간 당황하게 된다. 왜냐하면 아무리 도움말을 찾아 보아도 윈도 3.1 에서 사용하던 함수들이 사라져 버렸기 때문이다. 가장 기본적인 함수라고 볼 수 있는 OpenComm, CloseComm, ReadComm, WriteComm 같은 함수들 조차 찾아 볼 수가 없다. 윈도 95 의 32 비트 환경에서는 통신 자원에 읽기 쓰기를 위해 통신자원을 열고 닫는데 사용하는 함수가 윈도 3.1 에서 사용하던 API 와는 완전히 다르다. 윈도 3.1 에서는 통신용 함수가 따로 존재했지만 WIN32 API 에서는 파일 입출력 함수가 사용된다. 다음의 표는 윈도 3.1 과 WIN32 의 통신 함수를 비교한 것이다.

아래 표를 살펴보면 윈도 3.1 에는 있던 OpenComm, CloseComm, ReadComm, WriteComm 같은 기본적인 함수가 WIN32 에는 보이지 않는 것을 알 수 있다. 앞에서도 잠깐 살펴보았지만 윈도 95 같은 32 비트 환경에는 통신 장치를 열고 닫고, 읽고 쓰고 하는 함수가 따로 존재하지 않는다. CreateFile, CloseHandle, ReadFile, ReadFileEx, WriteFile, WriteFileEx 같은 파일 입출력 함수를 통신 자원에도 그대로 이용하게 된다. 따라서 윈도 95 에서 통신 프로그램을 작성하려면 통신 API 뿐만 아니라, 파일 입출력 함수의 사용법도 잘 알아야 한다.



# 윈도 95 의 통신함수

## 통신 자원의 핸들 얻기

통신 자원의 핸들을 열기 위해서는 CreateFile 함수를 이용한다. 통신 자원을 CreateFile 함수를 이용해서 열기 위해서는 다음과 같은 속성을 설정해 주어야 한다.

(ㄱ) 지정한 자원의 읽기-쓰기 접근의 종류

(ㄴ) 핸들이 자 프로세스(child process)로 상속되는지 여부

(ㄷ) 핸들이 중첩(overlapped) 입출력을 사용할 것인지 여부

```
HANDLE CreateFile(
  LPCTSTR lpFileName,  // pointer to name of the file
  DWORD dwDesiredAccess,      // access (read-write) mode
  DWORD dwShareMode,          // share mode
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,// pointer to security descriptor
  DWORD dwCreationDistribution,          // how to create
  DWORD dwFlagsAndAttributes,          // file attributes
  HANDLE hTemplateFile          // handle to file with attributes to copy
 );
```

통신 자원을 열기 위해서는 다음에 열거하는 인수들을 반드시 지정된 값으로 설정해 주어야 한다.

(ㄹ) fdwShareMode 는 반드시 0 으로 해야 한다. 통신 자원은 공유 자원이 아니기 때문이다.

(ㅁ) fdwCreate 는 반드시 OPEN_EXISTING 으로 한다.

(ㅂ) hTemplateFile 은 반드시 NULL 로 한다.

```
HANDLE hCom;
hCom = CreateFile("COM1",
                  GENERIC_READ | GENERIC_WIRTE,
(ㄱ)
                  0,        // 통신자원은 배타접근 방식으로 열어야 한다.          (ㄹ)
                  NULL,     // 보안 속성 없음                                    (ㄴ)
                  OPEN_EXISTING,// 통신자원은 항상 이 플래그를 이용해야 한다(ㅁ)
                  0,        // 중첩 입출력 아님
        (ㄷ)
                  NULL      // hTemplateFile 은 반드시 NULL 이여야 한다.
        (ㅂ)
                  );
```

# 통신 자원의 설정값 수정

CreateFile 함수가 직렬 통신 자원을 열면, 시스템은 지난번 자원이 열렸을 때 설정값에 따라 초기화를 수행한다. 이전의 설정값을 저장해 두었다가 나중에 복구해 주는 것이 바람직하다. 만약 이전에 한번도 통신 자원이 열린적이 없다면 물론 시스템의 기본값으로 열린다. 직렬 통신 자원의 초기 구성을 알아내려면 GetCommState 함수를 이용한다. 이 함수는 장치 제어 블록(device control block : DCB) 구조체에 현재의 설정값들을 채워 넣는다. 이 구성값을 수정하려면 DCB 구조체의 각 필드에 적당한 값을 써 넣은 다음 SetCommState 함수를 이용하면 된다.

```
typedef struct _DCB { // dcb
    DWORD DCBlength;           // sizeof(DCB)
    DWORD BaudRate;            // current baud rate
    DWORD fBinary: 1;          // binary mode, no EOF check
    DWORD fParity: 1;          // enable parity checking
    DWORD fOutxCtsFlow:1;      // CTS output flow control
    DWORD fOutxDsrFlow:1;      // DSR output flow control
    DWORD fDtrControl:2;       // DTR flow control type
    DWORD fDsrSensitivity:1;   // DSR sensitivity
    DWORD fTXContinueOnXoff:1; // XOFF continues Tx
    DWORD fOutX: 1;            // XON/XOFF out flow control
    DWORD fInX: 1;             // XON/XOFF in flow control
    DWORD fErrorChar: 1;       // enable error replacement
    DWORD fNull: 1;            // enable null stripping
    DWORD fRtsControl:2;       // RTS flow control
    DWORD fAbortOnError:1;     // abort reads/writes on error
    DWORD fDummy2:17;          // reserved
    WORD wReserved;            // not currently used
    WORD XonLim;               // transmit XON threshold
    WORD XoffLim;              // transmit XOFF threshold
    BYTE ByteSize;             // number of bits/byte, 4-8
    BYTE Parity;               // 0-4=no,odd,even,mark,space
    BYTE StopBits;             // 0,1,2 = 1, 1.5, 2
    char XonChar;              // Tx and Rx XON character
    char XoffChar;             // Tx and Rx XOFF character
    char ErrorChar;            // error replacement character
    char EofChar;              // end of input character
    char EvtChar;              // received event character
    WORD wReserved1;           // reserved; do not use
} DCB;
```

DCB 구조체의 각 필드는 는 전송률, 바이트 당 데이터 비트의 수, 바이트 당 정지 비트의 수와 같은 값들과 흐름 제어, 패리티 검사 등에 대한 값들을 나타낸다.

GetCommState 와 SetCommState 를 이용해서 DCB 의 설정값을 바꿔주는 것은 상당히 복잡해 보일뿐만 아니라, 사실 DCB 의 구조체의 모든 변수들을 다 알아야 할 필요도 없다.

BuildCommDCB 함수는 DCB 구조체를 수정하는 좀 더 쉬운 방법을 제공해 준다. 이 함수는 DOS 의 MODE 명령에서 사용하는 명령행 인자와 같은 형식의 문자열을 이용해서 DCB 를 설정하는 함수이다.

```
C:\WIN95>MODE /?
Configures system devices.
```

```
Printer port:      MODE LPTn[:] [COLS=c] [LINES=l] [RETRY=r]
Serial  port:            MODE  COMm[:]  [BAUD=b]  [PARITY=p]  [DATA=d]  [STOP=s]
[RETRY=r]
Device Status:     MODE [device] [/STATUS]
Redirect printing: MODE LPTn[:]=COMm[:]
Prepare code page: MODE device CP PREPARE=((yyy[...]) [drive:][path]filename)
Select code page:  MODE device CP SELECT=yyy
Refresh code page: MODE device CP REFRESH
Code page status:  MODE device CP [/STATUS]
Display mode:      MODE [display-adapter][,n]
                MODE CON[:] [COLS=c] [LINES=n]
Typematic rate:    MODE CON[:] [RATE=r DELAY=d]
```

```
BOOL BuildCommDCB(
   LPCTSTR lpDef,          // pointer to device-control string
   LPDCB lpDCB // pointer to device-control block
  );
```

```
DCB dcb;
BuildCommDCB("COM1: baud=14400 parity=N data=8 stop=1", &dcb);
SetCommState(&dcb);
```

SetCommState 함수는 통신 자원을 재구성하기는 하지만 지정한 드라이버 내부의 입출력 버퍼에는 영향을 미치지 못한다. 버퍼는 초기화 되지 않으면 읽기 쓰기 동작이 아직 대기 중이라면 역시 그대로 대기 중인체로 있게 된다. 통신 자원을 완전히 재 초기화하려면 SetupComm 함수를 사용한다. 이 함수는 다음과 같은 기능을 수행한다.

(1) 대기 중인 읽기 쓰기 작업을 완료되지 않았더라도 무조건 끝마친다.

(2) 내부 입출력 버퍼에 있는 문자들은 모두 버려진다.

(3) 내부 입출력 버퍼를 다시 할당받는다.

# 통신 자원의 구성

## 읽기 쓰기 동작

Win32 API 는 직렬 통신 장치에서 동기 파일 입출력과 비동기 파일 입출력 둘 모두를 지원한다. 동기 파일 입출력 방식이란 파일에서 자료를 읽거나 쓸 때, 그 동작이 끝나기 전까지는 주 프로세스로 제어가 돌아 오지 않는 방식을 말한다. 예를 들어, 어떤 파일에서 2M 바이트를 읽는다고 가정해 보자. 파일 읽기 함수는 2M 바이트를 다 읽을 때까지는 종료되지 않는다. 이렇게 파일을 읽는 도중에는 다른 작업이 불가능하다. 도스와 같은 단일 태스킹 운영체제에서는 기본적으로 파일 입출력은 동기 방식으로만 동작한다.

비동기 파일 입출력 방식은 파일에서 자료를 읽거나 쓸 때, 그 동작이 완전히 끝나기 전에도 주 프로세스로 제어가 돌아 오는 방식을 말한다. 앞에서 예를 들었던 2M 바이트 짜리 파일을 한꺼번에 읽는다고 가정해 보자. 비동기 파일 입출력 방식에서는 파일 읽기 함수는 호출되자 마자 곧바로 종료된다. 그렇다고 해서 실제 파일 읽기가 종료되는 것은 아니다. 파일 읽기 동작은 배경(background) 작업으로 계속되고 있는 것이다. 이렇게 되면 파일 읽기를 시작한 직후에 바로 다른 작업을 수행할 수 있게 되는 것이다. 대신에 비동기 방식의 파일 입출력은 그 작업의 종료 여부를 따로 판단해야 하기 때문에 사용법이 상대적으로 복잡하다. 비동기 방식의 파일 입출력은 중첩(overlapped) 파일 입출력이라고도 한다. overlap 의 사전적 의미에는 "시간적으로 일부분이 일치하다, 중첩되다"는 뜻이 있는데 비동기 방식의 파일 입출력에 적당한 이름이라 생각된다. 통신 자원에 자료를 쓸 때는 WriteFile, WriteFileEx 함수를 사용하고, 통신 자원에서 자료를 읽어 올 때는 ReadFile, ReadFileEx 함수를 이용한다. WriteFile 과 ReadFile 은 동기, 비동기 방식 어떤 것으로도 동작할 수 있으나 WriteFileEx 와 ReadFileEx 는 비동기 방식으로만 동작한다.

이런 읽기 쓰기 함수는 중첩 작업으로 수행되는지, 그 통신 자원 핸들에 시간 초과 인수나 흐름 제어 인수에 영향을 받게 된다.

TransmitCommChar 는 출력 버퍼에 남아 있는 자료들 가장 앞에 문자를 위치 시킨다. 이 함수는 수신 시스템으로 우선 순위가 높은 어떤 신호 문자를 보낼 때 유용하다. 이 문자의 전송 역시 흐름제어나 쓰기 시간 초과 등에 영향을 받긴 하지만 동작은 동기적으로 이뤄진다.

PurgeComm 은 읽기 쓰기 동작이 아직 종료되지 않았더라도 그 작업을 중지시켜 버린다. 만약 출력 버퍼를 비워 버리고 싶다면 PurgeComm 함수를 사용하면 된다. 이렇게 하면 출력 버퍼에 남아 있던 문자들은 전송되지 않고 삭제된다. PurgeComm 함수는 윈도 3.1 의 FlushComm 함수에 해당한다고 볼 수 있겠다. FlushFileBuffers 함수는 같은 동작을 하지만 동기적으로만 동작한다. 하지만 이 FlushFileBuffers 함수는 모든 대기 중인 동작이 완료되기 전까지는 복귀하지 않는다.

# 중첩 작업(Overlapped Operaions)

중첩 작업은 하나의 쓰레드가 시간이 많이 걸리는 작업을 배경(background) 작업으로 수행 하면서 다른 작업을 계속 할 수 있도록 해 준다. 통신 자원에 대해 중첩 입출력 작업을 가능하게 하려면 그 쓰레드는 통신 자원의 핸들을 열 때, CreateFile 함수에 FILE_FLAG_OVERLAPPED 플래그를 지정해 주어야 한다. ReadFile 이나 WriteFile 이 중첩 작업을 수행할 수 있도록 하려면 OVERLAPPED 구조체의 포인터를 지정해 주어야 한다.

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

```
BOOL ReadFile(
    HANDLE hFile,            // handle of file 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 data
   );
```

```
```

```
BOOL WriteFile(
  HANDLE hFile,            // handle to file to write to
  LPCVOID lpBuffer,        // pointer to data to write to file
  DWORD nNumberOfBytesToWrite,        // number of bytes to write
  LPDWORD lpNumberOfBytesWritten,// pointer to number of bytes written
  LPOVERLAPPED lpOverlapped // addr. of structure needed for overlapped I/O
 );
```

```
BOOL ReadFileEx(
  HANDLE hFile,            // handle of file to read
  LPVOID lpBuffer,         // address of buffer
  DWORD nNumberOfBytesToRead,        // number of bytes to read
  LPOVERLAPPED lpOverlapped,// address of offset
  LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
                                     // address of completion routine
 );
```

```
BOOL WriteFileEx(
  HANDLE hFile,            // handle to output file
  LPCVOID lpBuffer,        // pointer to input buffer
  DWORD nNumberOfBytesToWrite,       // number of bytes to write
  LPOVERLAPPED lpOverlapped,         // pointer to async. i/o data
  LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine
                                     // ptr. to completion routine
 );
```

OVERLAPPED 구조체는 수동-리셋 이벤트 객체(manual-reset event object)의 핸들을 포함해야 한다.

시스템은 입출력 작업이 완료되기 전에는 이벤트 객체의 상태를 비-시그널(not-signaled) 상태로 설정한다. 작업이 완료되면 이벤트 객체의 상태를 시그널(signaled) 상태로 만든다. signal 에 "신호를 보내다"는 뜻이 있는 걸 생각하면 이해하기 쉬울 것이다. 쓰레드는 이벤트 객체의 현재 상태를 검사하거나, 그 상태가 시그널로 되기를 기다리기 위해 대기 함수를 사용한다.

ReadFileEx 와 WriteFileEx 함수는 중첩 작업으로만 동작된다. 이 함수를 호출하는 쓰레드는 중첩 작업이 완료되었을 때 수행되는 FileIoCompletionRoutine 함수에 포인터를 지정

해 준다.

# 시간 초과(Time-Outs)

시간 초과는 ReadFile, ReadFileEx, WriteFile, WriteFileEx 함수 수행 때 지정된 숫자의 문자가 읽히거나 쓰여지지 않은 상태로 일정시간이 흐르면 발생하게 된다. 실제로 읽거나 쓴 바이트 수는 ReadFile 이나 WriteFile 함수의 반환값으로 알아 낼 수 있다. 입출력이 중첩 작업으로 이뤄졌다면 GetOverlappedResult 나 FileIoCompletionRoutine 함수로 알아 낼 수 있다.

응용 프로그램이 통신 자원을 열었을 때, 운영체제는 그 자원의 시간초과 값을 이전에 사용했던 값으로 초기화한다. 물론 이전에 그 통신 자원이 사용된 적이 없다면 시스템 기본 값으로 설정된다. 통신 자원의 시간초과값을 읽어오려면 GetCommTimeouts 함수를 사용한다. 값을 변경하려면 SetCommTimeouts 함수를 사용한다.

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

```
BOOL GetCommTimeouts(
    HANDLE hFile,             // handle of communications device
    LPCOMMTIMEOUTS lpCommTimeouts   // address of comm. time-outs structure
   );
```

```
BOOL SetCommTimeouts(
    HANDLE hFile,             // handle of communications device
      LPCOMMTIMEOUTS  lpCommTimeouts//  address  of  communications  time-out
```

```
structure
  );
```

시간 초과에는 간격 시간 초과(interval time-out)와 총 시간 초과(total time-out) 두가지가 사용되는데, 먼저 간격 시간초과는 두 개의 문자가 수신되는 간격이 지정된 시간보다 길어 지면 발생하게 된다. 시간은 처음 문자가 수신되면 시작되었다가 새로운 문자가 수신되면 다시 시작된다. 다음으로 총 시간 초과는 읽기나 쓰기에 소요된 총 시간이 "계산된 시간"보다 길어지면 발생하게 된다. 시간이 입출력 작업이 시작되면 즉시 시작된다. 쓰기 작업은 총 시간초과만 지원하며 읽기 작업은 간격 시간초과와 총 시간초과 둘 모두를 지원하며 따로 따로 사용할 수도 함께 사용할 수도 있다. 앞에서 "계산된 시간"이라고 했는데 이 시간 계산법은 다음과 같다

읽기 쓰기 작업에 대한 총 시간 초과는 COMMTIMEOUTS 구조체의 multiplier 값과 constant 값을 이용해서 계산한다.

총 시간 초과 = (MULTIPLIER * 바이트 수) + CONSTANT

이 공식에 따르면 바이트 수에 따라 총 시간 초과값이 달라지게 된다. 응용프로그램은 MULTIPLIER 를 0 으로 함으로써 총시간 초과값을 CONSTANT 로 일정하게 유지할 수도 있으며 MULTIPLIER 와 CONSTANT 값을 둘 다 0 으로 하면 총 시간 초과는 사용되지 않는다.

만약 읽기에 대한 총 시간초과와 간격 시간초과 값이 모두 0 이면 읽기 작업에 대한 시간초과는 사용되지 않으며 일기 동작은 요구된 바이트를 다 읽거나 오류가 발생하기 전까지는 완료되지 않는다. 비슷하게, 쓰기에 대한 시간초과값이 모두 0 이라면 쓰기 작업은 요구된 바이트를 다 쓰거나 오류가 발생할 때까지 완료되지 않는다.

읽기 간격 시간초과 값이 MAXDWORD 이고 총 시간초과 값이 0 이면 읽기 작업은 입력버퍼의 내용을 읽은 후 곧바로 완료하게 된다. 입력버퍼가 비어 있더라도 상관없다. 이렇게 되면 윈도 3.1 에서 사용했던 ReadComm 함수와 똑같은 방식으로 동작하게 된다.

간격 시간초과는 자료를 수신할 때 버퍼가 비어있더라도 읽기 작업에서 복귀하도록 한다. 간격 시간초과값을 아주 짧게 설정하면 몇바이트 정도만 읽고 나서 곧바로 복귀하게 된다. 물론 이 값을 크게 설정해서 큰 버퍼에 자료가 쌓일 때까지 기다리게 할 수도 있다.

쓰기 작업에 대한 시간 초과는 전송이 어떤 흐름 제어에 의해 전송이 막혀 있거나 문자 전송을 일시적으로 중지시키기 위해 SetCommBreak 함수가 호출된 경우에 유용하게 사용될

수 있다.

다음은 총 시간초과와 간격 시간초과 값에 따른 읽기 작업을 정리한 것이다.

# 통신 오류

시간 초과 오류가 생기지 않더라도 읽기 쓰기 작업 중에 요청된 문자의 개수보다 적은 문자만이 완료되는 경우가 생긴다. 다음은 그런 예이다.

어떤 드라이버는 특정한 문자를 읽으면 그 즉시 읽기 작업을 중지하게 하는 특수문자를 사용한다.

PurgeComm 함수는 대기 중인 읽기 쓰기 작업을 마치기 위해 호출된다. 이 함수는 입출력 버퍼의 내용을 모두 지워버린다.

읽기 쓰기 작업 중에 어떤 오류가 발생하면 통신 자원 상의 모든 입출력 작업은 종료된다. 오류가 발생하면 ClearCommError 함수를 호출해야만 한다. 이 함수는 입출력 작업을 계속할 수 있도록 오류 플래그를 지워준다. 이 함수는 또한 그 장치의 현재 상태와 발생한 오류가 무엇인지를 알려준다.

# 통신 사건

프로세스는 통신 자원에서 일어나는 일련의 사건 집합을 감시한다. 응용프로그램이 통신 자원에서 CTS(Clear-To-Send)와 DSR(Data-Set-Ready)신호가 변화되는 것을 감시할 수가 있다. SetCommMask 함수가 바로 그것인데, 이 함수는 '사건 마스크'(event mask)를 만드는데 사용한다. 사건 마스크란 통신 자원에서 일어날 수 있는 여러 가지 사건(event) 중에서 원하는 것들만 선택해서 사용할 수 있도록 한 것이다. 현재 사건 마스크 상태를 알려면 GetCommEvent 함수를 사용한다.

다음은 감시할 수 있는 사건 들이다.

사건 마스크가 많다고 해서 골치 아파할 필요는 없다. 왜냐하면 통신 에뮬레이터에서는 이 중 아주 일부 이벤트 만을 사용하기 때문이다. 일반적인 통신 에뮬레이터를 실행시킨 상태에서 전화벨이 울리면 화면에

| |
|---|
| RING |
| RING |
| RING |

하는 문자들이 출력되는 것을 볼 수 있다. 이 문자들은 사실 모뎀이 통신 포트로 보내 주는 것들이다. 이 문자들은 통신 포트의 입력 버퍼에 들어오게 되는데, 이런 경우, EV_RING 뿐만 아니라, EV_RXCHAR 사건도 같이 발생하게 된다. 그러므로 EV_RXCHAR 만을 사건 이벤트로 설정해도 전화가 걸려 왔다는 정도는 쉽게 알아낼 수 있게 된다.

사건의 집합이 지정된 다음, 프로세스는 지정한 사건 중에 하나가 발생하기를 기다리기 위해 WaitCommEvent 함수를 사용한다. 이 함수는 동기적으로도, 비동기적으로 사용될 수 있다.

```
BOOL WaitCommEvent(
  HANDLE hFile,            // handle of communications device
  LPDWORD lpEvtMask, // address of variable for event that occurred
  LPOVERLAPPED lpOverlapped,            // address of overlapped structure
 );
```

```
BOOL SetCommMask(
  HANDLE hFile,            // handle of communications device
  DWORD dwEvtMask    // mask that identifies enabled events
 );
```

사건 마스크에 지정한 사건이 발생하게 되면 프로세스는 대기 함수를 마치고 사건 마스크 변수에 발생한 사건을 넣게 된다. 만약 대기 중에 SetCommMask 함수가 호출되면 WaitCommEvent 함수는 오류를 반환한다. WaitCommEvent 함수는 SetCommMask 나 WaitCommEvent 함수가 호출된 이후에 발생한 사건에 대해서 검출한다. WaitCommEvent 함수를 이용하면 CTS, DTR 같은 신호가 상태가 바뀔 때 발생하는 사건은 현재의 상태가 아니라 변화된 이후의 상태를 보고한다. CTS, DTR, RLSD 따위의 신호에 대한 현재 상태를 읽어오려면 GetCommModemStatus 함수를 사용한다.

```
BOOL GetCommModemStatus(
  HANDLE hFile,            // handle of communications device
  LPDWORD lpModemStat          // address of control-register values
 );
```

# 확장 함수

그 밖의 통신 기능을 위해서는 EscapeCommFuction 함수가 사용된다. 이 함수는 확장 기능을 수행하기 위해 그 장치에 직접 코드를 보내게 된다. 만약 응용 프로그램이 SETBREAK 코드를 보내 문자 전송을 일시 중지시키거나 CLRBREAK를 보내 전송을 재개하고 싶다면 SetCommBreak 함수와 ClearCommBreak 함수를 이용하면 된다. EscapeCommFunction 함수 역시 이러한 모뎀 제어에 사용할 수 있다. 예를 들어, CLRDTR, SETDTR 코드는 DTR 흐름 제어를 구현하기 위해 사용된다.

```
BOOL EscapeCommFunction(
   HANDLE hFile,          // handle to communications device
   DWORD dwFunc           // extended function to perform
  );
```

```
BOOL SetCommBreak(
   HANDLE hFile           // handle of communications device
  );
```

```
BOOL ClearCommBreak(
   HANDLE hFile           // handle to communications device
  );
```

DeviceIoControl 함수는 특정한 장치 조정자(device driver)에 확장 기능 코드를 직접 보내는데 사용한다. 이 함수는 표준 직렬 통신 함수에서 지원되지 않는 기능에 대한 확장 코드도 사용할 수 있게 해 준다.

```
BOOL DeviceIoControl(
   HANDLE hDevice,        // handle to device of interest
   DWORD dwIoControlCode,      // control code of operation to perform
   LPVOID lpInBuffer,     // pointer to buffer to supply input data
   DWORD nInBufferSize,           // size of input buffer
   LPVOID lpOutBuffer,    // pointer to buffer to receive output data
```

```
    DWORD nOutBufferSize,          // size of output buffer
    LPDWORD lpBytesReturned,// pointer to variable to receive output byte count
    LPOVERLAPPED lpOverlapped
                        // pointer to overlapped structure for asynchronous operation
   );
```

# 통신 함수의 사용

# 통신 자원 구성하기

다음은 COM1 에 대한 핸들을 여는 예이다.

```
DCB dcb;
HANDLE hCom;
DWORD dwError;
BOOL fSuccess;

hCom = CreateFile("COM1",
    GENERIC_READ | GENERIC_WRITE,
    0,    /* comm devices must be opened w/exclusive-access */
    NULL, /* no security attrs */
    OPEN_EXISTING, /* comm devices must use OPEN_EXISTING */
    0,    /* not overlapped I/O */
    NULL  /* hTemplate must be NULL for comm devices */
    );

if (hCom == INVALID_HANDLE_VALUE) {
    dwError = GetLastError();

    /* handle error */
}

/*
 * Omit the call to SetupComm to use the default queue sizes.
 * Get the current configuration.
```

```
 */

fSuccess = GetCommState(hCom, &dcb);

if (!fSuccess) {
   /* Handle the error. *
}

/* Fill in the DCB: baud=9600, 8 data bits, no parity, 1 stop bit. */

dcb.BaudRate = 9600;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;

fSuccess = SetCommState(hCom, &dcb);

if (!fSuccess) {
   /* Handle the error. *
}
```

# 통신 사건을 감시하기

다음은 중첩 입출력(overlapped I/O)으로 직렬 포트를 여는 예제이다. 사건 마스크는 CTS, DSR 신호에 대해서 설정한다. WaitCommEvent 함수는 기다리는 동안 프로세스의 다른 쓰레드가 입출력 작업을 수행할 수 없도록 중첩 작업으로 이루어 져야 한다.

```
HANDLE hCom;
OVERLAPPED o;
BOOL fSuccess;
DWORD dwEvtMask;

hCom = CreateFile("COM1",
   GENERIC_READ | GENERIC_WRITE,
   0, /* exclusive access */
   NULL, /* no security attrs */
   OPEN_EXISTING,
   FILE_FLAG_OVERLAPPED,
   NULL
   );

if (hCom == INVALID_HANDLE_VALUE) {
```

```
    /* Deal with the error. */
}

/* Set the event mask. */

fSuccess = SetCommMask(hCom, EV_CTS | EV_DSR);

if (!fSuccess) {
    /* deal with error */
}

/* Create an event object for use in WaitCommEvent. */

o.hEvent = CreateEvent(NULL, /* no security attributes */
    FALSE, /* auto reset event */
    FALSE, /* not signaled */
    NULL /* no name */
    );

assert(o.hEvent);

if (WaitCommEvent(hCom, &dwEvtMask, &o)) {
    if (dwEvtMask & EV_DSR) {
        /*
         * . . .
         */
    }

    if (dwEvtMask & EV_CTS) {
        /*
         * . . .
         */
    }
}
```

# 통신함수 설명

## *BuildCommDCB*

장치제어문자열로 지정된 DCB 구조체를 채운다. 장치제어문자열은 도스(DOS)의 mode

명령에서 쓰는 형식이다.

```
BOOL BuildCommDCB(
   LPCTSTR  lpDef,         // pointer to device-control string
   LPDCB  lpDCB            // pointer to device-control block
  );
```

## [인수]

lpDef

장치제어 정보를 나타내는 문자열에 대한 포인터. 이 문자열은 도스의 mode 명령의 명령행 인수와 같은 형식이다. 예를 들어, 전송률(baud rate) 1200, 패리티 없음, 데이터 비트 8, 정지비트 1 이라면,

COM1: baud=1200 parity=N data=8 stop=1

lpDCB

값을 채워 넣을 DCB 구조체에 대한 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

lpDef 인수에 따라 DCB 구조체의 각 변수들을 설정하는데 다음과 같은 예외가 있다.

(가) 전송률이 110 이면 정지비트는 2 이다. 왜냐하면 도스의 mode 명령과 호환성을 지키기 위해서이다.

(나) 기본적으로, BuildCommDCB 는 XON/XOFF 와 하드웨어 흐름제어(flow control)를 기능을 억제한다. 따라서 이 설정값을 바꿔 주려면 DCB 구조체의 변수에 값을 직접 써 주어야 한다.

이 함수는 단지 DCB 구조체에 값을 넣어주기만 할 뿐이다. 이 설정값을 직렬 포트에 적용하려면 SetCommState 함수를 사용한다.

DCB 구조체의 흐름제어에 대한 변수에 값을 설정하기 위해 mode 명령의 확장형식이 있다.

(가) 96,n,8,1

- fInX, fOutX, fOutXDsrFlow, fOutXCtsFlow 는 모두 FALSE
- fDtrControl 은 DTR_CONTROL_ENABLE
- fRtsControl 은 RTS_CONTROL_ENABLE

(나) 96,n,8,1,x

- fInX, fOutX 는 모두 TRUE
- fOutXDsrFlow, fOutXCtsFlow 는 둘 다 FALSE
- fDtrContol 은 DTR_CONTROL_ENABLE
- fRtsControl 은 RTS_CONTROL_ENABLE

(다) 96,n,8,1,p

- fInX, fOutX 는 모두 FALSE
- fOutXDsrFlow, fOutXCtsFlow 는 둘 다 TRUE
- fDtrContol 은 DTR_CONTROL_HANDSHAKE
- fRtsControl 은 RTS_CONTROL_HANDSHAKE

# *BuildCommDCBAndTimeouts*

이 함수는 BuildCommDCB 와 같은 기능을 하며 여기에 시간 초과(time-out) 값까지 설정할 수 있게 되어 있다.

```
BOOL BuildCommDCBAndTimeouts(
  LPCTSTR  lpDef, // pointer to the device-control string
  LPDCB  lpDCB,           // pointer to the device-control block
  LPCOMMTIMEOUTS  lpCommTimeouts // pointer to comm. time-out structure
 );
```

## [인수]

lpDef

장치제어 정보를 나타내는 문자열에 대한 포인터. 이 문자열은 도스의 mode 명령의 명령행 인수와 같은 형식이다.

lpDCB

값을 채워 넣을 DCB 구조체에 대한 포인터

lpCommTimeOuts

시간초과 값을 설정하기 위한 COMMTIMEOUTS 구조체에 대한 포인터.

lpDef 에 "TO=xxx"가 있는지 없는지에 따라 시간초과를 설정할 것인지 설정하지 않을

것인지를 결정한다.

(가) "TO=ON"이면 함수는 lpCommTimeOuts 에 설정된 시간초과 값에 따라 총읽기 쓰기 시간 초과값을 설정한다.

(나) "TO=OFF"이면 시간초과값을 설정하지 않는다.

(다) "TO=xxx"문자열을 포함하고 있지 않다면 lpCommTimeOuts 의 값은 무시된다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

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

# *ClearCommBreak*

지정된 통신 장치에 대한 문자 전송을 다시 시작하도록 한다.

```
BOOL ClearCommBreak(
    HANDLE  hFile            // handle to communications device
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

통신자원의 상태가 SetCommBreak 나 EscapeCommFunction 함수에 의해 중단(break) 상태가 되었을 때, ClearCommBreak 함수를 호출하면 중단 상태가 해제되면서 문자 전송이 다시 시작된다.

# *ClearCommError*

통신 오류에 대한 정보를 읽어오고 통신자원의 현재 상태를 읽어온다. 이 함수는 통신 오류가 발생했을 때 호출되며 장치의 오류 플래그를 해제하고 입출력 동작이 계속될 수 있도록 한다.

```
BOOL ClearCommError(
    HANDLE  hFile,          // handle to communications device
   LPDWORD  lpErrors,    // pointer to variable to receive error codes
   LPCOMSTAT  lpStat      // pointer to buffer for communications status
  );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpError

오류의 종류를 나타내는 32 비트 변수. 오류의 종류별로 이 변수에 마스크(mask)된다.

lpStat

장치의 상태정보가 반환될 COMSTAT 구조체의 포인터. 이 값이 NULL 이면 상태정보가 반환되지 않은 것이다.

```
typedef struct _COMSTAT { // cst
    DWORD fCtsHold : 1;   // Tx waiting for CTS signal
    DWORD fDsrHold : 1;   // Tx waiting for DSR signal
    DWORD fRlsdHold : 1;  // Tx waiting for RLSD signal
    DWORD fXoffHold : 1;  // Tx waiting, XOFF char rec'd
    DWORD fXoffSent : 1;  // Tx waiting, XOFF char sent
    DWORD fEof : 1;       // EOF character sent
    DWORD fTxim : 1;      // character waiting for Tx
    DWORD fReserved : 25; // reserved
    DWORD cbInQue;        // bytes in input buffer
    DWORD cbOutQue;       // bytes in output buffer
} COMSTAT, *LPCOMSTAT;
```

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

통신포트가 DCB 구조체의 fAbortOnError 변수가 TRUE 인 상태로 설정되면 통신 소프트웨어는 통신 오류가 발생했을 때, 통신 포트의 모든 읽기 쓰기 작업을 중단한다. 새로운 읽기 쓰기 작업은 응용프로그램이 ClearCommError 함수를 호출해서 통신 오류를 해제해야 새로 시작될 수 있다. ClearCommError 함수는 hFile 로 지정된 통신 자원의 현재 상태를 lpStat 가 지시하는 상태 버퍼에 채워준다.

# *DeviceIoControl*

지정한 장치 드라이버에 특정한 동작을 하도록 제어 코드를 직접 보낸다.

```
BOOL DeviceIoControl(
```

```
  HANDLE  hDevice,      // handle to device of interest
  DWORD  dwIoControlCode,// control code of operation to perform
  LPVOID  lpInBuffer,     // pointer to buffer to supply input data
  DWORD  nInBufferSize,          // size of input buffer
  LPVOID  lpOutBuffer,  // pointer to buffer to receive output data
  DWORD  nOutBufferSize,        // size of output buffer
  LPDWORD  lpBytesReturned,// pointer to variable to receive output byte count
  LPOVERLAPPED  lpOverlapped
                          //pointer to overlapped structure for asynchronous
operation
  );
```

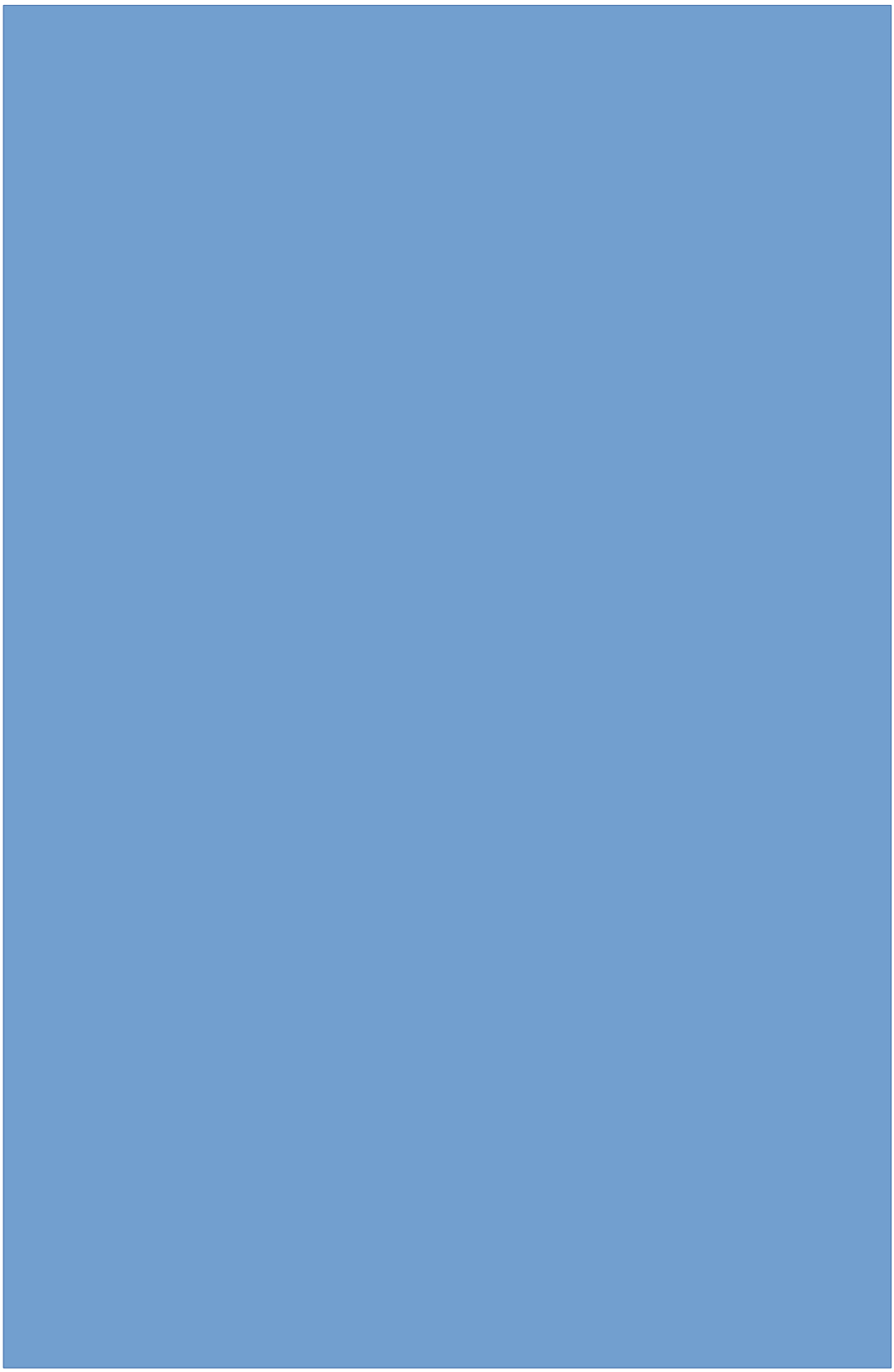
## [인수]

hDevice

동작을 수행할 장치에 대한 핸들. CreateFile 함수로 이 장치에 대한 핸들을 구할 수 있다.

dwIoControlCode

제어코드를 지정한다.



lpInBuffer

작업을 수행하는데 필요한 자료를 포함하는 버퍼에 대한 포인터

dwIoControlCode 가 입력 데이터가 필요하지 않은 작업을 지정했다면 이 값이 NULL 일 수 있다.

nInBufferSize

lpInBuffer 가 가리키는 버퍼의 바이트 단위 크기이다.

lpOutBuffer

작업의 출력 데이터를 받는 버퍼에 대한 포인터

nOutBufferSize

lpOutBufferSize 가 가리키는 버퍼의 바이트 단위 크기

lpBytesReturned

lpOutBuffer 가 가리키는 버퍼에 저장된 데이터의 바이트 단위 크기 값에 대한 포인터

lpOverlapped

비동기 작업에 사용되는 OVERLAPPED 구조체에 대한 포인터.

이 인수는 hDevice 를 FILE_FLAG_OVERLAPPED 플래그를 지정하지 않고 열었다면 무시된다. 비동기 작업을 원하지 않으면 이 변수를 NULL 로 한다.

```
typedef struct _OVERLAPPED { // o
    DWORD  Internal;
    DWORD  InternalHigh;
    DWORD  Offset;
```

```
    DWORD  OffsetHigh;
    HANDLE hEvent;
} OVERLAPPED;
```

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

lpOverlapped 가 NULL 이거나 hDevice 를 FILE_FLAG_OVERLAPPED 플래그를 지정하지 않고 열었다면 DeviceIoControl 함수는 작업이 모두 완수되거나 오류가 발생하기 전까지는 복귀하지 않는다.

lpOverlapped 가 OVERLAPPED 구조체를 지시하고, hDevice 가 FILE_FLAG_OVERLAPPED 플래그로 열렸다면, 이 함수는 중첩(overlapped) 작업(즉, 비동기적인 작업)으로 수행된다. 이 경우, OVERLAPPED 구조체는 수동 떨기(manual-reset) 사건 객체(event object)의 핸들을 포함해야 한다. 이 핸들은 CreateEvent 함수로 구한다.

중첩작업이 즉시 완수되지 못하면 함수는 FALSE 를 반환하고, GetLastError 가 ERROR_IO_PENDING 값을 반환하면 이것은 중첩작업이 배경으로 수행되고 있다는 것을 나타낸다. 이렇게 되면 운영체제는 DeviceIoControl 함수에서 복귀하기 전에 OVERLAPPED 구조체의 사건 객체를 비신호(nonsignaled) 상태로 설정한다. 운영체제는 작업이 모두 완료되면 사건 객체를 신호(signaled)상태로 설정한다. 이 함수를 호출하는 쓰레드는 사건 객체의 상태를 알아내기 위해 대기(wait) 함수 중 어떤 것이라도 사용할 수 있다. GetOverlappedResult 함수가 그 동작의 결과를 알아내는데 사용될 수 있다.

```
BOOL GetOverlappedResult(
    HANDLE  hFile,            // handle of file, pipe, or communications device
    LPOVERLAPPED  lpOverlapped,            // address of overlapped structure
    LPDWORD  lpNumberOfBytesTransferred,          // address of actual bytes count
    BOOL  bWait  // wait flag
   );
```

# *EscapeCommFunction*

통신 자원에 확장 기능을 수행도록 지시한다.

```
BOOL EscapeCommFunction(
    HANDLE  hFile,          // handle to communications device
    DWORD  dwFunc        // extended function to perform
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

dwFunc

수행할 확장기능의 코드. 이 인수의 값은 다음 중 하나이다.

[반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## *GetCommConfig*

통신 장치의 현재 구성을 얻어온다.

```
BOOL GetCommConfig(
  HANDLE hCommDev, // handle of communications service
  LPCOMMCONFIG lpCC,            // address of comm.configuration structure
  LPDWORD lpdwSize    // address of size of buffer
 );
```

[인수]

hCommDev

열어 놓은 통신 장치에 대한 핸들

lpCC

값을 받아올 COMMCONFIG 구조체에 대한 포인터

```
typedef struct _COMM_CONFIG {
    DWORD dwSize;
    WORD  wVersion;
    WORD  wReserved;
    DCB   dcb;
    DWORD dwProviderSubType;
    DWORD dwProviderOffset;
    DWORD dwProviderSize;
    WCHAR wcProviderData[1];
} COMMCONFIG, *LPCOMMCONFIG;
```

lpdwSize

lpCC 가 지시하는 버퍼의 바이트 단위 크기로서 32 비트 변수에 대한 포인터.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

GetCommProperties 함수를 호출함으로써 장치의 구성 구조체의 최대 크기를 결정할 수 있다.

# *GetCommMask*

지정한 통신 자원에 대한 사건 마스크의 값을 읽어온다.

```
BOOL GetCommMask(
  HANDLE  hFile,            // handle of communications device
  LPDWORD  lpEvtMask // address of variable to get event mask
 );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpEvtMask

현재 가능 상태로 되어 있는 사건들에 대한 마스크 값을 갖는 32 비트 변수의 포인터

다음 값들 중 하나 이상의 값을 마스크한 값이다.

[반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## *GetCommModemStatus*

모뎀의 제어(control) 레지스터 값을 읽어온다.

```
BOOL GetCommModemStatus(
    HANDLE  hFile,            // handle of communications device
    LPDWORD  lpModemStat         // address of control-register values
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpModemStat

모뎀의 현재 제어 레지스터 값을 갖는 32 비트 변수의 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

이 함수는 CTS, DSR, RLSD, 링 지시자 신호를 감시하기 위해 WaitCommEvent 함수를 사용할 때 유용하다. 이러한 신호 변화를 검출하기 위해 WaitCommEvent 함수를 사용하고 어떤 변화가 일어났는지 그 변화된 상태를 알아내기 위해 GetCommModemStatus 함수를 이용한다.

# *GetCommProperties*

지정한 통신 자원의 특성에 대한 정보를 버퍼에 채운다.

```
BOOL GetCommProperties(
   HANDLE  hFile,          // handle of communications device
   LPCOMMPROP  lpCommProp  //  address  of  communications  properties
structure
  );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpCommProp

통신 장치에 대한 특성 정보가 반환되는 COMMPROP 구조체에 대한 포인터. 여기서 얻은 정보는 SetCommStat, SetCommTimeouts, SetupComm 함수를 이용해서 통신 자원 구성에 적용한다.

```
typedef struct _COMMPROP { // cmmp
   WORD  wPacketLength;      // packet size, in bytes
   WORD  wPacketVersion;     // packet version
   DWORD dwServiceMask;      // services implemented
   DWORD dwReserved1;        // reserved
   DWORD dwMaxTxQueue;       // max Tx bufsize, in bytes
   DWORD dwMaxRxQueue;       // max Rx bufsize, in bytes
   DWORD dwMaxBaud;          // max baud rate, in bps
   DWORD dwProvSubType;      // specific provider type
```

```
    DWORD dwProvCapabilities;  // capabilities supported
    DWORD dwSettableParams;    // changable parameters
    DWORD dwSettableBaud;      // allowable baud rates
    WORD  wSettableData;       // allowable byte sizes
    WORD  wSettableStopParity; // stop bits/parity allowed
    DWORD dwCurrentTxQueue;    // Tx buffer size, in bytes
    DWORD dwCurrentRxQueue;    // Rx buffer size, in bytes
    DWORD dwProvSpec1;         // provider-specific data
    DWORD dwProvSpec2;         // provider-specific data
    WCHAR wcProvChar[1];       // provider-specific data
} COMMPROP;
```

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *GetCommState*

지정한 통신장치의 현재 제어 상태를 장치 제어 블록 (device-control-block: DCB) 구조체에 넣어준다.

```
BOOL GetCommState(
    HANDLE  hFile,          // handle of communications device
    LPDCB  lpDCB            // address of device-control block structure
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpDCB

제어 설정 정보가 반환될 DCB 구조체에 대한 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *GetCommTimeouts*

지정된 통신 장치상의 읽기 쓰기 작업에 대한 시간초과(time-out)값을 읽어온다.

```
BOOL GetCommTimeouts(
  HANDLE  hFile,           // handle of communications device
  LPCOMMTIMEOUTS  lpCommTimeouts // address of comm. time-outs structure
 );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpCommTimeouts

시간초과 정보가 반환될 COMMTIMEOUTS 구조체의 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *PurgeComm*

지정한 통신자원의 입력 출력 버퍼에 있는 모든 문자들을 버린다. 또한 통신 자원에 걸려있는 읽기 쓰기 작업을 모두 중단한다.

```
BOOL PurgeComm(
   HANDLE  hFile,          // handle of communications resource
   DWORD  dwFlags       // action to perform
  );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

dwFlags

취할 동작. 다음 값들의 조합으로 이루어 진다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

이 함수는 출력버퍼를 비울 때 출력 버퍼에 남아있는 문자들을 지워버린다. 출력버퍼에 남아있는 문자들을 지우지 않고 모두 전송되도록 하려면, FlushFileBuffers 함수를 이용한다. 이 함수는 동기적으로 동작하며, 흐름 제어에 영향을 받는다. 그리고 모든 걸려 있는 쓰기 작업이 완료되기 전까지는 반환되지 않는다.

# *SetCommBreak*

지정된 통신 장치에 대해 전송을 일시 중단시킨다. ClearCommBreak 함수가 호출되기 전까지는 계속 정지(break) 상태에 있게 된다.

```
BOOL SetCommBreak(
```

```
  HANDLE  hFile             // handle of communications device
);
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *SetCommConfig*

통신 자원의 현재 상태를 설정한다.

```
BOOL SetCommConfig(
  HANDLE hCommDev, // handle of communications device
  LPCOMMCONFIG lpCC,            // address of comm. configuration services
  DWORD dwSize          // size of structure
);
```

## [인수]

hCommDev

열어 놓은 통신 장치에 대한 핸들

lpCC

COMMCONFIG 구조체에 대한 포인터

dwSize

lpCC 가 지시하는 구조체의 바이트 단위 크기

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *SetCommMask*

통신장치에 대해 감시하고자 하는 사건을 설정한다.

```
BOOL SetCommMask(
    HANDLE  hFile,           // handle of communications device
    DWORD  dwEvtMask   // mask that identifies enabled events
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

dwEvtMask

감시하고자 하는 사건을 지정한다. 0 은 아무 사건도  감시하지 않겠다는 뜻이다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

중첩 작업이 대기 중이 통신 자원에 대해 이 함수를 호출하면 WaitCommEvent 함수는 오류를 반환한다.

# *SetCommState*

장치 제어 블록(device-control block:DCB)에 설정한 대로 통신 장치를 구성한다. 이 함수는 모든 하드웨어와 제어 설정을 재 초기화하지만 입력 출력 큐를 비우지는 않는다.

```
BOOL SetCommState(
   HANDLE  hFile,          // handle of communications device
   LPDCB  lpDCB            // address of device-control block structure
  );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpDCB

지정된 통신 장치에 대한 구성 정보를 포함하는 DCB 구조체에 대한 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호

출하면 된다.

## [설명]

DCB 구조체의 XonChar 와 XoffChar 가 같은 값으로 설정되어 있으면 SetCommState 함수는 실패한다.

이 함수로 8250 을 설정하려면 DCB 구조체의 ByteSize 와 StopBits 변수에 대해 다음과 같은 제약이 따른다.

데이터 비트는 5 비트에서 8 비트여야 한다.

데이터 비트를 5 비트에 정지비트 2 비트는 사용할 수 없다. 데이터 비트 6,7,8 비트에 정지비트 1.5 비트도 사용할 수 없다.

# *SetCommTimeouts*

지정된 통신 자원에 대한 모든 읽기 쓰기 작업에 대해 시간 초과 인수를 설정한다.

```
BOOL SetCommTimeouts(
  HANDLE  hFile,           // handle of communications device
  LPCOMMTIMEOUTS  lpCommTimeouts //  address  of  communications  time-out
structure
);
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpCommTimeouts

설정할 시간초과 정보를 갖는 COMMTIMEOUTS 구조체의 포인터

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

# *SetupComm*

지정한 통신 장치를 초기화 한다.

```
BOOL SetupComm(
  HANDLE  hFile,          // handle of communications device
  DWORD  dwInQueue, // size of input buffer
  DWORD  dwOutQueue           // size of output buffer
 );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

dwInQueue

통신 자원의 내부 입력 버퍼에 대한 바이트 단위 크기

dwOutQueue

통신 자원의 내부 출력 버퍼에 대한 바이트 단위 크기

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

이 함수를 이용해서 통신 자원을 초기화하지 않으면 다른 통신 함수가 처음 호출되는 순간에 통신 자원은 시스템의 기본값으로 설정된다.

dwInQueue, dwOutQueue 는 내부 버퍼의 크기를 나타낸다. 예를 들어, YMODEM 프로토콜의 패킷크기는 1024 바이트 보다 약간 크다. 따라서 YMODEM 에 대해서는 버퍼의 크기를 1024 정도로 해야 한다.

# *TransmitCommChar*

지정한 통신 장치의 출력 버퍼에 대기 중인 어떤 데이터보다 먼저 문자를 전송한다.

```
BOOL TransmitCommChar(
    HANDLE  hFile,            // handle of communications device
    char  cChar    // character to transmit
   );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

cChar

전송될 문자.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

CTRL+C 같은 인터럽트 문자를 보내는데 유용하다.

이 함수를 이용해서 출력 버퍼로 전달된 문자가 전송되기 전에는 이 함수가 다시 호출될 수 없다. 만약 이전에 문자가 아직 전송되지 않은 상태에서 이 함수를 이용해서 다시 문자를 보

내려 하면 오류가 발생한다. 따라서 이 함수를 이용해서 출력버퍼로 한 문자를 보내고 나서 다시 새로운 문자를 보내려면 반드시 이전에 이 함수로 전달한 문자는 미리 전송되어야 한다.

문자 전송은 일반적인 흐름 제어와 핸드쉐이킹에 따른다. 이 함수는 동기적으로만 동작한다.

# *WaitCommEvent*

지정한 통신 장치에 대해 일어나는 사건을 기다린다.

```
BOOL WaitCommEvent(
  HANDLE  hFile,           // handle of communications device
  LPDWORD  lpEvtMask,           // address of variable for event that occurred
  LPOVERLAPPED  lpOverlapped,           // address of overlapped structure
 );
```

## [인수]

hFile

통신 장치에 대한 핸들. CreateFile 함수가 이 핸들값을 반환한다.

lpEvtMask

발생한 사건을 나타내는 32 비트 변수. 오류가 발생했다면 값은 0 이다.

lpOverlapped

OVERLAPPED 구조체에 대한 포인터. 이 인수는 hFile 이 FILE_FLAG_OVERLAPPED 플래그로 열리지 않았으면 무시된다. 중첩 작업을 원하지 않으면 NULL 로 한다.

## [반환값]

성공하면 TRUE, 실패하면 FALSE 를 반환한다. 오류의 종류를 알려면 GetLastError 를 호출하면 된다.

## [설명]

lpOverlapped 가 NULL 이거나 hFile 핸들이 FILE_FLAG_OVERLAPPED 플래그를 지정하지 않고 열었다면, WaitCommEvent 는 지정한 사건이 발생하거나 오류가 발행하기 전까지는 반환되지 않는다.

lpOverlapped 가 OVERLAPPED 구조체에 대한 포인터이고 hFile 이 FLLE_FLAG_OVERLAPPED 를 플래그로 해서 열렸다면, WaitCommEvent 는 중첩동작으로 수행된다. 이 경우, OVERLAPPED 구조체는 수동 떨기(manual-reset)사건 객체의 핸들(CreateEvent 함수를 이용해서 만든다)을 포함해야 한다.

중첩작업이 즉시 완료되지 않고, 함수가 FALSE 를 반환하고 GetLastError 함수가 ERROR_IO_PENDING 을 반환하면, 중첩작업이 배경으로 이루어지고 있다는 뜻이 된다. 이렇게 되면, 시스템은 WaitCommEvent 함수를 마치기 전에 OVERLAPPED 구조체의 hEvent 변수를 비신호(non-signaled)상태로 설정한다. 지정한 사건 중 하나가 발생하거나 오류가 발생하면 다시 신호(signaled)상태로 된다. 사건 객체의 상태를 알아내기 위해 대기 함수를 사용하고, 대기 함수 WaitCommEvent 의 결과를 알아내려면 GetOverlappedResult 함수를 이용한다. GetOverlappedResult 함수는 동작의 성공 실패 여부를 돌려주고 lpEvtMask 가 가리키는 변수는 어떤 사건이 발생했는지를 알려준다.

중첩작업으로 WaitCommEvent 가 동작 중일 때 SetCommMask 로 장치 핸들의 사건 마스크가 변경하려고 하면 WaitCommEvent 함수는 즉시 반환되고, lpEvtMask 가 가리키는 변수는 0 으로 설정된다.