Capítulo

2

Comunicação Serial

Apostila adaptada a partir do texto:
Allen Denver, Serial Communications in Win32, Microsoft Windows Developer Support, 1995.

Autor: Constantino Seixas Filho
UFMG – Departamento de Engenharia Eletrônica

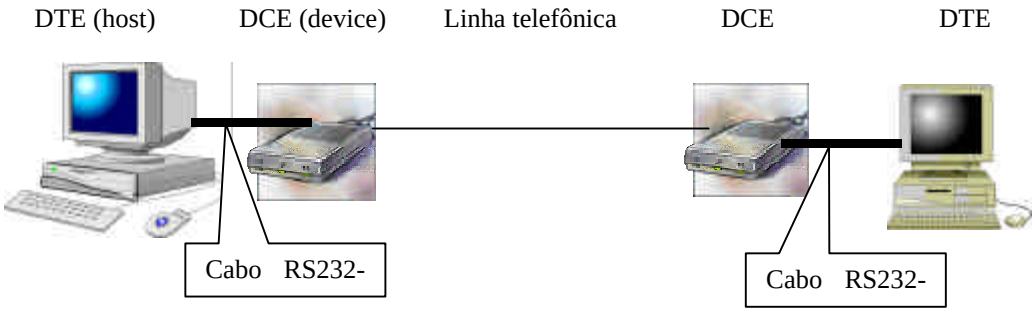
Comunicação Serial

O que será estudado:

- Sinais RS232-C
- Abrir uma porta
- Ler e escrever (com e sem sobreposição)
- Status serial (eventos e Erros)
- Configuração dos canais seriais

Sinais RS232-D:

DCE – *Data communication Equipment* = Modem
DTE – *Data Terminal Equipment* = Computador



Conector 25 pinos num dispositivo DTE			Origem	Destino
1		Terra de proteção		
2	TxD	Dado Transmitido	DTE	DCE
3	RxD	Dado Recebido	DCE	DTE
4	RTS	*Request to Send*	DTE	DCE
5	CTS	*Clear to Send*	DCE	DTE
6	DSR	*DCE Ready*	DCE	DTE
7		Terra de sinal		
8	CD	*Rec Line Signal Detect*	DCE	DTE
20	DTR	*DTE Ready*	DTE	DCE
22	RI	*Ring Indicator*	DCE	DTE

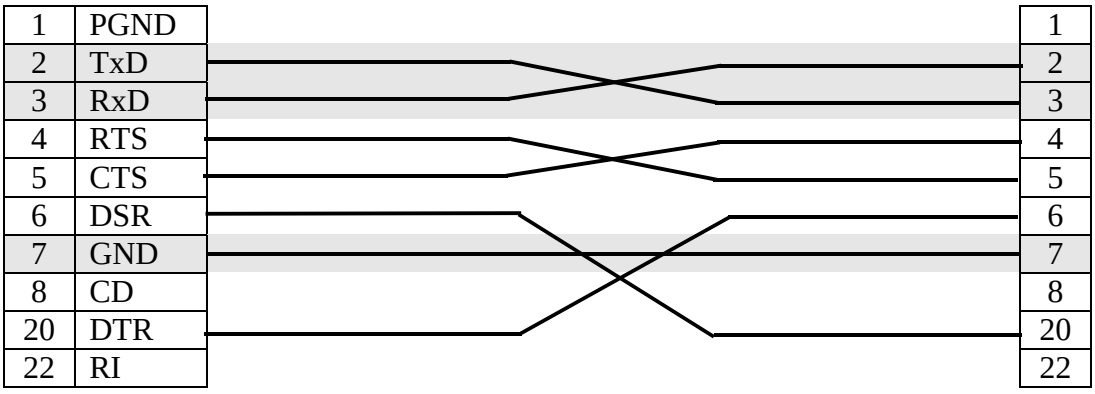
Conector 9 pinos num dispositivo DTE			Origem	Destino
1	CD	*Rec Line Signal Detect*	DCE	DTE
2	RxD	Dado Recebido	DCE	DTE
3	TxD	Dado Transmitido	DTE	DCE
4	DTR	DTE Ready	DTE	DCE
5		Terra de sinal		
6	DSR	*DCE Ready*	DCE	DTE
7	RTS	*Request to Send*	DTE	DCE

8	CTS	*Clear to Send*	DCE	DTE
9	RI	*Ring Indicator*	DCE	DTE

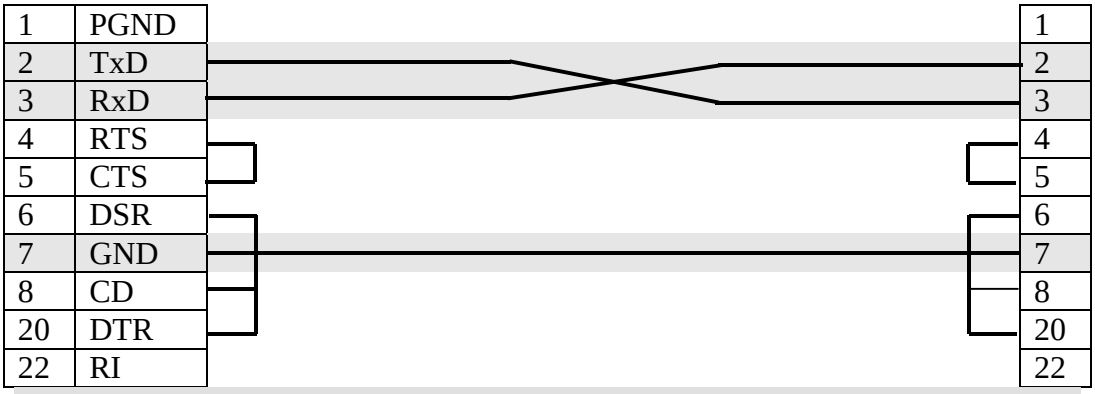
Principais sinais de controle (computador conectado ao modem)

Sinal	Comentário
DTR – Data Terminal Ready	Usado para avisar ao modem que o computador está alimnetado e pronto para comunicar
RTS – Request To Send	Usado para avisar o modem que o computador deseja enviar dados agora.
DSR – Data Set Ready	Usado para avisar ao computador que o modem está alimentado e pronto para operar.
CTS – Clear To Send	Usado para avisar ao computador que o modem está pronto para aceitar transmissão de dados.
CD – Data Carrier Detect	Usado para avisar so computador que o modem estabeleceu uma conexão com o modem remoto na outro extremo da linha.
RI – Ring Indicator	Usado para avisar o computador que a linha conectada ao modem está chamando.

Cabo Normal:



Cabo NullModem:



DICA:

Muito cuidado ao interligar dois dispositivos através de um cabo NULL Modem RS232-C. Antes meça a diferença de potencial AC e DC entre os pinos 7 (terra de sinal) dos conectores.

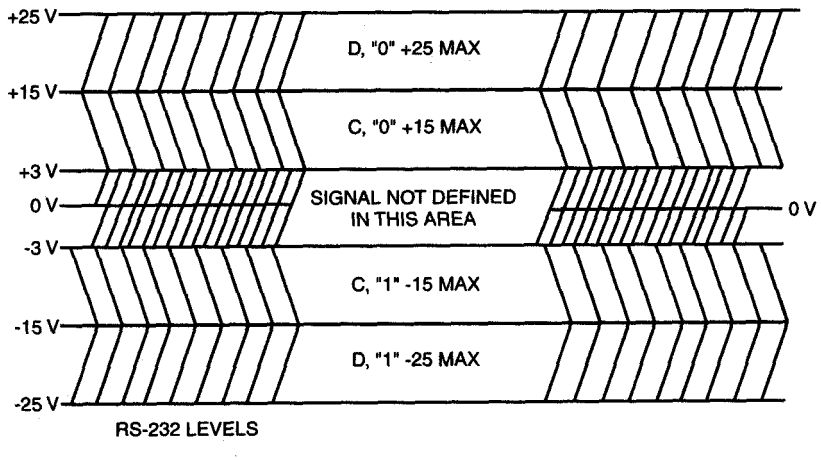


Figura 1: **níveis dos sinais RS232-D**

Comprimento do cabo

Pela norma, o comprimento máximo do cabo seria de cerca de 12 metros, mas distâncias muito maiores podem ser conseguidas com um cabo blindado de boa qualidade:

Baud Rate	Comprimento do cabo blindado	Comprimento de cabo não blindado
110	5000	1000
300	4000	1000
1200	3000	500
2400	2000	500
4800	500	250
9600	250	100

Tabela 1: Comprimento do cabo e velocidade da interface

Abrindo uma porta serial:

CreateFile Geral:

HANDLE CreateFile(

```
LPCTSTR lpFileName,                          // Nome do arquivo
DWORD dwDesiredAccess,                       // Tipo de acesso.
DWORD dwShareMode,                           // Compartilhamento
LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // Atributos de
                                                segurança
DWORD dwCreationDistribution,                // Criação ou abertura
DWORD dwFlagsAndAttributes,                  // Atributos e Flags
HANDLE hTemplateFile                         // Gabarito de atributos
);
```

Comentários sobre os parâmetros:

lpFileName	Nome do arquivo a ser aberto ou criado
dwDesiredAccess	GENERIC_READ: abre apenas para leitura GENERIC WRITE: abre apenas para escrita GERNERIC_READ|GENERIC_WRITE: abre para leitura e escrita
dwShareMode	Indica como o arquivo será compartilhado: 0: Nenhuma outra abertura pode ser efetuada FILE_SHARE_READ: só aberturas para leitura serão permitidas FILE_SHARE_WRITE: só aberturas para escrita serão permitidas FILE_SHARE_READ|FILE_SHARE_WRITE: quaisquer operações de aberturas serão permitidas
lpSecurityAttributes	Apontador para descritor de segurança para o handle do novo arquivo (contém a ACL do novo handle). Se NULL, será utilizada estrutura default e o handle para o arquivo será herdável.
dwCreationDistribution	CREATE_NEW: Tenta ciar um novo arquivo. Retorna erro se já existir CREATE_ALWAYS: Cria novo arquivo em qualquer situação. OPEN_EXISTING: Abre arquivo. Se não existir retorna erro. OPEN_ALWAYS: Tenta abrir um arquivo. Se não existir é criado. TRUNCATE_EXISTING: Abre arquivo existente truncando tamanho para 0 bytes. Se não existir: erro.
dwFlagsAndAttributes	Atributos: Usados na criação do arquivo: FILE_ATTRIBUTE_HIDDEN: Arquivo oculto. Não aparece em diretórios. FILE_ATTRIBUTE_SYSTEM: O arquivo é usado

	apenas pelo S.O. FILE_ATTRIBUTE_READONLY: Só pode ser lido. FILE_ATTRIBUTE_TEMPORARY: arquivo será mantido na memória. FILE_ATTRIBUTE_NORMAL: deve ser usado sozinho. Flags: Alteram o modo como as operações de escrita e leitura são realizadas: FILE_FLAG_RANDOM_ACCESS: otimiza acesso randômico. FILE_FLAG_SEQUENTIAL_SCAN: otimiza acesso seqüencial. FILE_FLAG_DELETE_ON_CLOSE: O arquivo será deletado quando for fechado. **FILE_FLAG_OVERLAPPED**: habilita operações assíncronas sobre o arquivo.
hTemplateFile	Especifica handle para arquivo que será usado como gabarito de atributos ou NULL.

Retorno da função:

Status	**Interpretação**
Handle para o Arquivo criado	Sucesso
INVALID_HANDLE_VALUE	Falha

Após utilizar o arquivo, o seu handle deve ser fechado com a função *CloseHandle()*. Se a flag FILE_FLAG_DELETE_ON_CLOSE tiver sido ativada na criação, o arquivo será deletado.

CreateFile Porta Serial:

```
HANDLE hComm;
char *gszPort = "COM1";

hComm = CreateFile( gszPort,   // lpFileName
        GENERIC_READ | GENERIC_WRITE,
        0,                     // dwDesiredAccess
        0,                     // dwShareMode deve ser 0
        OPEN_EXISTING,  // dwCreationDistribution
        FILE_FLAG_OVERLAPPED,  // NULL: operação sem sobreposição
        0);                    // hTemplateFile deve ser 0

if (hComm == INVALID_HANDLE_VALUE)  //  erro abrindo porta: aborta

    * * *
```

O Win32 não possui um mecanismo para determinar que portas estão instaladas.

Configuração da comunicação serial

O primeiro passo para se comunicar numa linha serial, logo após o canal ter sido aberto com o comando CreateFile é definir os parâmetros da comunicação serial compreendendo a velocidade, número de stop bits, tipo de paridade, etc. A estrutura DCB (*Device Control Block*) é utilizada para definir todos os parâmetros de comunicação.

Como incializar a estrutura DCB ?

Método 1: GetCommState

```
DCB dcb = {0};

If (!GetCommState(hComm, &dcb))
        // Erro na leitura de DCB
else
        // DCB está pronto para uso
```

Método 2: BuildCommDCB.

```
DCB dcb;

FillMemory(&dcb, sizeo(dcb), 0);
Dcb.DCBlength = sizeof(dcb);
If (!BuildCommDCB("9600,n,8,1", &dcb))  {
        // Não conseguiu definir DCB.
        // Possivelmente um erro no string de especificação da comunicação.
        Return FALSE;
}
else
        // DCB está pronto para uso
```

Método 3: Definir membros da estrutura DCB manualmente

Desvantagem: mudanças internas do Windows afetam esta solução.

```
DCB dcb;
FillMemory(&dcb, sizeof(dcb), 0);
If (!GetCommState(hComm, &dcb))              // busca DCB corrente
// Erro em GetCommState
return FALSE;

// Atualiza estrutura:
dcb.BaudRate = CBR_9600;

// Define novo estado
if (!SetCommState(hComm, &dcb))
```

```
// Erro em SetCommState. Possivelmente um problema com o handle da porta de
// comunicação ou um problema com a própria estrutura DCB.
```

GetCommState

```
BOOL GetCommState (

HANDLE hFile,          // Handle para dispositivo de comunicação
LPDCB lpDCB            // Apontador para a estrutura do tipo DCB, que receberá
                          a configuração corrente do canal serial.

);
```

Retorno da função:

Status	Interpretação
<> 0	Sucesso
0	Falha. Use *GetLastError()* para descobrir a falha.

SetCommState

```
BOOL SetCommState (

HANDLE hFile,          // Handle para dispositivo de comunicação
LPDCB lpDCB            // Apontador para a estrutura do tipo DCB, que contém a
                          configuração desejada para o canal serial.

);
```

Retorno da função:

Status	Interpretação
<> 0	Sucesso
0	Falha. Use *GetLastError()* para descobrir a falha.

Estrutura DCB

```
typedef struct _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 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;
```

BaudRate

Pode assumir um dos valores abaixo:

CBR_110	CBR_19200
CBR_300	CBR_38400
CBR_600	CBR_56000
CBR_1200	CBR_57600
CBR_2400	CBR_115200
CBR_4800	CBR_128000
CBR_9600	CBR_256000
CBR_14400	

Paridade

Pode assumir um dos valores abaixo:

EVENPARITY	MARKPARITY
NOPARITY	ODDPARITY

StopBits

Pode assumir um dos valores abaixo:

ONESTOPBIT
ONE5STOPBITS
TWOSYTOPBITS

Exemplo completo:
Definindo parâmetros de uma porta serial

```
#include <windows.h>

int main(int argc, char *argv[])
{
  DCB dcb;
  HANDLE hCom;
  BOOL fSuccess;
  char *pcCommPort = "COM2";

  hCom = CreateFile(
                pcCommPort,
                GENERIC_READ | GENERIC_WRITE,
                0,        // dispositivos comm abertos com acesso exclusivo
                NULL, // sem atributos de segurança
                OPEN_EXISTING, // deve usar OPEN_EXISTING
                0,        // I/O sem ovelap
                NULL  // hTemplate deve ser NULL para comm
        );

  if (hCom == INVALID_HANDLE_VALUE) {
    // Tata o erro
    printf ("CreateFile falhou com o erro %d.\n", GetLastError());
    return (1);
  }

// Vamos mudar a configuração corrente e saltar a definição de do tamanho dos buffers de
// entrada e saída com SetupComm.
fSuccess = GetCommState(hCom, &dcb);

if (!fSuccess) {
  // Trata o erro
  printf ("GetCommState falhou com erro %d.\n", GetLastError());
  return (2);
}

// Preenche DCB: baud=57,600 bps, 8 bits de dados, sem paridade, 1 stop bit

  dcb.BaudRate = CBR_57600;             // define o baud rate
  dcb.ByteSize = 8;                     // data size, xmit, and rcv
  dcb.Parity = NOPARITY;                // sem paridade
  dcb.StopBits = ONESTOPBIT;            // um stop bit

  fSuccess = SetCommState(hCom, &dcb);

  if (!fSuccess) { // Trata o erro
      printf ("SetCommState falhou com erro %d.\n", GetLastError());
      return (3);
  }

  printf ("Porta serial %s configurada com sucesso.\n", pcCommPort);
  return (0);
}
```

Leitura e escrita:

Podem ser:

- Sem sobreposição (*Nonoverlapped*)
  Leitura e escrita são síncronas. A aplicação escreve e aguarda a operação ser completada.
- Com sobreposição (*Overlapped*)
  Várias operações simultâneas podem ser iniciadas.
  Utiliza estrutura:

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

Exemplo: Leitura com overlap

```
#define READ_TIMEOUT 500 // milliseconds

DWORD dwRead;       // Número de bytes lidos
BOOL fWaitingOnRead = FALSE;        // Flag: leitura pendente
OVERLAPPED osReader = {0};          // Estrutura OVERLAPPED
DWORD dwRes;

// Cria evento com reset manual no estado não sinalizado
osReader.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL};
if (osReader.hEvent == NULL)
       // Erro de criação do Evento: aborta

if (!fWaitingOnRead) { // Se não há leitura pendente ...
   // Realiza operação de leitura
   if (!ReadFile ( hComm,
                   lpBuf,
                   READ_BUF_SIZE,
                   &dwRead,
                   &osReader)) // lpOverlapped
     {
        if (GetLastError()!= ERROR_IO_PENDING) // Leitura não foi enfileirada
                   // Erro de comunicação: relate-o
             else
                   fWaitingOnRead = TRUE;  // ativa flag leitura pendente
       }
else // leitura foi completada imediatamente
```

```
        TrateLeituraComSucesso(lpBuf, dwRead) ;
}

if (fWaitingOnRead) {  // Existe uma leitura pendente: espere-a
    dwRes = WaitForSingleObject(osReader.hEvent, READ_TIMEOUT);
    switch (dwRes)
    {
        // Leitura completada
        case WAIT OBJECT 0:
                if (!GetOverlappedResult(hComm, &osReader, &dwRead, FALSE) )
                        // Erro de comunicação: relate-o
                else
                        // Leitura completada com sucesso
                        TrateLeituraComSucesso(lpBuf, dwRead);
                // Reseta flag para que outra operação possa ser iniciada
                fWaitingOnRead = FALSE;
                break;
        case WAIT TIMEOUT:
                // Operação ainda não foi completada. Pode-se reemitir o comando e
                // continuar esperando ou cancelá-lo.
                // Este é um bom momento para realizar alguma outra atividade.
                break;
        default:
                // Erro em WaitForSingleObject; aborte.
                // Isto indica algum problema com o handle do evento na estrutura
                // OVERLAPPED.
                break;
    }
}

CloseHandle(osReaderhEvent);
```

Exemplo: Escrita com overlap

```
BOOL WriteABuffer(char *lpBuf, DWORD dwToWrite)
{
    OVERLAPPED osWrite = {0};
    DWORD dwWritten;     // Número de bytes escritos
    DWORD dwRes;         // Status da operação WaitForSingleObject
    BOOL fRes;           // Valor de retorno da função: TRUE=SUCESSO

    // Cria evento e armazena  handle na estrutura OVERLAPPED
    osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (osWrite.hEvent == NULL) // Erro na criação do objeto evento
        return FALSE;

    // Emite operação de escrita
    if (!WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, &osWrite) ) {
      if (GetLastError() != ERROR IO PENDING) {
        // WriteFile falhou, mas não está atrasado: erro e aborta.
        fRes = FALSE;
      }
      else {// Operação de Escrita está pendente. Espera conclusão para sempre…
          dwRes = WaitForSingleObject(osWrite.hEvent, INFINITE);
          switch(dwRes)
          {
              // Evento associado à estrutura OVERLAPPED foi sinalizado
              case WAIT OBJECT 0:
                  if  (!GetOverlappedResult(hComm,  &osWrite,  &dwWritten,
                  FALSE))
                      fRes = FALSE;
                  else // Operação concluída com sucesso
                  fRes = TRUE;
                  break;
              default:
                  // Erro em WaitForSingleObject; aborte.
                  // Isto indica algum problema com o handle do evento na
                  // estrutura  OVERLAPPED.
                  fRes = FALSE;
                  break;
          }  // switch
       }  // else
    }  // if
    else
       // Operação de escrita em arquivo completou imediatamente
       fRes = TRUE;

    CloseHandle(osWrite.hEvent);
    return fRes;
} // WriteABuffer
```

O trecho abaixo utiliza *GetOverlappedResult()* para esperar pela conclusão da operação ao invés de *WaitForSingleObject()*. Qual diretiva é a melhor ?

```
BOOL WriteABuffer(char * lpBuf, DWORD dwToWrite)
{
   OVERLAPPED osWrite = {0};
   DWORD dwWritten;
   BOOL fRes;

   // Cria Evento e associa à estrutura OVERLAPPED
   osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
   if (osWrite.hEvent == NULL)
        // Erro na criação do evento
        return FALSE;

   // Emite operação de escrita
   if (!WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, &bsWrite) ) {
        if (GetLastError() != ERROR_IO_ PENDING) {
             // Escrita falhou, mas não está atrasada. Reporta erro e aborta.
             fRes = FALSE;
        }
        else {
            // Escrita está pendente
            if (!GetOverlappedResult(hComm, &osWrite, &dwWritten, TRUE})
                 fRes = FALSE;
            else
                 // Operação de escrita completada com sucesso
                 fRes = TRUE;
        }
   }
   else
        // Operação de escrita completou imediatamente
        fRes = TRUE;

   CloseHandle(osWrite.hEvent};
   return fRes;
}
```

Para evitar espera infinita por final de transmissão:

- Colocar o código de espera pela conclusão da operação em uma thread à parte. Completion ports são melhores para esta finalidade. Entretanto as operações não concluídas devem ser excluídas de toda forma, após certo tempo de espera.
- Use COMMTIMEOUTS para forçar a operação de escrita a se completar (será estudado na página 26).
- Incluir valor de *timeout* em WaitForSingleObject. Complica o gerenciamento das operações não completadas e força a necessidade de alocação dinâmica da estrutura OVERLAPPED.

Serial Status

Ler o status de uma porta de comunicação:

1. Por evento:
   - Definir uma máscara de eventos (*SetCommMask*) que causa uma notificação a ser enviada quando o evento ocorre.
   - Uma thread deve ficar à espera das notificações (*WaitCommEvent*).
   - *GetCommModemStatus()* pode ser usada para determinar o valor corrente das linhas de controle de modem.
2. Por polling
   - Ler o status das portas periodicamente (estrutura *COMSTAT* em *ClearCommError*).

SetCommMask:

BOOL SetCommMask (

```
HANDLE hFile,        // Handle para dispositivo retornado por CreateFile
DWORD dwEvtMask,     // Especifica evento a ser habilitado.
                        0 desabilita todos os eventos (ver relação a seguir)
);
```

Flag de Evento	Descrição
EV_BREAK	Um break foi detectado na entrada. O sinal break é um sinal especial que ocorre fora do quadro de dados. O caracter break ocorre quando a linha muda do estado mark (tensão positiva) para space (tensão negativa) e é mantido lá por um período superior à duração de um caracter. O sinal de break é semelhante ao caracter ASCII NUL (um string de 0s), mas existe em um nível mais baixo do que a codificação ASCII que governa a codificação de informação dentro do quadro de caracter.
EV_CTS	O sinal CTS mudou de estado. Para ler o valor corrente do sinal use *GetCommModemStatus()*.
EV_DSR	O sinal DSR mudou de estado. Para ler o valor corrente do sinal use *GetCommModemStatus()*.
EV_ERR	Ocorreu erro de transmissão. Os erros podem ser: CE_OVERRUN, CE_FRAME, CE_RXPARITY. Para descobrir a causa do erro use *ClearCommError()*.
EV_RING	O sinal ring indicator foi detectado.
EV_RLSD	O sinal RLSD mudou de estado. Para ler o valor corrente do sinal use *GetCommModemStatus()*. RLSD= Receive Line Signal Detect (RS232-D) ~ CD = Carrier Detect (RS2332-C).
EV_RXCHAR	Um novo caracter foi recebido e está no buffer de entrada.

EV_RXFLAG	O caracter de evento foi recebido e colocado no buffer de entrada.
EV_TXEMPTY	O último caracter a ser transmitido foi enviado.

Retorno da função:

Status	Interpretação
<> 0	Sucesso
0	Falha. Use *GetLastError()* para descobrir a falha.

WaitCommEvent:

```
BOOL WaitCommEvent(

HANDLE hFile,                   // Handle para dispositivo de comunicação
LPDWORD lpEvtMask,              // Ponteiro para variável de 32 bits que
                                   receberá máscara de eventos.
LPOVERLAPPED lpOverlapped,      // Apontador para estrutura overlapped.
);
```

Retorno da função:

Status	Interpretação
<> 0	Sucesso
0	Falha. Use *GetLastError()* para descobrir a falha.

Exemplo 1:

```
DWORD dwStoredFlags;
DwStoredFlags = EV_BREAK | EV_CTS | EV_DSR | EV_ERR | EV_RING | \
                EV_RLSD | EV_RXCHAR | EV_RXFLAG | EV_TXEMPTY;
If (!SetCommMask(hComm, dwStoredFlags))
// Erro ao definir máscara de comunicação
```

Exemplo 2:

```
DWORD dwCommEvent;

If (!SetCommMask(hComm, EV_RING)) // EV_RING não é notificado no W95
    // Erro ao definir máscara de comunicação
    return FALSE;

If (!WaitCommEvent(hComm, &dwCommEvent, NULL))
    // Erro ocorreu durante espera do evento
    return FALSE;
else // Evento ocorreu
     return TRUE;
```

Este código pode ficar bloqueado para sempre, se um evento nunca acontece.

Solução: Definir uma operação com overlap e esperar por um evento.

Exemplo – Espera de evento de comunicação com overlap

```
#define STATUS_CHECK_TIMEOUT   500 // Milliseconds

DWORD   dwRes;                  // Retorno de WaitForSingleObject
DWORD   dwCommEvent;            // Evento de comunicação
DWORD   dwStoredFlags;          // flags a serem monitoradas
BOOL    fWaitingOnStat = FALSE; // flag leitura de status pendente
OVERLAPPED osStatus = {0};

dwStoredFlags = EV_BREAK | EV_CTS | EV_DSR | EV_ERR | EV_RING |\
                EV_RLSD | EV_RXCHAR | EV_RXFLAG | EV_TXEMPTY;
if (!SetCommMask(hComm, dwStoredFlags))
    // erro definindo mascara de comunicação: aborta
    return 0;

osStatus.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
if (osStatus.hEvent == NULL)
    // erro na criação de um evento: aborta
    return 0;

for ( ; ; ) {
// Emite um comando de espera de evento de mudança de status
if (!fWaitingOnStat) {
    if (!WaitCommEvent(hComm, &dwCommEvent, &osStatus) ) {
        if (GetLastError() == ERROR_IO_PENDING)
            bWaitingOnStatusHandle = TRUE;
        else
            // erro na rotina WaitCommEvent: aborta
            break;
    }
    else
        // WaitCommEvent retornou imediatamente
        // Trata evento de mudança de status
        ReportStatusEvent(dwCommEvent);
}  // if

// Verifica operação com sobreposição
if   (fWaitingOnStat) {
// Espera pela ocorrência de um evento
dwRes = WaitForSingleObject(osStatus.hEvent, STATUS_CHECK_TIMEOUT);
switch (dwRes)
{
   case WAIT OBJECT 0:    // Evento ocorreu
        if (!GetOverlappedResult(hComm, &osStatus, &dwOvRes, FALSE))
           // Ocorreu um erro na operação com sobreposição
           // call GetLastError para determinar o erro e aborta se for fatal
```

```
            else
                // O evento de status é armazenado na flag de evento especificada na
                // chamada a WaitCommEvent().
                // Trate o evento de status convenientemente.
                ReportStatusEvent(dwCommEvent);

            // Ativa flag fWaitingOnStat para indicar que um novo comando
            // WaitCommEvent pode ser emitido
            fWaitingOnStat = FALSE;
            break;

        case WAIT TIMEOUT:
            // Operação ainda não completou. A flag WaitingOnStatusHandle
            // não mudou. Outro comando WaitCommEvent não pode ser emitido
            // até o primeiro comando terminar.
            //
            // Esta é uma boa oportunidade de fazer algum trabalho extra.
            RealizaAtividadeExtra();
            break;

        default:
            // Erro em WaitForSingleObject; aborta
            // Isto indica um problema com o handle para o Evento na
            // estrutura OVERLAPPED
            CloseHandle(osStatus.hEvent);
            return 0;
       } // switch
    } // if
} // for

CloseHandle(osStatus.hEvent);
```

Observações

Comunicação sem sobreposição (síncrona):

- *WaitCommEvent* ficará bloqueada até que um evento ocorra.
- Se uma segunda thread chamar *SetCommMask*, ela ficará bloqueada até que *WaitCommEvent* da primeira thread retorne.

Generalizando:

- Se uma thread estiver bloqueada em qualquer função de comunicação, e outra thread chama uma função de comunicação, a segunda thread fica bloqueada na função até que a função de comunicação retorne da primeira thread.

Comunicação com sobreposição (assíncrona):

Se *SetCommMask* define uma nova máscara de eventos qualquer operação de *WaitCommEvent* pendente irá ser completada com sucesso e a máscara de evento retornada será NULL.

Leitura de caracter após um evento de comunicação

```
DWORD dwCommEvent ;
DWORD dwRead;
char chRead;

if  (!SetCommMask(hComm, EV_RXCHAR))
    // Erro definindo máscara de evento de comunicação
for ( ; ; ) {
    if (WaitCommEvent(hComm, &dwCommEvent, NULL)) {
        if (ReadFile(hComm, &chRead, 1, &dwRead, NULL))
            // Um byte foi lido e deve ser processado
        else
            // Ocorreu um erro na chamada de ReadFile
            break;
    }
else
    // Erro em WaitCommEvent.
    break;
}
```

Problema:
Se mais de dois caracteres chegarem antes que uma leitura seja completada, haverá perda de caracteres.

Solução:
Ler das porta até que nenhum byte reste nos buffers.

Leitura de caracter após um evento de comunicação

```
DWORD dwCommEvent ;
DWORD dwRead;
char chRead;

if  (!SetCommMask(hComm, EV_RXCHAR))
    // Erro definindo mascara de evento de comunicação

for ( ; ; ) {
    if (WaitCommEvent(hComm, &dwCommEvent, NULL)) {
        do {
            if (ReadFile(hComm, &chRead, 1, &dwRead, NULL))
                // Um byte foi lido e deve ser processado
            else
                // Ocorreu um erro na chamada à ReadFile
                break;
        } while (dwRead);
    }
else
    // Erro em WaitCommEvent.
    break;
```

```
} // for
```

O código acima só funciona se os valores de TIMEOUT de comunicação forem definidos convenientemente. A definição de Timeout altera o funcionamento da função ReadFile fazendo com ela retorne se não existirem caracteres a serem lidos.

Gerenciamento de erros

Erros de comunicação causam a suspensão de todas as operações de I/O, até que a condição de erro seja removida.

ClearCommError() indica os erros ocorridos e limpa a condição de erro.

Valor	**Significado**
CE_BREAK	The hardware detected a break condition.
CE_DNS	**Windows 95/98:** A parallel device is not selected.
CE_FRAME	The hardware detected a framing error.
CE_IOE	An I/O error occurred during communications with the device.
CE_MODE	The requested mode is not supported, or the *hFile* parameter is invalid. If this value is specified, it is the only valid error.
CE_OOP	**Windows 95/98:** A parallel device signaled that it is out of paper.
CE_OVERRUN	A character-buffer overrun has occurred. The next character is lost.
CE_PTO	**Windows 95/98:** A time-out occurred on a parallel device.
CE_RXOVER	An input buffer overflow has occurred. There is either no room in the input buffer, or a character was received after the end-of-file (EOF) character.
CE_RXPARITY	The hardware detected a parity error.
CE_TXFULL	The application tried to transmit a character, but the output buffer was full.

C l e a r C o m m E r r o r

```
BOOL ClearCommError (

HANDLE hComme,          // Handle para dispositivo de comunicação
LPDWORD lpErrors,       // Ponteiro para variável de 32 bits que
                           receberá o código de erro
LPCOHSTAT lpStat       // Ponteiro para buffer para status de
                           comunicação

);
```

```
COMSTAT comStat;
DWORD dwErrors;
BOOL fOOP, fOVERRUN, fPTO, fRXOVER, fRXPARlTY, fTXFULL;
BOOL fBREAK, fDNS, fFRAME, flOE, fMODE;

// Leia e limpe todos os erros correntes na porta
if  (!ClearCommError(hComm, &dwErrors, &comStat))
      // Comunica erro em ClearCommError
      return;

// Leia flags de erros
fDNS = dwErrors & CE_DNS;
flOE = dwErrors & CE_IOE;
fOOP = dwErrors & CE_OOP;
fPTO = dwErrors & CE_PTO;
fMODE = dwErrors & CE_ MODE;
fBREAK = dwErrors & CE_BREAK;
fFRAME = dwErrors & CE_FRAME;
fRXOVER = dwErrors & CE_RXOVER;
fTXFULL = dwErrors & CE_TXFULL;
fOVERRUN = dwErrors & CE_OVERRUN;
fRXPARlTY = dwErrors & CE_RXPARlTY;

// A estrutura COMSTAT contém informação referente ao status de comunicação
if (comStat.fCtsHold)
       // Tx esperando por sinal CTS

if (comStat.fDsrHold)
       // Tx esperando por sinal DSR

if (comStat.fRlsdHold)
       // Tx esperando por sinal RLSD

if (comStat.fXoffHold)
       // Tx esperando, pela recepção do caracter XOFF

if (comStat.fXoffSent)
       // Tx esperando, pela recepção do caracter XOFF

if (comStat.fEof)
       // caracter EOF recebido

if (comStat.fTxim)
       //caracter esperando transmissão; char enfileirado com TransmitCommChar

if (comcbInQue)
       // comStat.cbInQue bytes foram recebidos, mas não lidos

if (comStat.cbOutQue)
       // comStat.cbOutQue bytes esperando por transferência
```

Leitura do status das linhas de Modem

GetCommModemStatus

```
BOOL GetCommModemStatus (
    HANDLE hFile,          // Handle para dispositivo de comunicação
    LPDWORD lpModemStat    // Ponteiro para variável de 32 bits que
                           //   receberá o status corrente das linhas de
                           //   controle de modem.
);
```

Valor	Significado
MS_CTS_ON	O sinal CTS (*clear-to-send*) está ativo.
MS_DSR_ON	O sinal DSR (*data-set-ready*) está ativo.
MS_RING_ON	O sinal *ring indicator* está ativo.
MS_RLSD_ON	O sinal RLSD (*receive-line-signal-detect*) está ativo.

Retorno da função:

Status	Interpretação
<> 0	Sucesso
0	Falha. Use *GetLastError()* para descobrir a falha.

Exemplo – Leitura de status do modem

```
DWORD dwModemStatus;
BOOL fCTS, fDSR, fRING, fRLSD;

If (!GetCommModemStatus(hComm, &dwModemStatus))
// Erro em GetCommModemStatus
return;

fCTS = MS_CTS_ON & dwModemStatus;
fDSR = MS_DSR_ON & dwModemStatus;
fRING = MS_RING_ON & dwModemStatus;
fROLSD = MS_RLSD_ON & dwModemStatus;

// Use as flags
```

Controle de fluxo

Mecanismo para suspender a comunicação enquanto um dos dispositivos está ocupado e não pode processar a comunicação.
O controle de fluxo é geralmente feito através dos sinais RTS e CTS. DTR e DSR são geralmente utilizados para confirmar que um dispositivo está conectado e ligado.
O drive realiza o controle das linhas de fluxo automaticamente.
Se uma aplicação deseja controlar o fluxo diretamente deve usar: *EscapeCommFunction()*.

Controle de fluxo: Software

É usado em protocolos baseados em caracter.
Utiliza um caracter para habilitar a transmissão (**XON**) e outro para desabilitá-la (**XOFF**).

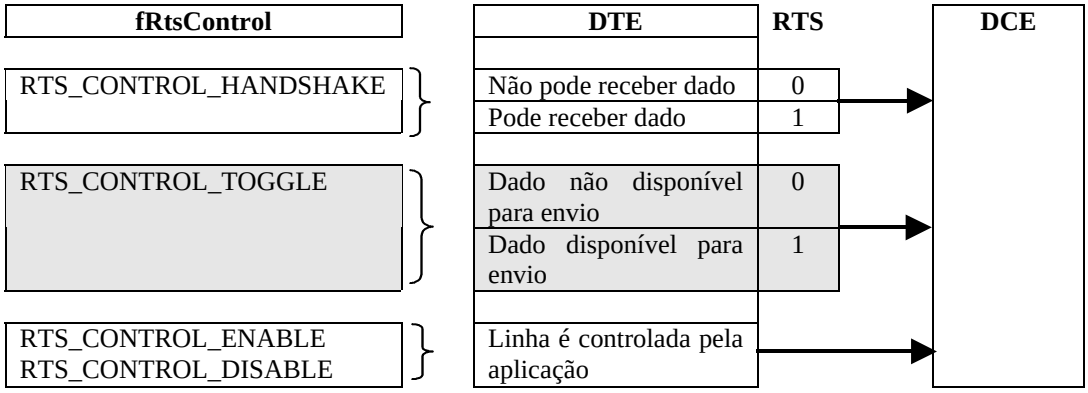
Para habilitar controle por software:
1. Ativar flags de habilitação: **fOutX** = **fInX** = 1;
2. Definir:
   XoffChar = caracter XOFF
   XonChar = caracter XON
   XoffLim indica a quantidade mínima de memória livre para disparar o envio de XOFF.
   XonLim indica a quantidade mínima de memória livre para disparar o envio de XON.
   FTXContinueOnXoff :
   TRUE: Transmissão continua após DTE ter emitido XOFF.
   FALSE: Transmissão é interrompida após DTE ter emitido XOFF.

   A recepção dos caracteres XON e XOFF causa o retorno da operação de leitura com 0 bytes.

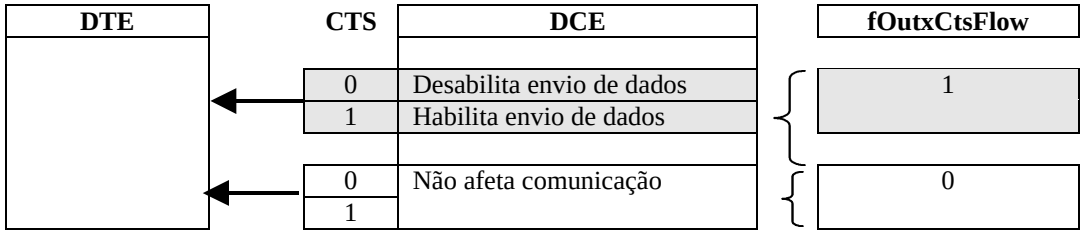
É realizado através dos sinais: RTS/CTS e DSR/DTR.
A programação afeta apenas o DTE (computador). O DCE deve ser programado à parte. Os dois dispositivos devem possuir a mesma programação.

RTS (*Ready To Send*) Controle de fluxo de saída Controlado pelo DTE	Se **fRtsControl** = RTS_CONTROL_HANDSHAKE, o controle de fluxo é automático e constitui erro se a aplicação altera o estado da linha RTS através de ***EscapeCommFunction***. Se o buffer de entrada tem espaço suficiente para receber dados (> ½buffer), o driver seta RTS. Se o espaço é pequeno (< ¼ buffer) o driver reseta RTS. Se **fRtsControl** = RTS_CONTROL_TOGGLE, o drive seta RTS para 1 quando há dado para ser enviado. O driver desativa RTS se não há dado para ser enviado. Se **fRtsControl** = RTS_CONTROL_ENABLE ou RTS_CONTROL_DISABLE, o usuário é livre para ativar o estado da linha RTS, segundo sua conveniência. O DCE irá suspender a transmissão se a linha for para 0. O DCE irá continuar a transmissão se a linha for 1.
CTS (*Clear To Send*) Controlado pelo DCE	**fOutxCtsFlow** deve ser ativado. DCE seta linha se pode receber dados. DCE reseta linha se não pode receber.
DSR (*Data Set Ready*) Controle de fluxo de saída	**fOutxDsrFlow** deve ser ativado. DCE seta linha se pode receber dados. DCE reseta linha se não pode receber.
DSR (*Data Set Ready*) Controle de fluxo de entrada	**fDsrSensitivity** deve ser ativado. DSR=0: dado que chega ao port é ignorado DSR=1: dado que chega ao port é recebido
DTR (*Data Terminal Ready*) Controle de fluxo de entrada Controlado pelo DTE	Se **fDtrControl** = DTR_CONTROL_HANDSHAKE, o controle de fluxo é automático e constitui erro se a aplicação altera o estado da linha DTR através de ***EscapeCommFunction***. Se o buffer de entrada tem espaço suficiente para receber dados (> ½buffer), o driver seta DTR. Se o espaço é pequeno (< ¼ buffer), o driver reseta DTR. Se **fDtrControl** = DTR_CONTROL_ENABLE ou DTR_CONTROL_DISABLE, o usuário é livre para setar o estado da linha DTR segundo sua conveniência. O DCE irá suspender a transmissão se a linha for para 0. O DCE irá continuar a transmissão se a linha for para 1.

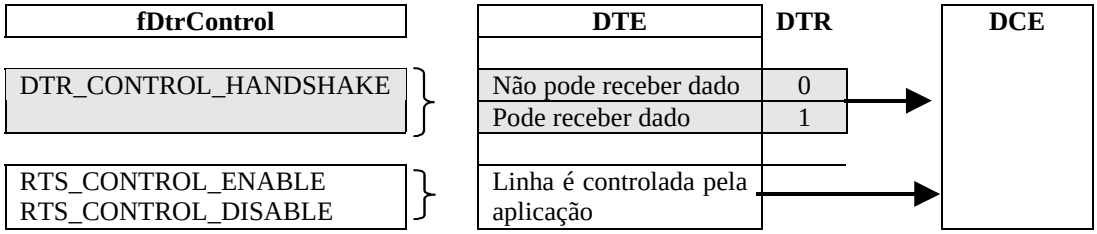
R T S



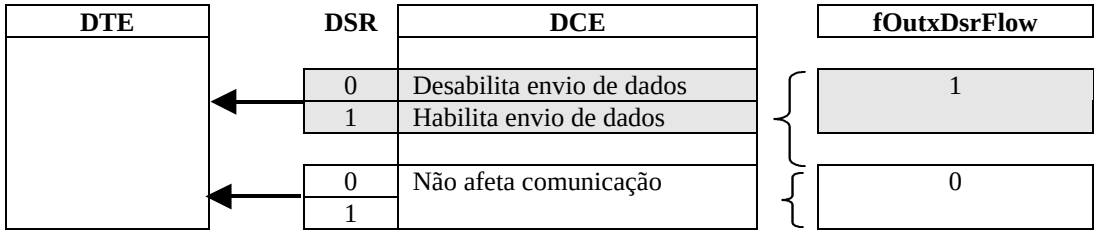
C T S



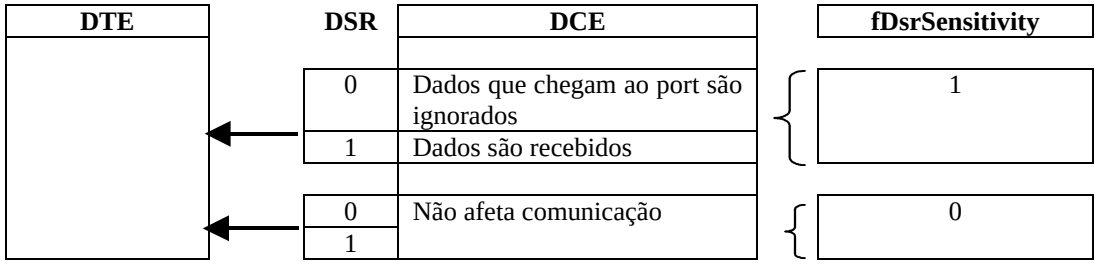
D T R



D S R - C o n t r o l e d e F l u x o d e S a í d a



D S R – C o n t r o l e d e F l u x o d e E n t r a d a



Timeout de comunicação

Devemos gerenciar quando uma comunicação não se completou devido a *timeout* e tratar este evento. A operação de leitura e escrita retorna toda vez que algum tempo de *timeout* for excedido. **A aplicação detecta este fato observando se o número de bytes escritos/lidos é menor que o valor comandado.**

Os valores dos *timeouts* de comunicação são definidos pelo usuário através de SetCommTimeouts. GetCommTimeouts deve ser chamado antes para buscar os valores correntes de *timeouts* definidos.

Exemplo: Definindo novos valores de timeouts de comunicação

```
COMMTIMEOUTS timeouts;

GetCommTimeouts(hComm, &timeouts);

timeouts.ReadlntervalTimeout = 20;
timeouts.ReadTotalTimeoutMultiplier = 10;
timeouts.ReadTotalTimeoutConstant = l00;
timeouts.WriteTotalTimeoutMultiplier = 10;
timeouts.WriteTotalTimeoutConstant = 100;

if (!SetCommTimeouts(hComm, &timeouts))
      // Erro definindo time-outs.
```

Parâmetro	Definição
ReadIntervalTimeout	Tempo máximo entre a chegada de dois caracteres consecutivos (ms). Pode ser 0.
ReadTotalTimeoutMultiplier	Recepção: Tempo de timeout em ms para cada caracter. Será multiplicado pelo número de caracteres esperados (nbytes).
ReadTotalTimeoutConstant	Recepção: Tempo de timeout em ms a ser adicionado a ReadTotalTimeoutMultiplier * nbytes. (Timeout_total = multiplier * nbytes + constant)
WriteTotalTimeoutMultiplier	Transmissão: Tempo de timeout em ms para cada caracter. Será multiplicado pelo número de caracteres enviados (nbytes).
WriteTotalTimeoutConstant	Transmissão: Tempo de timeout em ms a ser adicionado a WriteTotalTimeoutMultiplier * nbytes. (Timeout_total = multiplier * nbytes + constant)

Se definirmos todos os valores da estrutura para 0, então nenhum *timeout* irá ocorrer e uma operação síncrona irá ficar bloqueada até que todos os bytes sejam transferidos.

A definição abaixo, força as operações de leitura a se completarem imediatamente, sem esperar pela chegada de novos dados. Esta definição é necessária quando realizando leituras baseadas em eventos:

```
COMMTIMEOUTS timeouts;

GetCommTimeouts(hComm, &timeouts);

timeouts.ReadlntervalTimeout = MAXDWORD;
timeouts.ReadTotalTimeoutMultiplier = 0;
timeouts.ReadTotalTimeoutConstant = 0;
timeouts.WriteTotalTimeoutMultiplier = 0;
timeouts.WriteTotalTimeoutConstant = 0;

if (!SetCommTimeouts(hComm, &timeouts))
// Erro definindo time-outs.
```

Exemplo: Detectando timeout de comunicação

```
BOOL WriteABuffer(char * lpBuf, DWORD dwToWrite)
{
    OVERLAPPED osWrite = {0);
    DWORD dwWritten;
    DWORD dwRes;
    BOOL fRes;

    // Cria evento e salva na estrutura OVERLAPPED
    osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (osWrite.hEvent == NULL)
       // Erro na criação do objeto Evento.
       return FALSE;

    // Emite operação de escrita
    if (!WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, &osWrite) ) {
       if (GetLastError() != ERROR IO PENDING) {
          // Operação de escrita falhou e não está pendente. Reportar erro.
          fRes = FALSE;
       }
       else
         // Operação está pendente
         dwRes = WaitForSingleObject(osWrite.hEvent, INFINITE);
         switch(dwRes) {
            case WAIT OBJECT 0: // Evento foi sinalizado
              if (GetOverlappedResult(hComm, &osWrite, &dwWritten, FALSE))
                        fRes = FALSE;
                  else {
                     if (dwWritten != dwToWrite} {
                        // A operação de escrita apresentou timeout. Decisões
                        // possíveis: abortar ou tentar de novo (retry)
```

```
                        // Retry: Enviar apenas bytes que não havia sido enviados
                        // Abortar: fRes= FLASE e return
                        fRes = FALSE;
                    }
                    else
                        // Operação completada com sucesso.
                        fRes = TRUE;
                }
                break;
            default:
                // Ocorreu um erro em WaitForSingleObject. Isto indica um
                // problema com o handle com o evento da estrutura ovelapped
                fRes = FALSE;
                break;
        }
    }
}
else {
    // Operação de escrita completada imediatamente
    if (dwWritten != dwToWrite) {
        // A operação de escrita apresentou timeout. Decisões possíveis:
        // Abortar ou tentar de novo (retry)
        // Retry: Enviar apenas bytes que não havia sido enviados
        // Abortar: fRes= FLASE e return
        fRes = FALSE;
    }
    else
        fRes = TRUE;
    }
    CloseHandle(osWrite.hEvent);
    return fRes;
}
```

Usando um componente em Delphi para implementar a comunicação serial

Vamos utilizar o componente TserialNG desenvolvido por Ekkehard Domning. Para iniciar compile e rode o exemplo1: SerialNGBasicDemo. Antes você deve instalar o componente conforme roteiro abaixo. Para este teste utilize um cabo *loopback* apenas com os pinos 2 e 3 curto circuitados. Nós utilizaremos a mesma porta para ler e escrevewr.

Como instalar o componente TSerialPortNG

Abra o diretório ..SerialNg\.
Abra o arquivo SerialNg.pas

O Delphi será chamado e o arquivo será exibido.
Escolha o Menu **Component >Install Component...**

O UnitFile name já virá completo com o nome do arquivo SerialNg.pas
O Search path será completado automaticamente
Nós iremos colocar o componente no package de nome: ATR.dpk (Automação em Tempo Real)
Complete a descrição do pacote

Clique OK e o componente será compilado e criado.

Abra o Menu Component >Configure Palette
Procure na coluna da esquerda (pages) o nome da página configurada no comando Register:

```
procedure Register;
begin
  RegisterComponents('ATR', [TSerialPortNG]);
end;
```

Na pasta ATR você verá o nome do componente e o seu ícone, definido no arquivo TSerialPortNG.dcr.
Você pode alterar a ordem das abas das pastas atuando sobre as teclas Move Up e Move Down do menu.

Exemplo1: Programa SerialNGBasicDemo

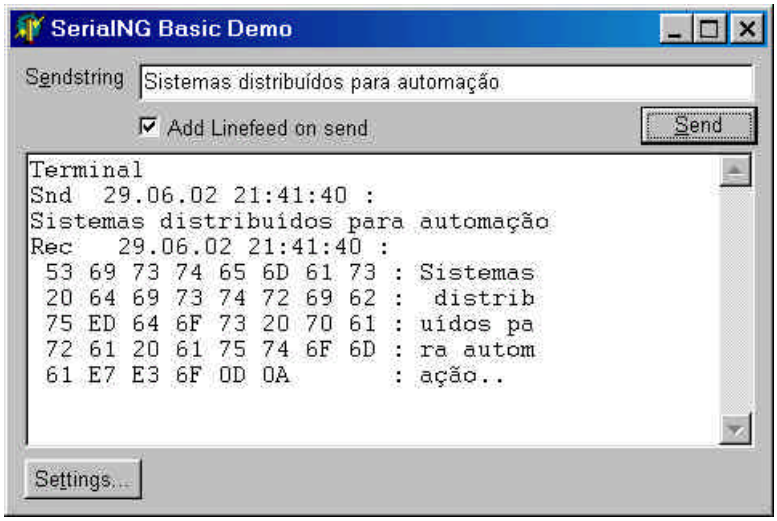


Figura 2 Janela principal do demo SerialNGBasicDemo

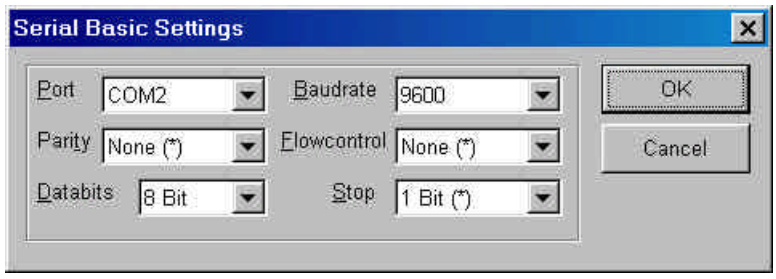


Figura 3: Janela de Stettings do demo SerialNGBasicDemo

Exemplo1: Programa SerialNGAdvDemo

Utilizando o mesmo cabo null modem faça um teste do port utilizando o programa SerialNGAdvDemo.

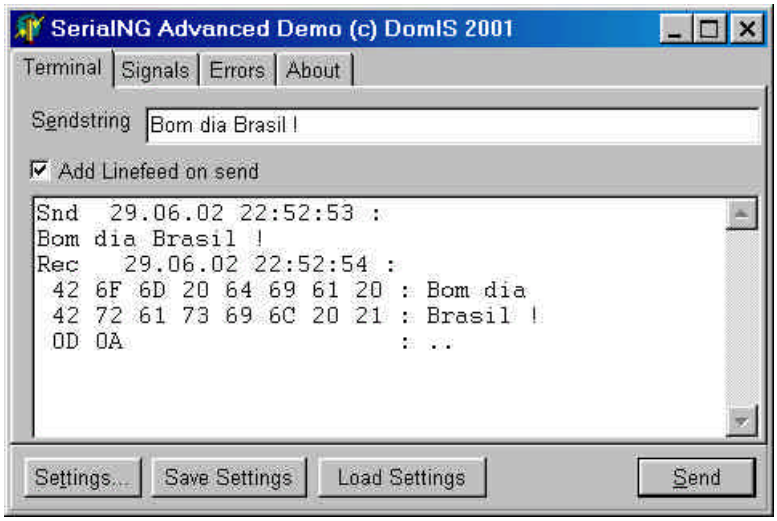


Figura 4: Teste da interface com programa SerialNGAdvDemo

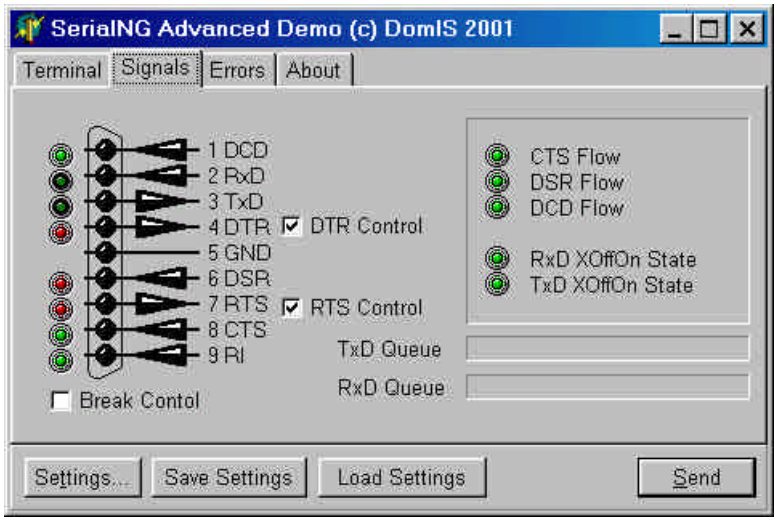


Figura 5: Janela de análise de sinais da interface. Facilita teste do HW

As principais funções utilizadas no primeiro exemplo são:

procedure SendString(S : String);	Envia um string	
function NextClusterSize : Integer;	Checa a quantidade de dados recebidos no próximo cluster. -1: lista do cluster esta vazia 0: Sem dados mas com informação de erro >=1: Bytes recebidos	
function NextClusterCCError : DWord;	Checa por ErrorCXode do próximo Cluster. MAXDWORD se não há cluster na lista, senão ErrorCode. O ErrorCode é um campo de bits e contém:	
	CE_RXOVER	Overflow da fila interna do Windows. Você pode ampliar a fila com SetRxQueueSize
	CE_OVERRUN	Erro de overrun
	CE_RXPARITY	Erro de paridade
	CE_FRAME	Erro de formação de quadro
	CE_BREAK	Break detectado
property OnRxClusterEvent : TNotifyEvent	Um cluster foi recebido e colocado na ClusterQueue. Você pode ler o cluster com uma das funções ReadCluster.	
function ReadNextClusterAsString : String;	Lê o próximo cluster como string.	
property Active : Boolean;	Se Active é falso o port está fechado. Se Active é feito igual a True, o Componente abre o port. Se oport não pode ser aberto, o sestado do port permanece fechado. Definido o valor para False, o port é fechado.	
property OnProcessError : TnotifyErrorEvent;	Todo erro de processamento é reportado aqui. A quantidade de mensagens é controlada pelo campo erroNoise.	

Programas adicionais

1. O programa MTTTY do diretório \4918 é um exemplo completo de um aplicativo de comunicação serial e inclui os programas fontes. Este programa acompanha o paper [Denver 95]

2. O programa smap.exe do diretório \Bessonov é um espião do processo de comunicação serial. Este programa intercepta todas as chamadas ao Win32 referentes o processo de comunicação e exibe um trace dos resultados [Bessonov 99]. Todos os programas fontes estão incluídos.

3. O programa breakout.exe usa um PC como caixa para interceptar um link serial. Uma das portas do PC será utilizada como entrada e a outra como saída. Todo o processo de comunicação será monitorado.

4. O programa shoports.exe exibe as portas existentes em um PC.

5. O programa comshow.exe é um monitor simples de comunicação serial e exibe os dados em formato *row* (bruto).

6. O programa hyperterminal é um emulador de terminais muito útil para interfacear com dispositivos que comunicam com terminais burros.

7. Serial demo contém uma biblioteca de classes em C++ para uso por threads especializadas em comunicação (Cserial) uma outra para uso por uma GUI thread que recebe as mensagens na fila de mensagens (CserialWnd) e outra para uso em MFC (CSerialMFC) extraídas do site: www.codeproject.com/system/serial.asp. Inclui exemplos de utilização.

8. SerialNg.zip contém um componente em Delphi que implementa a comunicação serial extraído de www.domis.de/serial.htm e três programas de teste:
   SerialNGBasicDemo: Demo muito simples. Inclui apenas funções básicas do componente.
   SerialNGAdvDemo: Um demo muito complexo, que usa muitas das capabilidades do componente.
   SerialNDStress: Um demo para testar o tratamento de erros de transmissão.

9. O programa asciich.exe exibe uma tabela de caracteres ASCII na tela do PC.

Bibliografia

[Denver 95] Allen Denver, Serial Communications in Win32, Microsoft Windows Developer Support, 1995. http://msdn.microsoft.com/library/techart/msdn_serial.htm.

[Bessonov 99] Alex V. Bessonov. A serial Port Spy for NT. Windows Developer's Journal. October 1999, pg 8..35.

[Thompson 97] Lawrence M. Thompson, Industrial Data Communications, 2nd edition, ISA series, 1997.

[Msdn 99] MSDSN Library, Microsoft, July 1999.

[Lein 99] Ramon de Klein, Serial library for C++, www.codeproject.com/system/serial.asp

Sites

Serial Port Central	www.lvr.com/serport.htm
Componente serial em Delphi	www.domis.de/serial.htm
Biblioteca serial em C++ para Win32 e MFC	www.codeproject.com/system/serial.asp
Tutorial sobre comunicação serial da Taltech	www.taltech.com/TALtech_web/resources/

Exercícios

1) Qual o caracter ASCII que quando transmitido em uma linha seria geral uma onda quadrada perfeita ?

2) Instale o programa PortSpy e monitore a comunicação do seu mouse serial.

3) O que acontece se passamos TRUE no parâmetro bWait na instrução *GetOverlappedResult()* ?

4) Num processo de comunicação você está tendo uma alta incidência do erro CE_RXOVER. O que isto significa ? Como você pode corrigir este problema?

5) Num processo de comunicação você está tendo uma alta incidência do erro CE_OVERRUN. O que isto significa ? Como você pode corrigir este problema ([Denver 95, pag 21]) ?

6) Projete um programa para envio e recepção de arquivos utilizando uma linha serial em um PC. O seu programa deverá ler um arquivo em setores de 512 bytes , dividir cada bloco em quadros de 128 bytes, que serão enviados por uma thread especializada.

   Esta thread reformatará o quadro adicionando um cabeçalho no início, contendo o tamanho do quadro e sua identificação (um número seqüencial de 16 bits) e uma palavra de CRC ao final, calculado segundo o polinômio CRC_CCITT.

   Cada quadro recebido terá o seu CRC conferido na recepção. Para cada quadro recebido, a thread de recepção enviará um quadro de ACK, se o quadro tiver sido recebido corretamente, ou NACK em caso contrário. O arquivo será armazenado na recepção em um diretório de nome \CommRx e deverá receber o mesmo nome do arquivo origem.

   A comunicação será síncrona. Se houver erro de transmissão o quadro com problema deverá ser reenviado até 3 vezes. Depois disso, a transmissão será abortada e a condição de erro reportada.

   O programa principal deverá configurar os canais seriais, programar os tempos de timeout de comunicação e atualizar o status da transmissão/recepção da mensagem na tela.

   O programa deve ser escrito em C/C++ utilizando o compilador Microsoft Visual C 6 ou superior.

Non-Printing Characters					Printing Characters								
Name	**Ctrl char**	**Dec**	**Hex**	**Char**	**Dec**	**Hex**	**Char**	**Dec**	**Hex**	**Char**	**Dec**	**Hex**	**Char**
null	ctrl-@	0	00	NUL	32	20	Space	64	40	@	96	60	`
start of heading	ctrl-A	1	01	SOH	33	21	!	65	41	A	97	61	a
start of text	ctrl-B	2	02	STX	34	22	"	66	42	B	98	62	b
end of text	ctrl-C	3	03	ETX	35	23	#	67	43	C	99	63	c
end of xmit	ctrl-D	4	04	EOT	36	24	$	68	44	D	100	64	d
enquiry	ctrl-E	5	05	ENQ	37	25	%	69	45	E	101	65	e
acknowledge	ctrl-F	6	06	ACK	38	26	&	70	46	F	102	66	f
bell	ctrl-G	7	07	BEL	39	27	'	71	47	G	103	67	g
backspace	ctrl-H	8	08	BS	40	28	(	72	48	H	104	68	h
horizontal tab	ctrl-I	9	09	HT	41	29	)	73	49	I	105	69	i
line feed	ctrl-J	10	0A	LF	42	2A	*	74	4A	J	106	6A	j
vertical tab	ctrl-K	11	0B	VT	43	2B	+	75	4B	K	107	6B	k
form feed	ctrl-L	12	0C	FF	44	2C	,	76	4C	L	108	6C	l
carriage feed	ctrl-M	13	0D	CR	45	2D	-	77	4D	M	109	6D	m
shift out	ctrl-N	14	0E	SO	46	2E	.	78	4E	N	110	6E	n
shift in	ctrl-O	15	0F	SI	47	2F	/	79	4F	O	111	6F	o
data line escape	ctrl-P	16	10	DLE	48	30	0	80	50	P	112	70	p
device control 1	ctrl-Q	17	11	DC1	49	31	1	81	51	Q	113	71	q
device control 2	ctrl-R	18	12	DC2	50	32	2	82	52	R	114	72	r
device control 3	ctrl-S	19	13	DC3	51	33	3	83	53	S	115	73	s
device control 4	ctrl-T	20	14	DC4	52	34	4	84	54	T	116	74	t
neg acknowledge	ctrl-U	21	15	NAK	53	35	5	85	55	U	117	75	u
synchronous idel	ctrl-V	22	16	SYN	54	36	6	86	56	V	118	76	v
end of xmit block	ctrl-W	23	17	ETB	55	37	7	87	57	W	119	77	w
cancel	ctrl-X	24	18	CAN	56	38	8	88	58	X	120	78	x
end of medium	ctrl-Y	25	19	EM	57	39	9	89	59	Y	121	79	y
substitute	ctrl-Z	26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
escape	ctrl-[	27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
file separator	ctrl-\	28	1C	FS	60	3C	<	92	5C	\	124	7C	|
group separator	ctrl-]	29	1D	GS	61	3D	=	93	5D	]	125	7D	}
record separator	ctrl-^	30	1E	RS	62	3E	>	94	5E	^	126	7E	~
unit separator	ctrl-_	31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

"http://www.physics.udel.edu/~watson/scen103/ascii.html".Copyright George Watson, Univ. of Delaware, 1996.