Protocolo de Ligação de Dados

(1º Trabalho Laboratorial)

FEUP

Redes de Computadores

Descrição do trabalho

- Objectivos
  - » Implementar um protocolo de ligação de dados, de acordo com a especificação a seguir descrita
  - » Testar o protocolo com uma aplicação simples de transferência de ficheiros, igualmente especificada

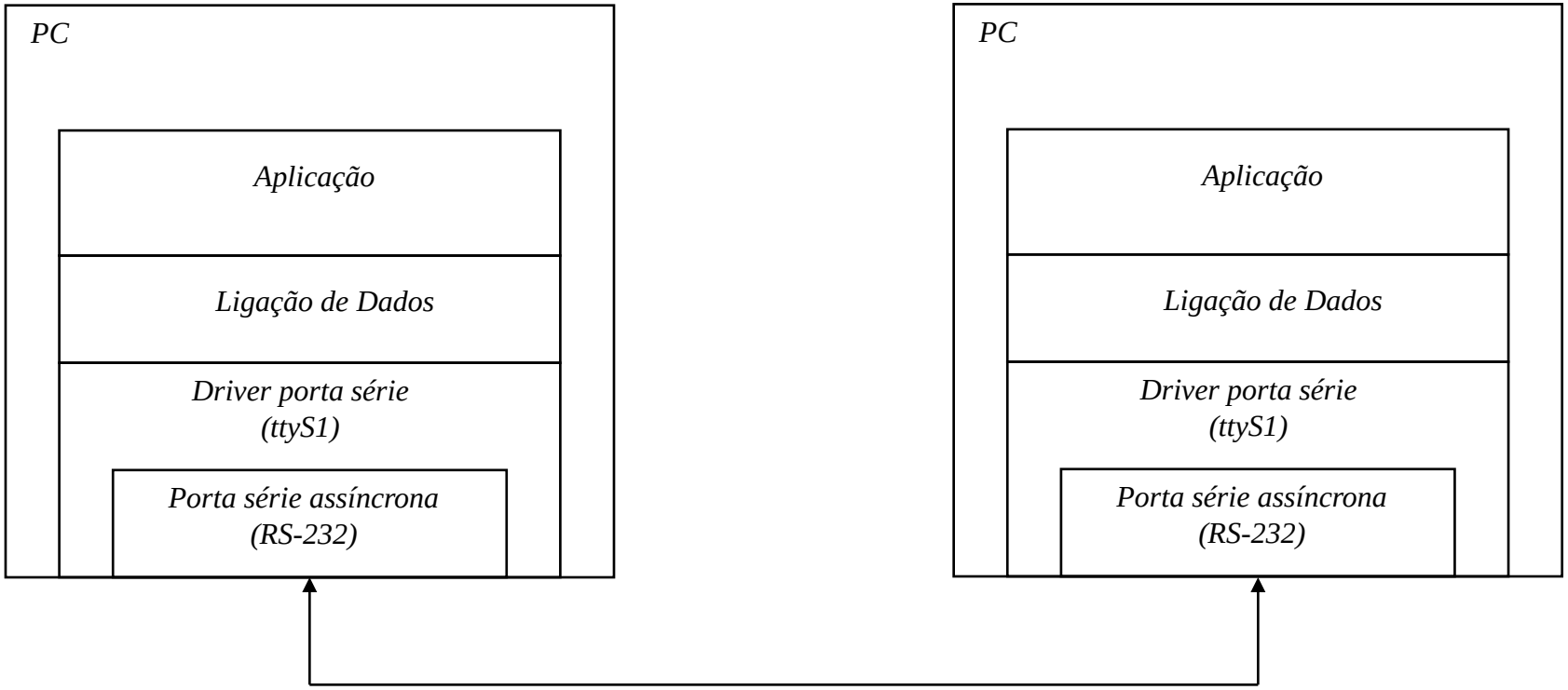
- Ambiente de desenvolvimento
  - » PC com LINUX
  - » Linguagem de programação – C
  - » Portas série RS-232 (comunicação assíncrona)

Funcionamento e avaliação

- Funcionamento
  - » Grupos com 3 elementos
  - » Cada grupo realiza o emissor e o receptor

- Elementos de Avaliação
  - » Participação nas aulas (avaliação contínua)
  - » Apresentação e demonstração do trabalho
  - » Relatório final

Configuração de teste



Protocolos de Ligação de Dados – Funções

- Objetivo do Protocolo de Ligação de Dados
  - » Fornecer um serviço de comunicação de dados fiável entre dois sistemas ligados por um meio (canal) de transmissão – neste caso, um cabo série
- Funções genéricas de protocolos de ligação de dados
  - » Sincronismo (delimitação) de trama – dados organizados em tramas (*framing*)
    - – Principais alternativas: caracteres / *flags* para delimitar início e fim de trama
    - – Tamanho dos dados pode ser implícito ou indicado explicitamente no cabeçalho
  - » Estabelecimento / terminação da ligação
  - » Numeração de tramas
  - » Confirmação positiva
    - – Após recepção de uma trama sem erros e na sequência correcta
  - » Controlo de erros (exemplos: *Stop-and-Wait*, *Go-back-N*, *Selective Repeat*)
    - – Temporizadores (*time-out*) – retransmissão decidida pelo emissor
    - – Confirmação negativa (tramas fora da sequência esperada) – retransmissão a pedido do receptor
    - – Retransmissões podem originar duplicados (que devem ser detetados e eliminados)
  - » Controlo de fluxo

Protocolo de Ligação de Dados – Especificação

- O protocolo a implementar combina características de protocolos de ligação de dados existentes
  - » O protocolo garante transmissão de dados independente de códigos (transparência)
  - » A transmissão é organizada em tramas, que podem ser de três tipos – Informação (I), Supervisão (S) e Não Numeradas (U)
    - – As tramas têm um cabeçalho com um formato comum
    - – Apenas as tramas de Informação possuem um campo para transporte de dados (este campo transporta um pacote gerado pela aplicação, mas o seu conteúdo não é processado pelo protocolo de ligação de dados – independência entre camadas)
  - » A delimitação de tramas é feita por meio de uma sequência especial de oito bits (*flag*) e a transparência é assegurada pela técnica de *byte stuffing*
  - » As tramas são protegidas por um código detetor de erros
    - – Nas tramas S e U existe proteção simples da trama (visto que não transportam dados)
    - – Nas tramas I existe proteção dupla e independente do cabeçalho e do campo de dados (o que permite usar um cabeçalho válido, mesmo que ocorra erro no campo de dados)
  - » Usa-se a variante *Stop and Wait* (janela unitária e numeração módulo 2)

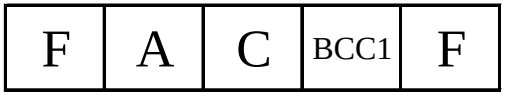
Formato e tipos de tramas

» Tramas de Informação (I)

(antes de *stuffing* e após *destuffing*)

F **Flag**
A **Campo de Endereço**
C **Campo de Controlo** **0 0 S 0 0 0 0 0** **S = N(s)**
D_1 ... D_N **Campo de Informação (contém pacote gerado pela Aplicação)**
$BCC_{1, 2}$ **Campos de Protecção independentes (1 – cabeçalho, 2 – dados)**

» Tramas de Supervisão (S) e Não Numeradas (U)



F **Flag**
A **Campo de Endereço**
C **Campo de Controlo**
SET (set up) **0 0 0 0 0 1 1 1**
DISC (disconnect) **0 0 0 0 1 0 1 1**
UA (unnumbered acknowledgment) **0 0 0 0 0 0 1 1**
RR (receiver ready / positive ACK) **0 0 R 0 0 0 0 1**
REJ (reject / negative ACK) **0 0 R 0 0 1 0 1** **R = N(r)**
BCC_1 **Campo de Proteção (cabeçalho)**

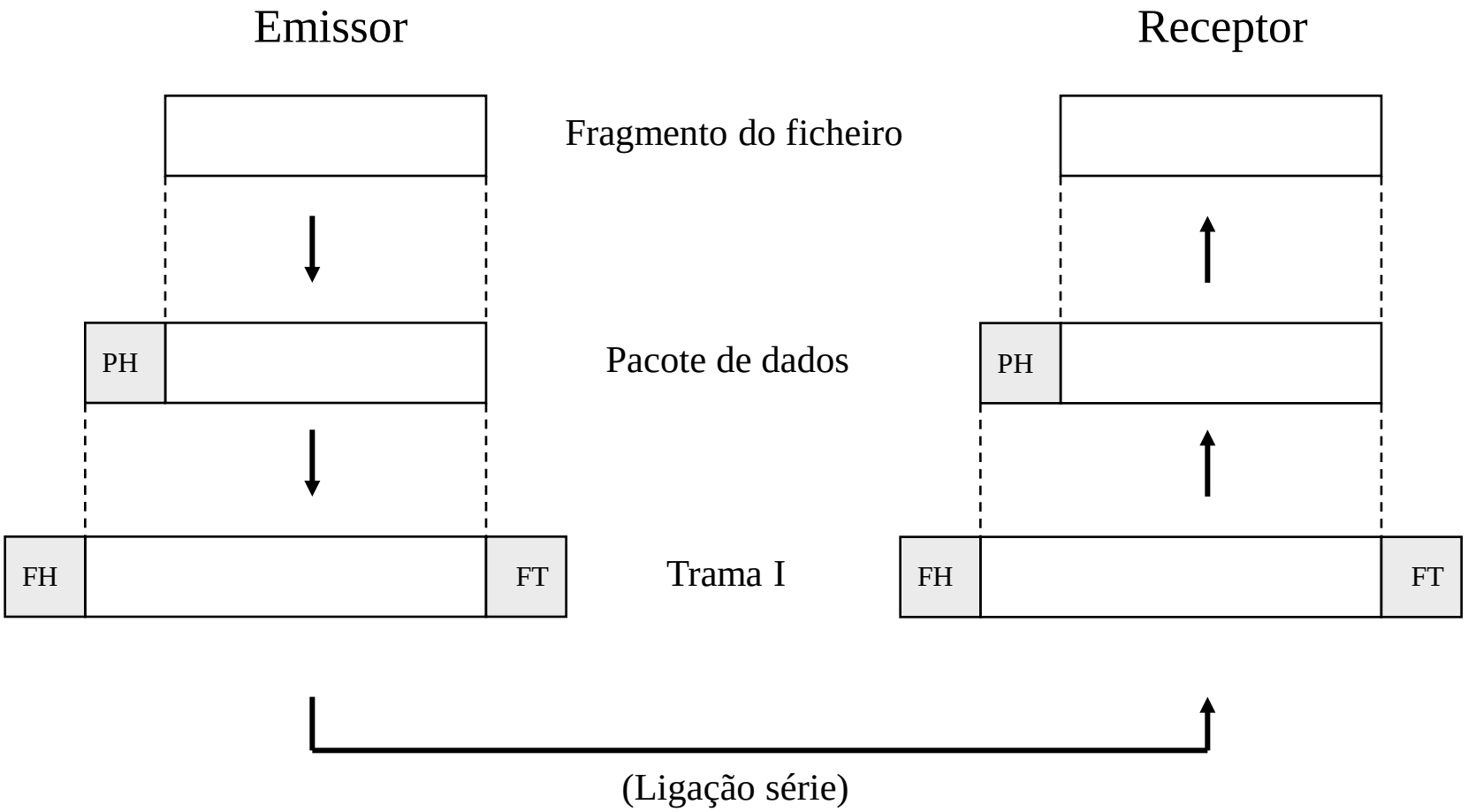
Pacotes e tramas

- O ficheiro a transmitir é fragmentado – os fragmentos são encapsulados em pacotes de dados e estes são transportados no campo de dados de tramas I
  - Para além de pacotes de dados (que contêm fragmentos do ficheiro), o protocolo de Aplicação usa pacotes de controlo
  - O formato dos pacotes (de dados e de controlo) é definido adiante

- Designa-se por Emissor a máquina que envia o ficheiro e por Receptor a máquina que recebe o ficheiro
  - Apenas o Emissor transmite pacotes (de dados ou de controlo) e portanto apenas o Emissor transmite tramas I

- Quer o Emissor quer o Receptor enviam e recebem tramas

Pacotes de dados e tramas I



PH – *Packet Header*
FH – *Frame Header*
FT – *Frame Trailer*

Os pacotes de controlo são também transportados em tramas I

Tramas – Delimitação e cabeçalho

- Todas as tramas são delimitadas por *flags* (**01111110**)
- Uma trama pode ser iniciada com uma ou mais *flags*, o que deve ser tido em conta pelo mecanismo de recepção de tramas
- Tramas I, SET e DISC são designadas Comandos e as restantes (UA, RR e REJ) Respostas
- As tramas têm um cabeçalho com um formato comum
  - A (Campo de Endereço)
    - **00000011** (**0x03**) em Comandos enviados pelo Emissor e Respostas enviadas pelo Receptor
    - **00000001** (**0x01**) em Comandos enviados pelo Receptor e Respostas enviadas pelo Emissor
  - C (Campo de Controlo) – define o tipo de trama e transporta números de sequência N(s) em tramas I e N(r) em tramas de Supervisão (RR, REJ)
  - BCC (*Block Check Character*) – detecção de erros baseada na geração de um octeto (BCC) tal que exista um número par de 1s em cada posição (bit), considerando todos os octetos protegidos pelo BCC (cabeçalho ou dados, conforme os casos) e o próprio BCC (antes de *stuffing*)

Recepção de tramas – Procedimentos

- Tramas I, S ou U com cabeçalho errado são ignoradas, sem qualquer acção
- O campo de dados das tramas I é protegido por um BCC próprio (paridade par sobre cada um dos bits dos octetos de dados e do BCC)
- Tramas I recebidas sem erros detectados no cabeçalho e no campo de dados são aceites para processamento
  - Se se tratar duma nova trama, o campo de dados é aceite (e passado à Aplicação), e a trama deve ser confirmada com RR
  - Se se tratar dum duplicado, o campo de dados é descartado, mas deve fazer-se confirmação da trama com RR
- Tramas I sem erro detectado no cabeçalho mas com erro detectado (pelo respectivo BCC) no campo de dados – o campo de dados é descartado, mas o campo de controlo pode ser usado para desencadear uma acção adequada
  - Se se tratar duma nova trama, é conveniente fazer um pedido de retransmissão com REJ, o que permite antecipar a ocorrência de *time-out* no emissor
  - Se se tratar dum duplicado, deve fazer-se confirmação com RR
- Tramas I, SET e DISC são protegidas por um temporizador
  - Em caso de ocorrência de *time-out*, deve ser efectuado um número máximo de tentativas de retransmissão (o valor deve ser configurável; por exemplo, três)

Transparência – Necessidade

- A transmissão entre os dois computadores é, neste trabalho, baseada numa técnica designada por transmissão assíncrona
  - Esta técnica caracteriza-se pela transmissão de “caracteres” (sequência curta de bits, cujo número pode ser configurado) delimitados por bits *Start* e *Stop*
  - Alguns protocolos usam caracteres (palavras) de um código (por exemplo ASCII) para delimitar e identificar os campos que constituem as tramas e para suportar a execução dos mecanismos protocolares
    - Nestes protocolos, a transmissão de dados de forma transparente (independente do código usado pelo protocolo) obriga a recorrer a mecanismos de escape
- O protocolo a implementar não se baseia na utilização de qualquer código, pelo que os caracteres transmitidos (constituídos por 8 bits) devem ser interpretados como simples octetos (*bytes*), podendo ocorrer qualquer uma das 256 combinações possíveis
- Para evitar o falso reconhecimento de uma *flag* no interior de uma trama, é necessário um mecanismo que garanta transparência

Transparência – Mecanismo de byte stuffing

» No protocolo a implementar adota-se o mecanismo usado em PPP, que recorre ao octeto de escape **01111101** (**0x7d**)

– Se no interior da trama ocorrer o octeto **01111110** (**0x7e**), isto é, o padrão que corresponde a uma *flag*, o octeto é substituído pela sequência **0x7d 0x5e** (octeto de escape seguido do resultado do ou exclusivo do octeto substituído com o octeto 0x20)

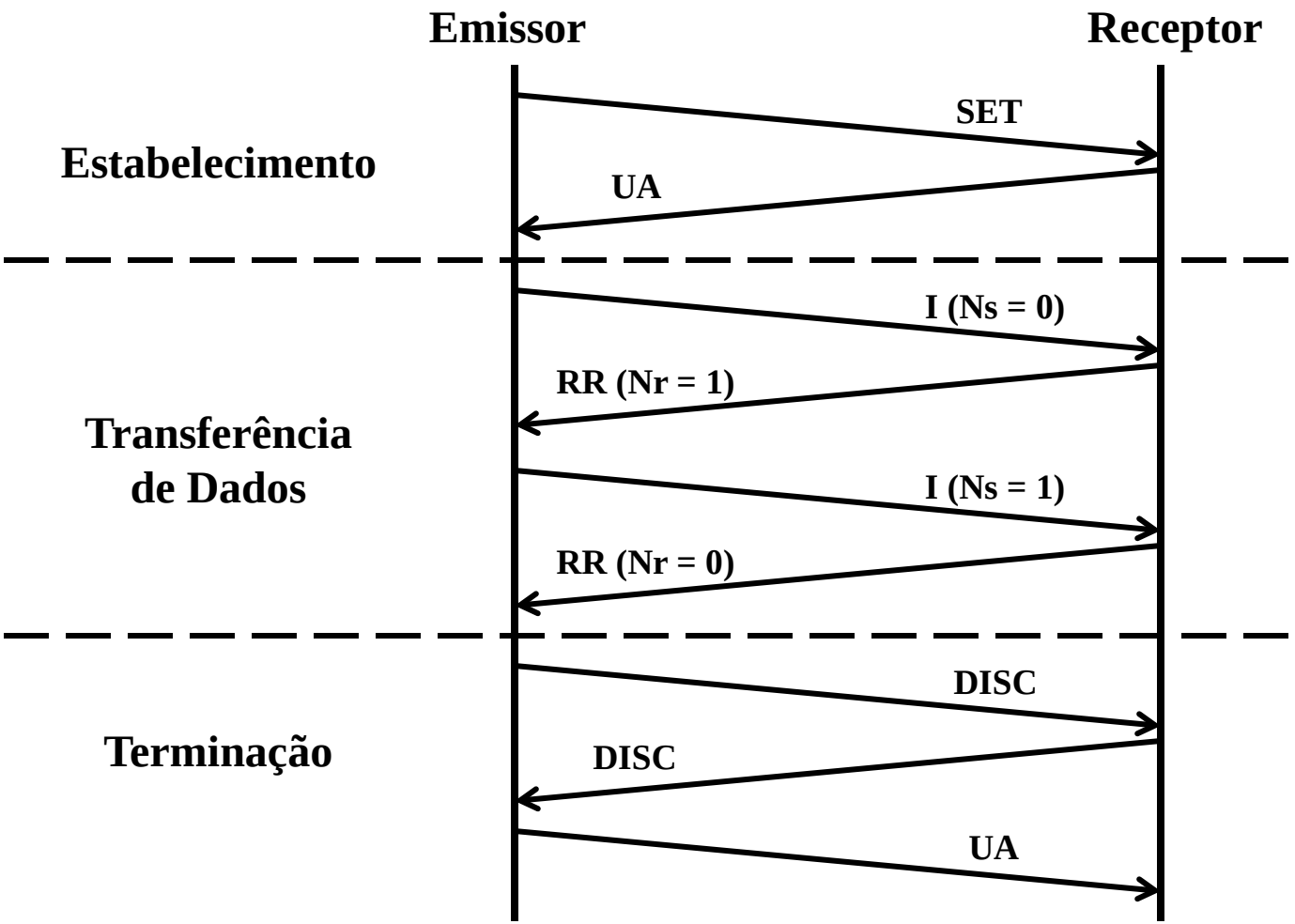
– Se no interior da trama ocorrer o octeto **01111101** (**0x7d**), isto é, o padrão que corresponde ao octeto de escape, o octeto é substituído pela sequência **0x7d 0x5d** (octeto de escape seguido do resultado do ou exclusivo do octeto substituído com o octeto 0x20)

– Na geração do BCC são considerados apenas os octetos originais (antes da operação de *stuffing*), mesmo que algum octeto (incluindo o próprio BCC) tenha de ser substituído pela correspondente sequência de escape

– A verificação do BCC é feita em relação aos octetos originais, isto é, depois de realizada a operação inversa (*destuffing*), caso tenha ocorrido na emissão a substituição de algum dos octetos especiais pela correspondente sequência de escape

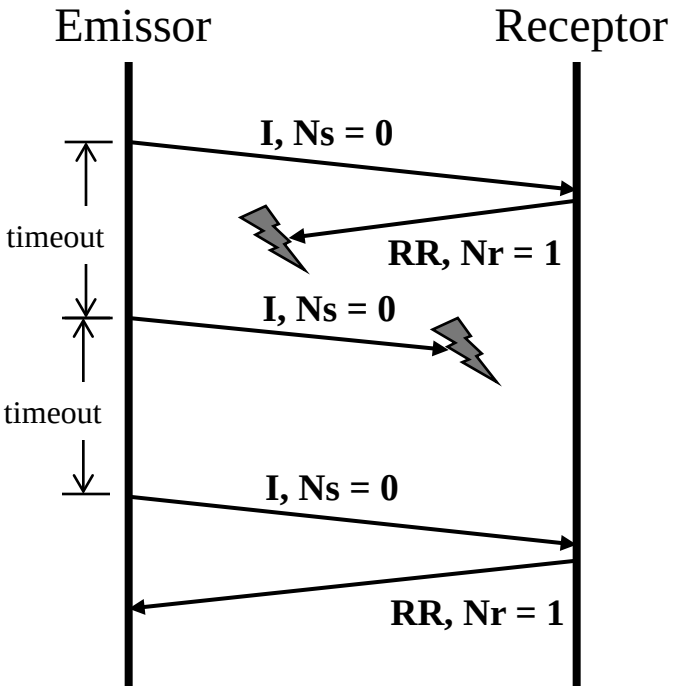
Fases do protocolo de Ligação de Dados

» Exemplo de uma sequência típica de tramas (sem erros)

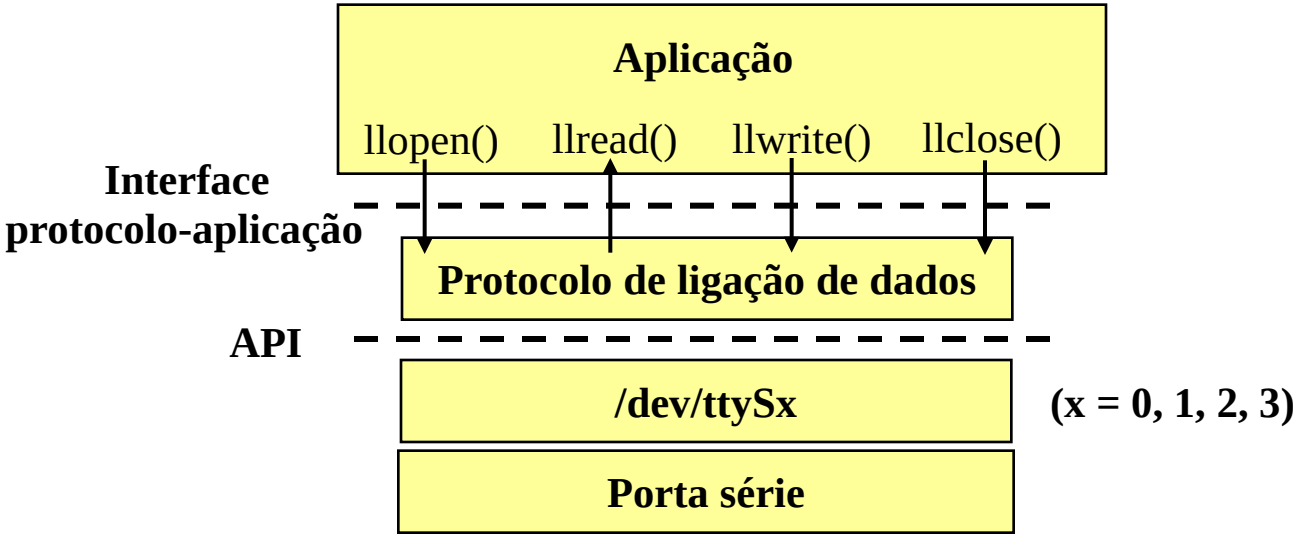


Transferência de dados – Retransmissões

- Confirmação / Controlo de Erros
  - » *Stop-and-Wait*
- Temporizador
  - » Activado após o envio de uma trama I, SET ou DISC
  - » Desactivado após recepção de uma resposta válida
  - » Se excedido (*time-out*), obriga a retransmissão
- Retransmissão de tramas I
  - » Se ocorrer *time-out*, devido à perda da trama I enviada ou da sua confirmação
    - – Número máximo predefinido (configurado) de tentativas de retransmissão
  - » Após recepção de confirmação negativa (REJ)
- Protecção da trama
  - » Geração e verificação do(s) campo(s) de protecção (BCC)



Interface Protocolo-Aplicação



Interface Protocolo-Aplicação

- Exemplos (possíveis) de estruturas de dados
  - » Aplicação

```
struct applicationLayer {
   int fileDescriptor;    /*Descritor correspondente à porta série*/
   int status;            /*TRANSMITTER | RECEIVER*/
  }
```

  - » Protocolo

```
struct linkLayer {
   char port[20];                  /*Dispositivo /dev/ttySx, x = 0, 1*/
   int baudRate;                   /*Velocidade de transmissão*/
   unsigned int sequenceNumber;    /*Número de sequência da trama: 0, 1*/
   unsigned int timeout;           /*Valor do temporizador: 1 s*/
   unsigned int numTransmissions; /*Número de tentativas em caso de
                                    falha*/
   char frame[MAX_SIZE];           /*Trama*/
}
```

Interface Protocolo-Aplicação – open

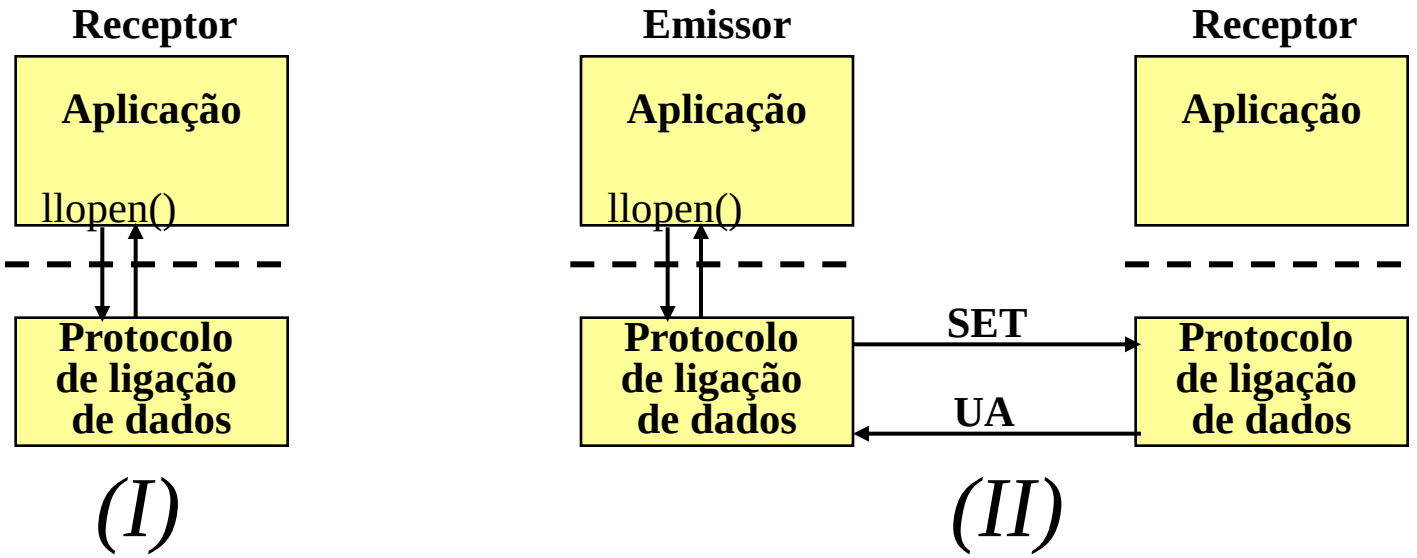
int llopen(int porta, TRANSMITTER | RECEIVER)

argumentos

- porta: COM1, COM2, ...
- flag: TRANSMITTER / RECEIVER

retorno

- identificador da ligação de dados
- valor negativo em caso de erro



Interface Protocolo-Aplicação – read / write

int llwrite(int fd, char * buffer, int length)

argumentos

- fd: identificador da ligação de dados
- buffer: *array* de caracteres a transmitir
- length: comprimento do *array* de caracteres

retorno

- número de caracteres escritos
- valor negativo em caso de erro

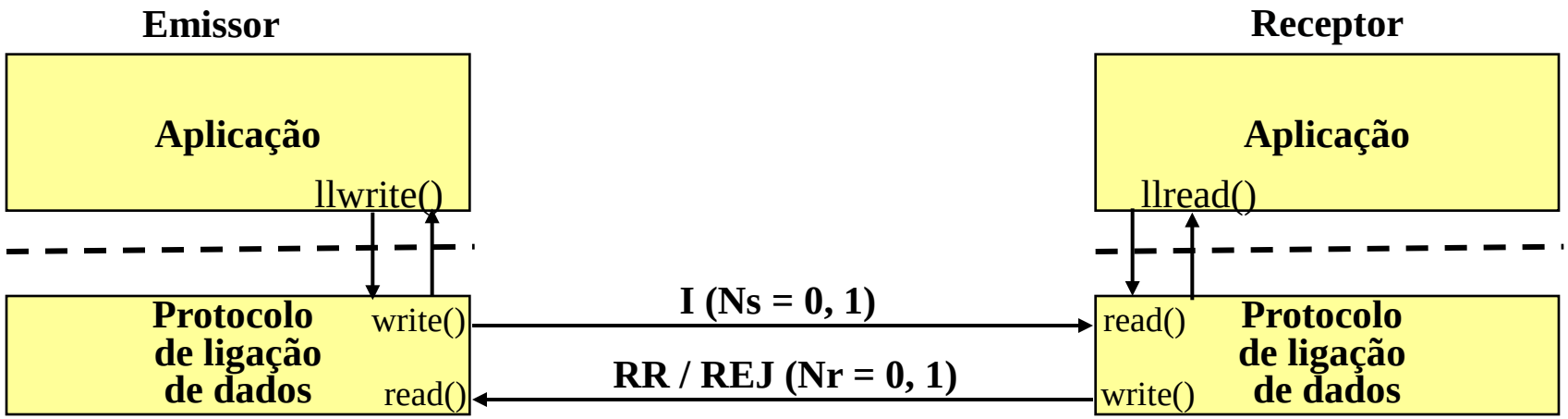
int llread(int fd, char * buffer)

argumentos

- fd: identificador da ligação de dados
- buffer: *array* de caracteres recebidos

retorno

- comprimento do *array* (número de caracteres lidos)
- valor negativo em caso de erro



Interface Protocolo-Aplicação – close

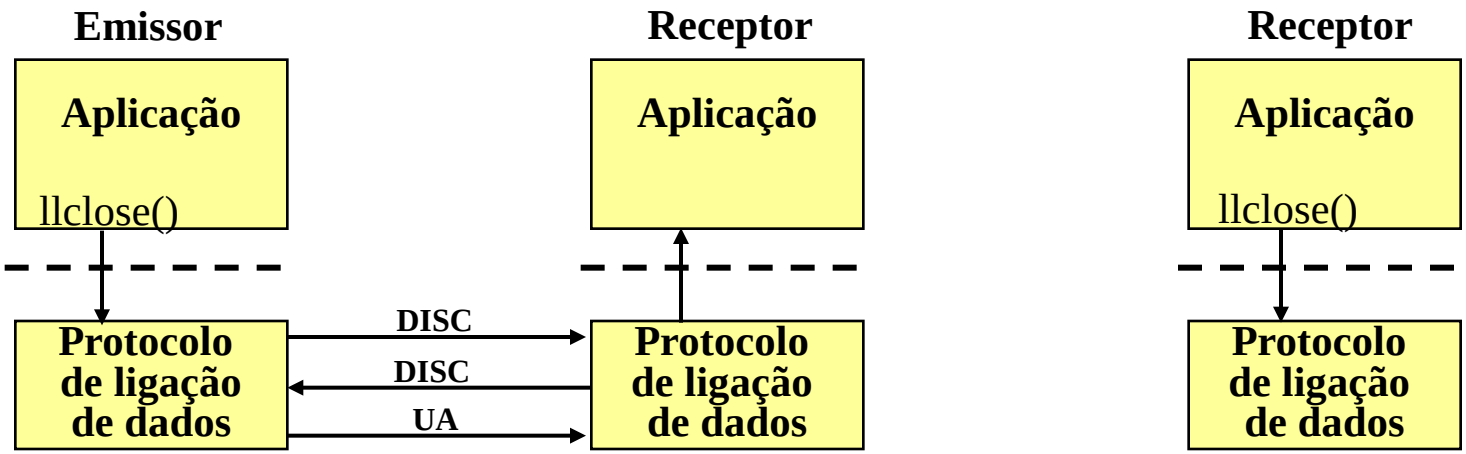
int llclose(int fd)

argumentos

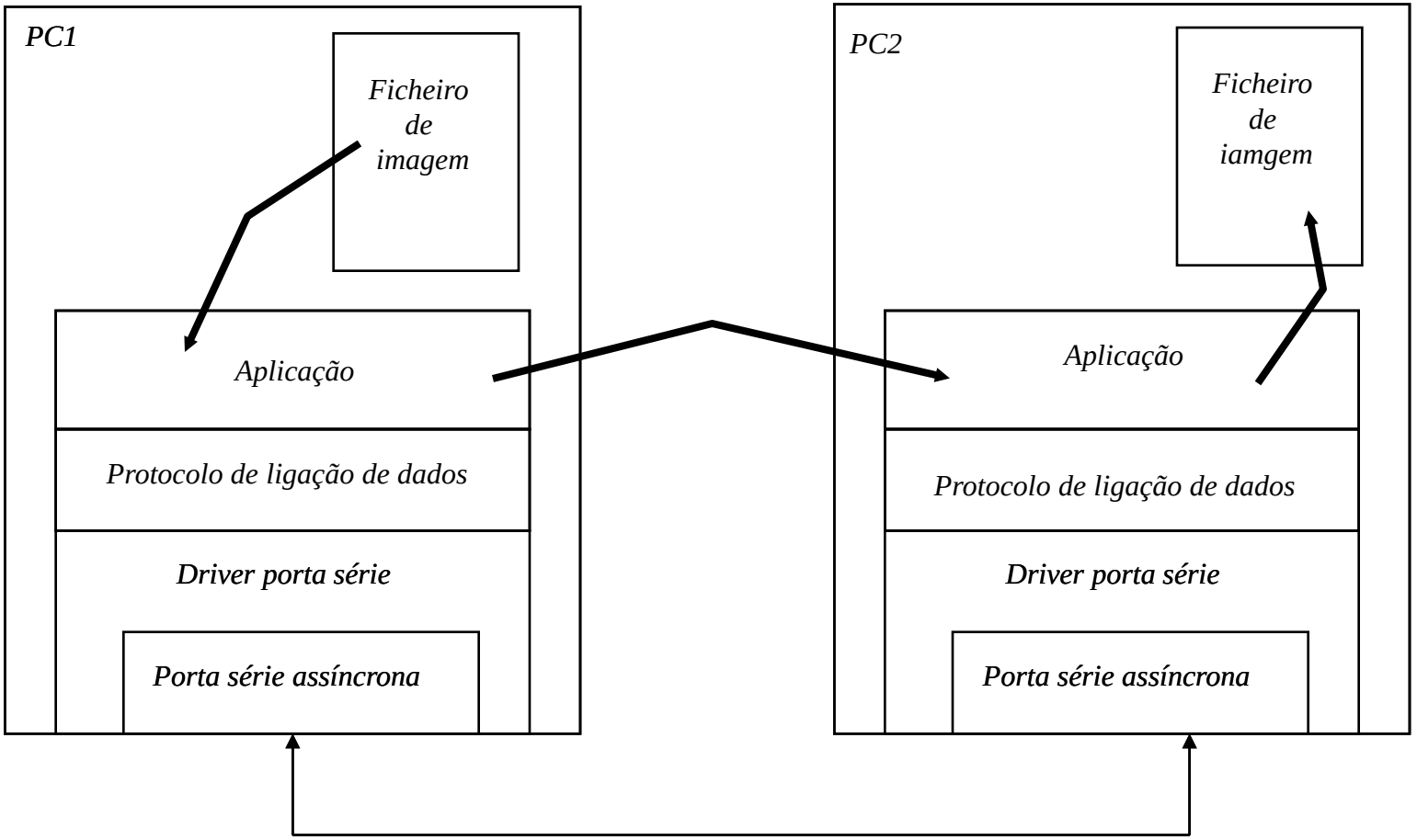
- fd: identificador da ligação de dados

retorno

- valor positivo em caso de sucesso
- valor negativo em caso de erro



Aplicação de teste

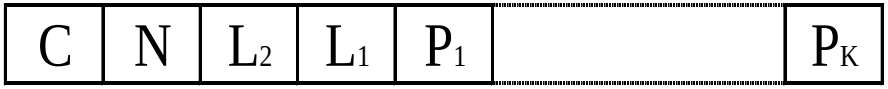


Aplicação de teste – Especificação

- Pretende-se desenvolver um protocolo de aplicação muito simples para transferência de um ficheiro, usando o serviço fiável oferecido pelo protocolo de ligação de dados
- A aplicação deve suportar dois tipos de pacotes enviados pelo Emissor
  - » Pacotes de controlo para sinalizar o início e o fim da transferência do ficheiro
  - » Pacotes de dados contendo fragmentos do ficheiro a transmitir
- O pacote de controlo que sinaliza o início da transmissão (*start*) deverá ter obrigatoriamente um campo com o tamanho do ficheiro e opcionalmente um campo com o nome do ficheiro (e eventualmente outros campos)
- O pacote de controlo que sinaliza o fim da transmissão (*end*) deverá repetir a informação contida no pacote de início de transmissão
- Os pacotes de dados contêm obrigatoriamente um campo (um octeto) com um número de sequência e um campo (dois octetos) que indica o tamanho do respectivo campo de dados
  - » Este tamanho depende do tamanho máximo do campo de Informação das tramas I
  - » Estes campos permitem verificações adicionais em relação à integridade dos dados
  - » A numeração de pacotes de dados é independente da numeração das tramas I

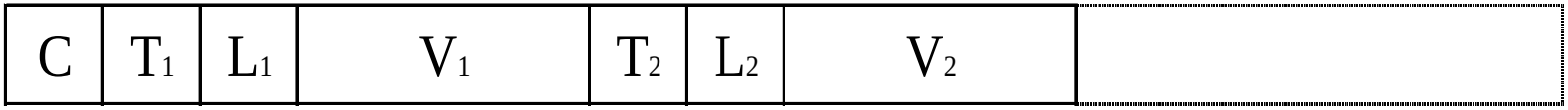
Pacotes do nível de Aplicação

- Pacote de dados



  - » C – campo de controlo (valor: 0 – dados)
  - » N – número de sequência (módulo 255)
  - » L_2 L_1 – indica o número de octetos (K) do campo de dados $(K = 256 * L_2 + L_1)$
  - » P_1 ... P_K – campo de dados do pacote (K octetos)

- Pacote de controlo



  - » C – campo de controlo (valores: 1 – *start*; 2 – *end*)
  - » Cada parâmetro (tamanho, nome do ficheiro, ou outro) é codificado na forma TLV (*Type*, *Length*, *Value*)
    - – T (um octeto) – indica qual o parâmetro (0 – tamanho do ficheiro, 1 – nome do ficheiro, outros valores – a definir, se necessário)
    - – L (um octeto) – indica o tamanho em octetos do campo V (valor do parâmetro)
    - – V (número de octetos indicado em L) – valor do parâmetro

Independência entre camadas

- As arquitecturas em camadas baseiam-se no princípio de independência entre camadas
- A aplicação deste princípio tem as seguintes consequências no âmbito deste trabalho
  - » Na camada de ligação de dados não é feito qualquer processamento que incida sobre o cabeçalho dos pacotes a transportar em tramas de Informação – esta informação é considerada inacessível ao protocolo de ligação de dados
    - – Ao nível da ligação de dados não existe qualquer distinção entre pacotes de controlo e de dados, nem é relevante (nem tida em conta) a numeração dos pacotes de dados
  - » A camada de aplicação não conhece os detalhes do protocolo de ligação de dados, mas apenas a forma como acede ao serviço
    - – O protocolo de aplicação desconhece a estrutura das tramas e o respectivo mecanismo de delineação, a existência de *stuffing* (e qual a opção adoptada), o mecanismo de protecção das tramas, eventuais retransmissões de tramas de Informação, etc.
    - – Todas estas funções são exclusivamente realizadas na camada de ligação de dados
  - » Em particular, os mecanismos de numeração de tramas I e de pacotes de dados são totalmente independentes e nenhuma relação deve ser estabelecida entre eles

Elementos de avaliação

- Protocolo de ligação de dados
  - » Processo de sincronização de trama
  - » Processo de retransmissão
  - » Robustez a erros
  - » Controlo de erros
- Protocolo de aplicação
  - » Pacotes de controlo
  - » Numeração dos pacotes de dados
- Organização do código
  - » Interface entre camadas (funções)
  - » Independência entre camadas
- Demonstração

- Penalizações
  - » Atrasos na demonstração e/ou na entrega do relatório

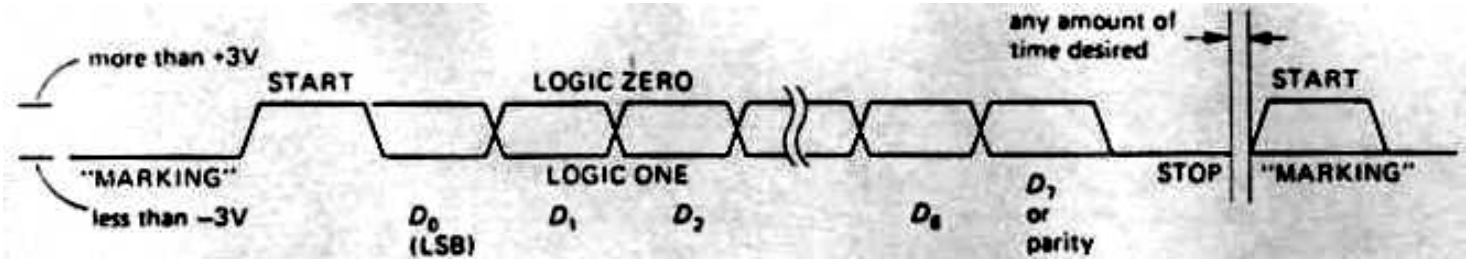
Elementos de valorização

- Selecção de parâmetros pelo utilizador
  - » *Baud rate*, tamanho máximo do campo de Informação das tramas I (sem *stuffing*), número máximo de retransmissões (*default*: 3), intervalo de *time-out*
- Geração aleatória de erros em tramas de Informação
  - » Sugestão – para cada trama I correctamente recebida, simular (no receptor) a ocorrência de erro no cabeçalho e no campo de dados com probabilidades predefinidas (e independentes), e proceder como se se tratasse dum erro real
- Implementação de REJ
- Verificação da integridade dos dados pela Aplicação
  - » Tamanho do ficheiro recebido (real vs. valor indicado nos pacotes de controlo)
  - » Pacotes de dados perdidos ou duplicados (campo de numeração do pacote)
  - » Recuperação em caso de erro – por exemplo, terminar a ligação (DISC), estabelecê-la novamente (SET) e recomeçar o processo
- Registo de ocorrências
  - » Número de tramas I (re)transmitidas / recebidas, número de ocorrências de *time-out*, número de REJ enviados / recebidos

Anexos

Transmissão série assíncrona

- Cada caracter é delimitado por
  - *Start* bit
  - *Stop* bit (tipicamente 1 ou 2)
- Cada caracter é constituído por 8 bits (D0 – D7)
- Paridade
  - Par – número par de 1s
  - Ímpar – número ímpar de 1s
  - Inibida (bit D7 usado para dados) – opção adoptada no protocolo a implementar
- Taxa de transmissão: 300 a 115200 bit/s



Sinais RS-232

- Protocolo de nível físico entre computador ou terminal (DTE) e modem (DCE)
  - » DTE (*Data Terminal Equipment*)
  - » DCE (*Data Circuit-Terminating Equipment*)

TABLE 10.4. RS-232 SIGNALS

Name	Pin number 25-pin	Pin number 9-pin	Direction (DTE↔DCE)	Function (as seen by DTE)	
TD	2	3	→	transmitted data	} data pair
RD	3	2	←	received data	
RTS	4	7	→	request to send (= DTE ready)	} handshake pair
CTS	5	8	←	clear to send (= DCE ready)	
DTR	20	4	→	data terminal ready	} handshake pair
DSR	6	6	←	data set ready	
DCD	8	1	←	data carrier detect	} enable DTE input
RI	22	9	←	ring indicator	
FG	1	–		frame ground (= chassis)	
SG	7	5		signal ground	

Conectores DB25 e DB9

Sinal activo

Sinais de controlo (> + 3 V)
Sinais de dados (< - 3 V)

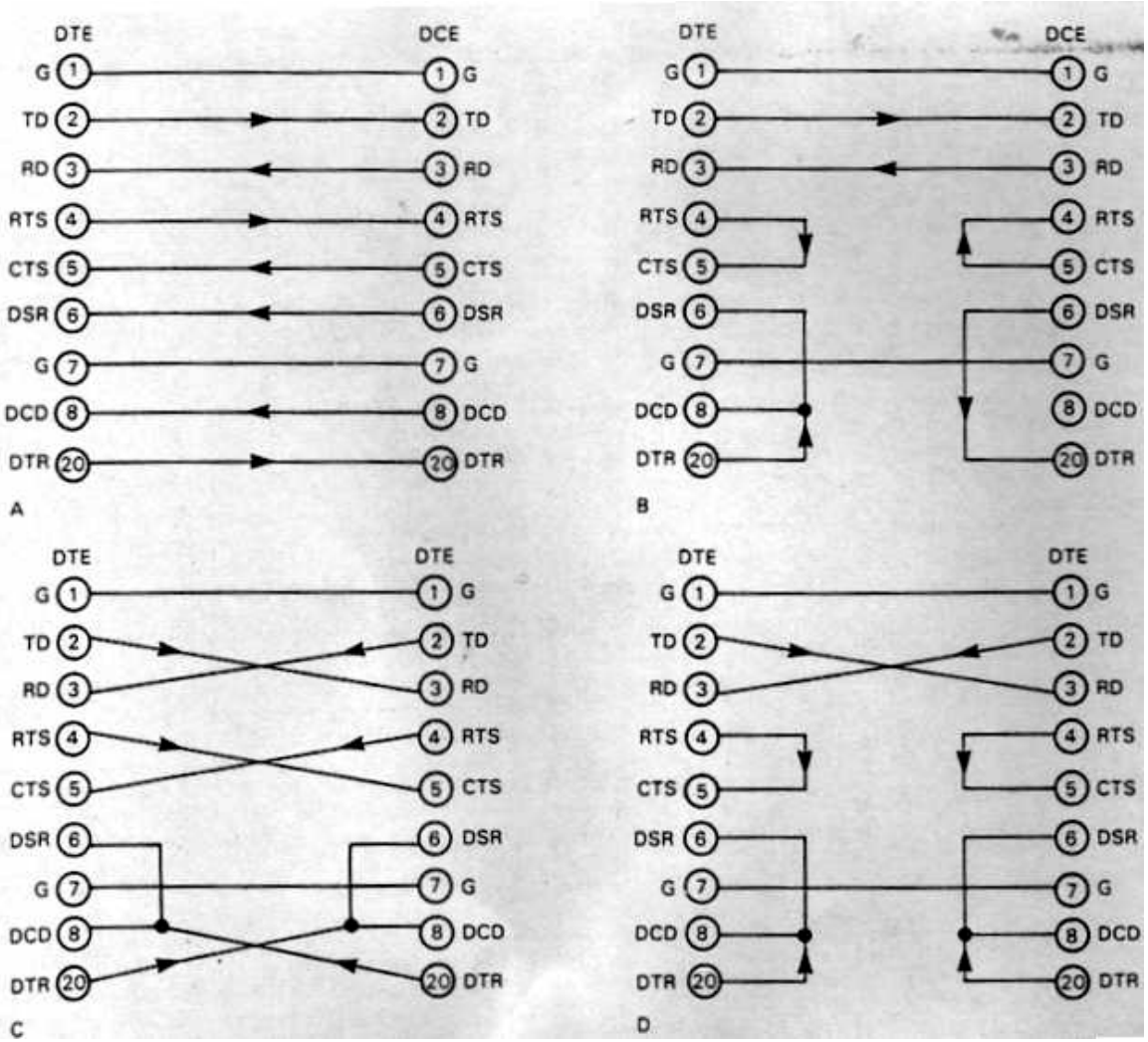
DTR (Data Terminal Ready) – Computador ligado
DSR (Data Set Ready) – Modem ligado

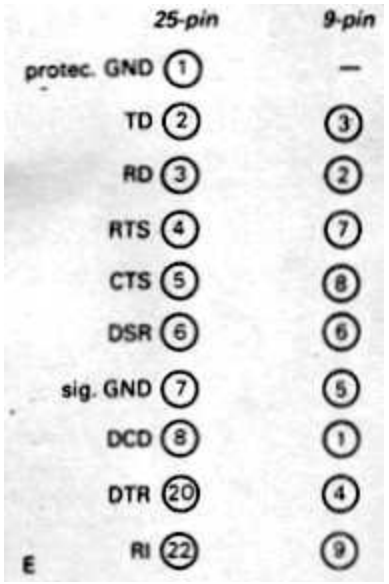
DCD (Data Carrier Detected) – Modem detecta portadora na linha telefónica
RI (Ring Indicator) – Modem detecta *ring*

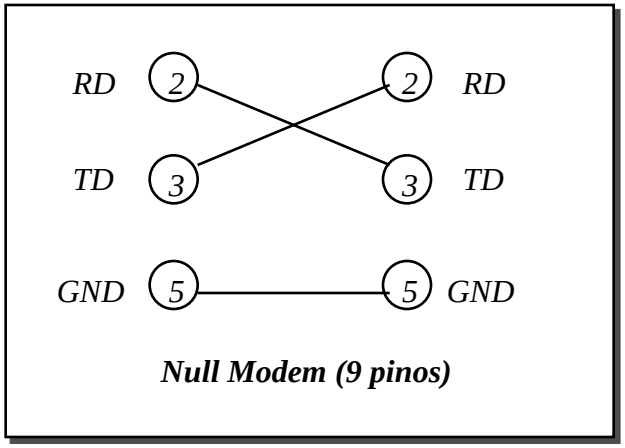
RTS (Request to Send) – Computador pronto a comunicar
CTS (Clear To Send) – Modem pronto a comunicar

TD (Transmit data) – Transmissão de dados
RD (Receive data) – Recepção de dados

Ligações entre equipamentos







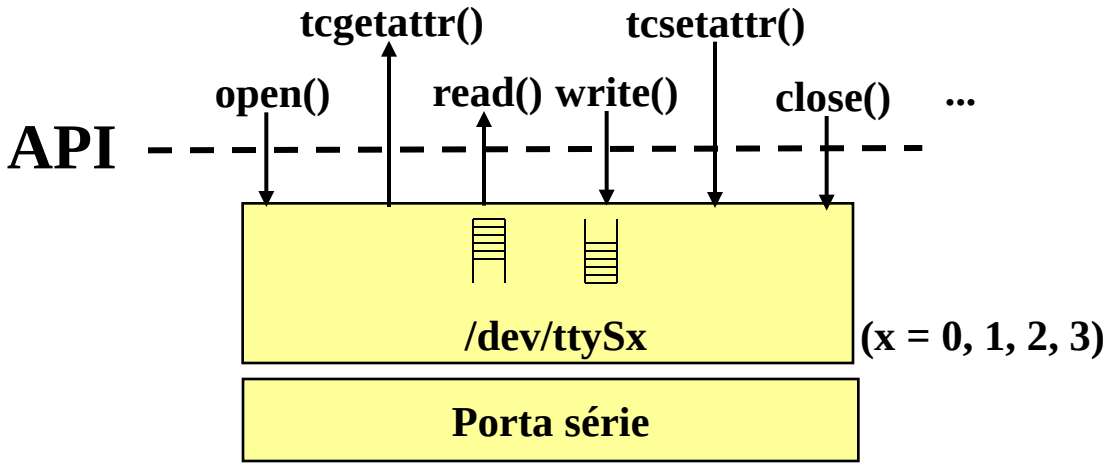
Null Modem (9 pinos)

Drivers Unix

- Características
  - *Software* que gere um controlador de *hardware*
  - Conjunto de rotinas de baixo nível com execução privilegiada
  - Residentes em memória (fazem parte do *kernel*)
  - Interrupção de *hardware* associada
- Método de acesso
  - Mapeados no sistema de ficheiros Unix (/dev/hda1, /dev/ttyS0)
  - Serviços oferecidos são semelhantes aos dos ficheiros (*open*, *close*, *read*, *write*)
- Tipos de *drivers*
  - Caracter
    - Leitura e escrita no controlador feita em múltiplos de caracteres
    - Acesso directo (dados não são guardados em *buffers*)
  - Bloco
    - Leitura/escrita em múltiplos de um bloco (bloco = 512 ou 1024 octetos)
    - Dados guardados em *buffers* e acesso aleatório
  - Rede
    - Leitura e escrita de pacotes de dados de comprimento variável
    - Interface de *sockets*

Driver da porta série – API

API – ***A****pplication* ***P****rogramming* ***I****nterface*



Algumas funções da API

int **open** (DEVICE, O_RDWR); /*retorna um descritor para ficheiro*/
int **read** (int descritorFicheiro, char * buffer, int numChars); /*retorna o número de caracteres lidos*/
int **write** (int descritorFicheiro, char * buffer, int numChars); /*retorna o número de caracteres escritos*/

int **close** (int descritorFicheiro);

int **tcgetattr** (int descritorFicheiro, struct termios *termios_p);
int **tcflush** (int descritorFicheiro, int selectorFila); /*TCIFLUSH, TCOFLUSH ou TCIOFLUSH*/
int **tcsetattr** (int descritorFicheiro, int modo, struct termios *termios_p);

Driver da porta série – API

Estrutura de dados *termios* – permite configurar e guardar todos os parâmetros de configuração da porta série

```
struct termios {
    tcflag_t  c_iflag;    /*flags de configuração da recepção*/
    tcflag_t  c_oflag;    /*flags de configuração da transmissão*/
    tcflag_t  c_cflag;    /*flags de controlo*/
    tcflag_t  c_lflag;    /*flags de configuração local*/
    cc_t      c_line;     /*não usado */
    cc_t      c_cc[NCCS]  /*caracteres de controlo; NCCS = 19*/
};
```

Exemplo:

```
#define BAUDRATE B38400
struct termios newtio;

/* CS8:     8n1 (8 bits, sem bit de paridade,1 stopbit)*/
/* CLOCAL:  ligação local, sem modem*/
/* CREAD:   activa a recepção de caracteres*/
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

/* IGNPAR:  Ignora erros de paridade*/
/* ICRNL:   Converte CR para NL*/
newtio.c_iflag = IGNPAR | ICRNL;

newtio.c_oflag = 0;  /*Saída não processada*/

/* ICANON:  activa modo de entrada canónico */
newtio.c_lflag = ICANON;
```

Tipos de receção na porta série

- Canónica
  - » *read*( ) retorna apenas linhas completas (terminadas por ASCII LF, EOF, EOL)
  - » Utilizada nos terminais

- Não canónica
  - » *read* ( ) retorna até um número máximo de caracteres
  - » Permite configurar o tempo máximo entre leitura de caracteres
  - » Adequada para leitura de grupos de caracteres

- Assíncrona
  - » *read*( ) retorna imediatamente
  - » Utilização de um *signal handler*

Exemplos de programas

Receção canónica

```
main() {

int fd,c, res;
struct termios oldtio,newtio;
char buf[255];

fd = open(/dev/ttyS1,O_RDONLY|O_NOCTTY);
tcgetattr(fd,&oldtio);

bzero(&newtio, sizeof(newtio));
newtio.c_cflag = B38400|CS8|CLOCAL|CREAD;
newtio.c_iflag = IGNPAR|ICRNL;
newtio.c_oflag = 0;
newtio.c_lflag = ICANON;
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);

res = read(fd,buf,255);

tcsetattr(fd,TCSANOW,&oldtio);
close(fd);
}
```

Receção não canónica

```
main() {

int fd,c, res;
struct termios oldtio,newtio;
char buf[255];

fd = open(argv[1], O_RDWR | O_NOCTTY );
tcgetattr(fd,&oldtio);

bzero(&newtio, sizeof(newtio));
newtio.c_cflag = B38400 | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0;
newtio.c_lflag = 0;
newtio.c_cc[VTIME] = 0; /* temporizador entre
                           caracteres*/
newtio.c_cc[VMIN] = 5;  /* bloqueia até ler 5
                           caracteres */
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);

res = read(fd,buf,255); /*pelo menos 5 caracteres*/

tcsetattr(fd,TCSANOW,&oldtio);
close(fd);
}
```

Exemplos de programas

Receção assíncrona

```
void signal_handler_IO (int status);   /*definição
signal handler */
main() {
   /*…declaração de variáveis e abertura do dispositivo
série...*/
    saio.sa_handler = signal_handler_IO;
    saio.sa_flags = 0;
    saio.sa_restorer = NULL;    /*obsoleto*/
    sigaction(SIGIO,&saio,NULL);
    fcntl(fd, F_SETOWN, getpid());
    fcntl(fd, F_SETFL, FASYNC);
   /*… configuração da porta através da estrutura
termios ...*/
  while (loop) {
      write(1, ".", 1);usleep(100000);
      /* após o sinal SIGIO, wait_flag = FALSE, existem
dados na entrada para o read */
      if (wait_flag==FALSE) {
        read(fd,buf,255); wait_flag = TRUE; /*aguardar
novos dados*/
      }
    }
    /* … configurar a porta com os valores iniciais e
fechar ...*/
}
void signal_handler_IO (int status) { wait_flag =
FALSE; }
```

Receção múltipla

```
main(){
int fd1, fd2;  /*input sources 1 and 2*/
fd_set readfs; /*file descriptor set */
int maxfd, loop = 1; int    loop=TRUE;
     /* open_input_source opens a device, sets the
port correctly, and returns a file descriptor */
     fd1 = open_input_source("/dev/ttyS1");   /*
COM2 */
     fd2 = open_input_source("/dev/ttyS2");   /*
COM3 */
     maxfd = MAX (fd1, fd2)+1;  /*max bit entry
(fd) to test*/
     while (loop) {     /* loop for input */
       FD_SET(fd1, &readfs);  /* set testing for
source 1 */
       FD_SET(fd2, &readfs);  /* set testing for
source 2 */
       /* block until input becomes available */
       select(maxfd, &readfs, NULL, NULL, NULL);
       if (FD_ISSET(fd1))        /* input from
source 1 available */
         handle_input_from_source1();
       if (FD_ISSET(fd2))        /* input from
source 2 available */
         handle_input_from_source2();
     }
 }
```