Protocolo de Ligação de Dados

(1º Trabalho Laboratorial)

FEUP/DEEC
Redes de Computadores
MIEEC – 2009/10
José Ruela

Descrição do Trabalho

- Objectivos
  - » Implementar um protocolo de ligação de dados, de acordo com a especificação proposta
  - » 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
  - » Duas opções
    - – Grupos com 3 elementos – cada grupo realiza o emissor **e** o receptor
    - – Grupos com 4 elementos constituídos por 2 sub-grupos de 2 elementos (com avaliação independente) – um sub-grupo realiza o emissor e o outro realiza o receptor, devendo garantir interoperação

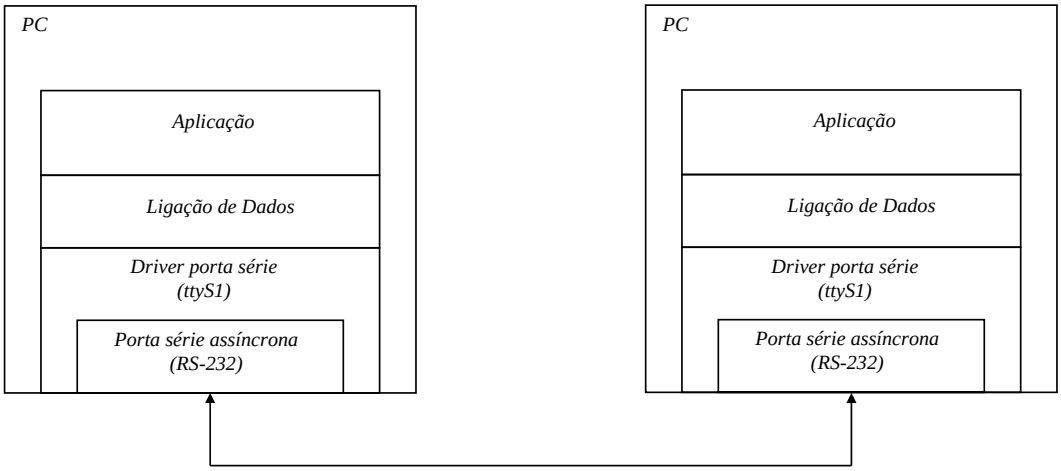
- Elementos de Avaliação
  - » Participação nas aulas (avaliação contínua)
  - » Apresentação e demonstração do trabalho
  - » Relatório final

Demonstração e Relatório – datas

- Apresentação e demonstração do trabalho
  - » Nas aulas teórico-prática (laboratoriais) da semana de 2 a 6 de Novembro de 2009

- Entrega do relatório
  - » Por email para o docente da aula teórico-prática respectiva
  - » Material a entregar
    - – Relatório, com código fonte em anexo
  - » Data limite: 6 de Novembro de 2009

Configuração de Teste



Protocolos de Ligação de Dados – funções típicas

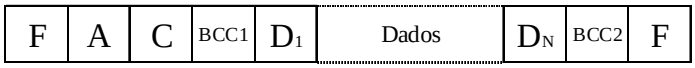
- Objectivo
  - » Fornecer um serviço de comunicação de dados fiável entre dois sistemas directamente ligados por um meio de transmissão
- Funções genéricas
  - » Sincronismo (delimitação) de trama – dados organizados em tramas (*framing*)
    - – Principais alternativas: caracteres / *flags* para delimitar início e fim de trama
    - – Tamanho dos dados pode ser implícito ou indicado explicitamente no cabeçalho
  - » Estabelecimento / terminação da ligação
  - » Numeração de tramas
    - – Permite confirmação de tramas, controlo de erros e controlo de fluxo
      - ◆ Caso mais simples – numeração módulo 2 (números de sequência: 0 e 1)
  - » 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*)
    - – Confirmação negativa (tramas fora de sequência) / retransmissão a pedido do receptor
    - – Temporizadores (*time-out*) / retransmissão decidida pelo emissor
    - – Retransmissões podem originar duplicados (que devem ser detectados e eliminados)
  - » Controlo de fluxo

Protocolo de Ligação de Dados proposto

- O protocolo a implementar reúne um conjunto de características que se encontram em 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 de controlo ou um pacote de dados gerado pela aplicação, mas o seu conteúdo não é processado pelo protocolo de ligação de dados)
  - » 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 detector de erros
    - – Nas tramas S e U a protecção da trama coincide com a protecção do cabeçalho
    - – Nas tramas I existe protecção 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)

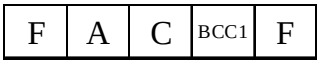
Formato e tipos de tramas

» Tramas de Informação (I)



F	**Flag**		
A	**Campo de Endereço**		
C	**Campo de Controlo**	**0 0 0 0 0 0 S 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 0 1 1**	
	DISC (disconnect)	**0 0 0 0 1 0 1 1**	
	UA (unnumbered acknowledgment)	**0 0 0 0 0 1 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 Protecção (cabeçalho)**		

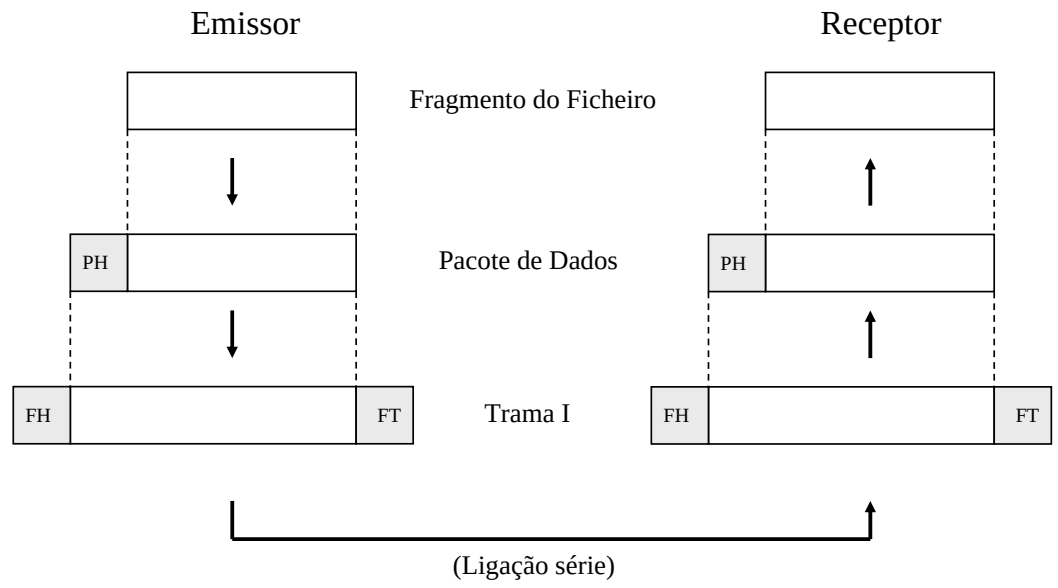
Pacotes e Tramas

» 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 transmite 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 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, incluindo o próprio BCC

Tramas – processamento de tramas recebidas

» 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 *time-out*, devem ser efectuadas três tentativas de retransmissão

Transparência

» A transmissão entre os dois computadores é baseada numa técnica designada por transmissão assíncrona
  – Esta técnica caracteriza-se pela transmissão de caracteres (constituídos por um número de bits que pode ser configurado) delimitados por bits *Start* e *Stop*
  – Alguns protocolos usam caracteres de um código (por exemplo ASCII) para delimitar e identificar os campos que constituem as tramas e para suportar a execução dos mecanismos protocolares
    ◆ A transmissão de dados de foram transparente (independente do código usado pelo protocolo) obriga recorrer a mecanismos de escape

» O protocolo a implementar não se baseia na utilização de qualquer código, pelo que os caracteres transmitidos devem ser interpretados como octetos (*bytes*), podendo ocorrer qualquer uma das 256 combinações possíveis

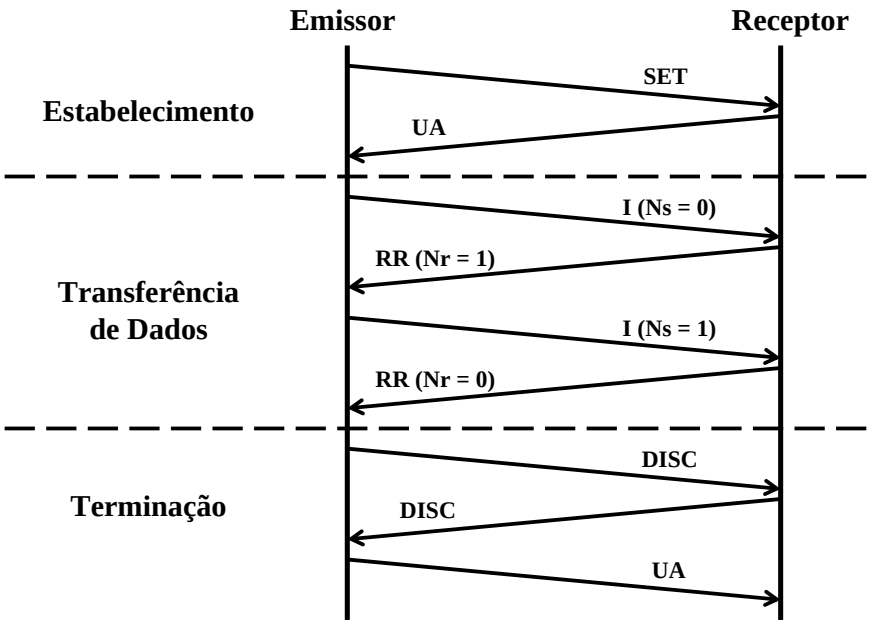
» Para evitar o falso reconhecimento de uma *flag* no interior de uma trama, é necessário um mecanismo que garanta transparência
  – Em HDLC é usado um mecanismo de *bit stuffing*
  – No protocolo PPP é adoptado o formato da trama HDLC (delimitação por *flags*), a trama é constituída por um número inteiro de octetos e é usado um mecanismo de *byte stuffing*

Transparência – mecanismo de byte stuffing

» No protocolo a implementar adopta-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 a substituição de qualquer octeto pela correspondente sequência de escape

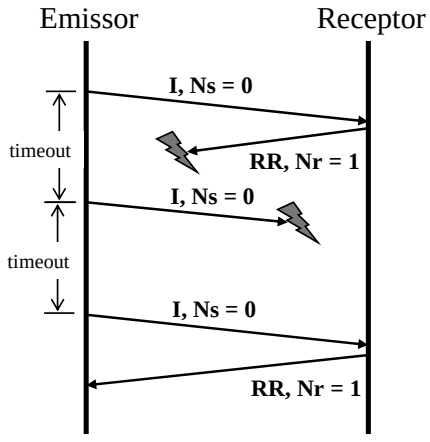
Fases do Protocolo de Ligação de Dados

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

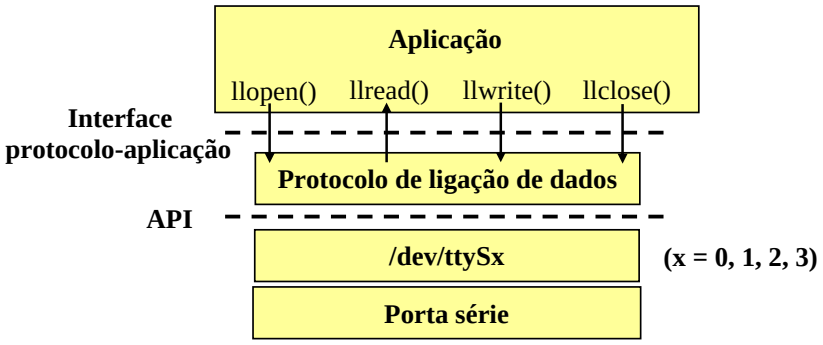


Transferência de Dados

- 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
  - » Após recepção de confirmação negativa (REJ)
  - » Se ocorrer *time-out*, devido à perda da trama I enviada ou da sua confirmação
  - » Três tentativas de retransmissão
- Protecção da trama
  - » Geração e verificação do(s) campo(s) de protecção (BCC)



Interface Protocolo-Aplicação



Interface Protocolo-Aplicação

- Exemplos de estruturas de dados
  - » Aplicação

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

  - » Protocolo

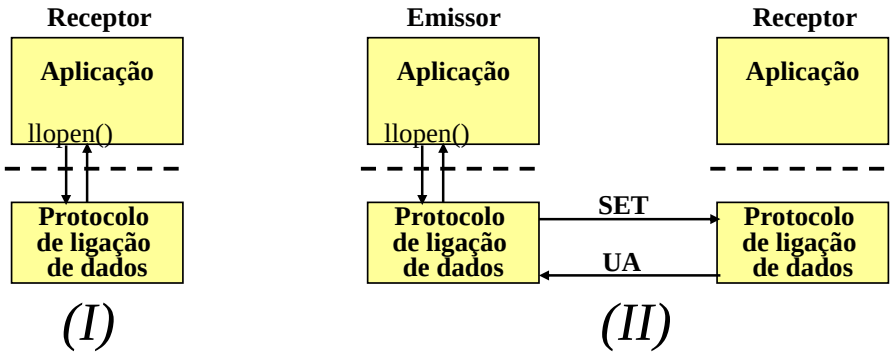
```
struct linkLayer {
  char port[20];           /*Dispositivo /dev/ttySx, x = 0, 1, 2, 3*/
  int baudRate;                  /*Velocidade de transmissão*/
  unsigned int sequenceNumber;   /*Número de sequência da trama: 0, 1*/
  unsigned int timeout;           /*Valor do temporizador: 1 s*/
  unsigned int numTransmissions; /*Número de tentativas em caso de
                                   falha*/
  char frame[MAX_SIZE];           /*Trama*/
}
```

Interface Protocolo-Aplicação

int llopen(int porta, TRANSMITTER | RECEIVER)

argumentos
- porta: COM1, COM2, ...
- flag: TRANSMITTER ou RECEIVER

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



Interface Protocolo-Aplicação

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

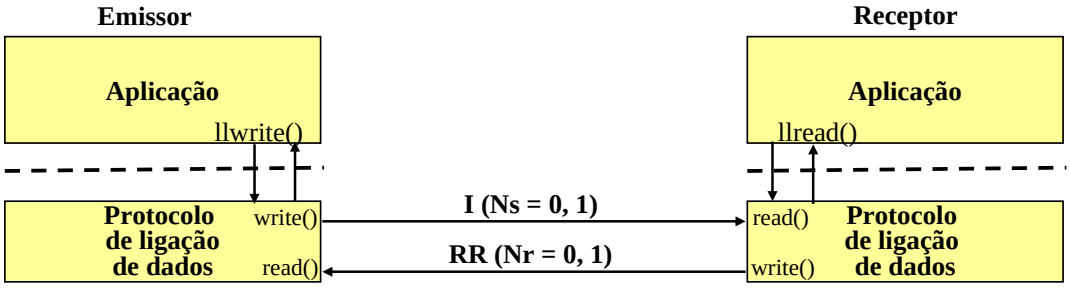
argumentos
- fd: identificador da ligação de dados
- buffer: *array* de caracteres a transmitir
- length: comprimento do *array* de caracteres

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

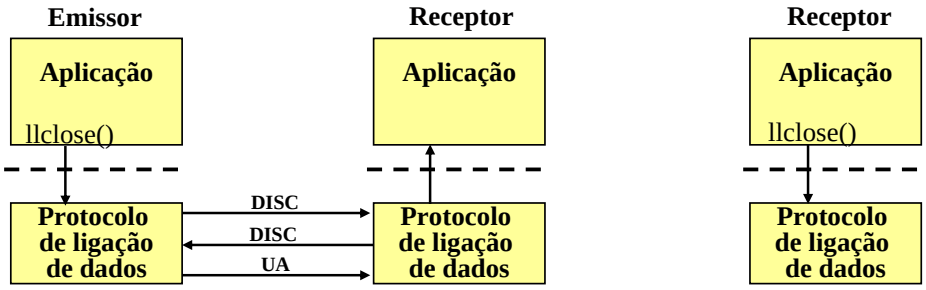
int llclose(int fd)

argumentos

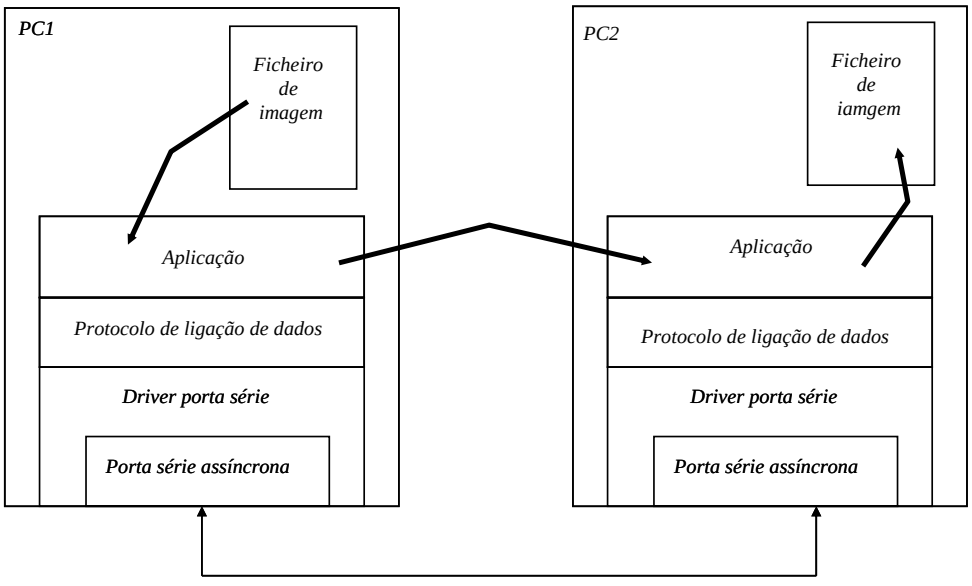
- fd: identificador da ligação de dados

retorno

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



Aplicação de Teste



Aplicação de Teste

- 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*) poderá repetir a informação contida no pacote de início de transmissão
- Os Pacotes de Dados contêm obrigatoriamente um campo com um número de sequência (um octeto) e um campo que indica o tamanho do respectivo campo de dados (dois octetos)
  - » Este tamanho depende do tamanho máximo do campo de Informação das tramas I
  - » Estes campos permitem verificações adicionais em relação à integridade dos dados

Pacotes do Nível de Aplicação

- Pacote de Dados

C	N	L_2	L_1	P_1	.....	P_K

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

C	T_1	L_1	V_1	T_2	L_2	V_2	.....

  - » C – campo de controlo (1 – *start*; 2 – *end*)
  - » Cada parâmetro (tamanho ou nome do ficheiro) é codificado na forma TLV (*Type*, *Length*, *Value*)
    - – T (um octeto) – indica qual o parâmetro (0 – tamanho do ficheiro, 1 – nome do ficheiro, outros – 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

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
- Demonstração
- Penalizações
  - » Por cada semana de atraso: 20%

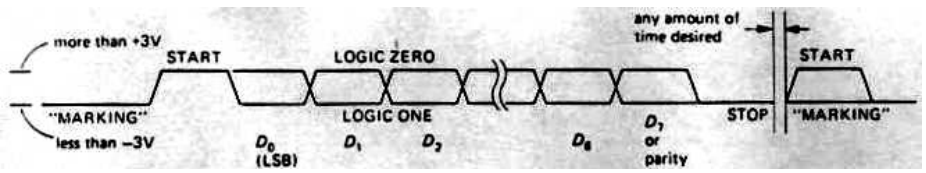
Elementos de valorização

- 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
  - » Alteração com probabilidades pré-definidas e de forma independente dos valores dos BCC do cabeçalho e do campo de dados (simulação de erro)
- Implementação de REJ
- Verificação da integridade dos dados pela Aplicação
  - » Tamanho do ficheiro recebido
  - » 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 carácter é delimitado por
  - *Start* bit
  - *Stop* bit (tipicamente 1 ou 2)
- Cada carácter 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	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)

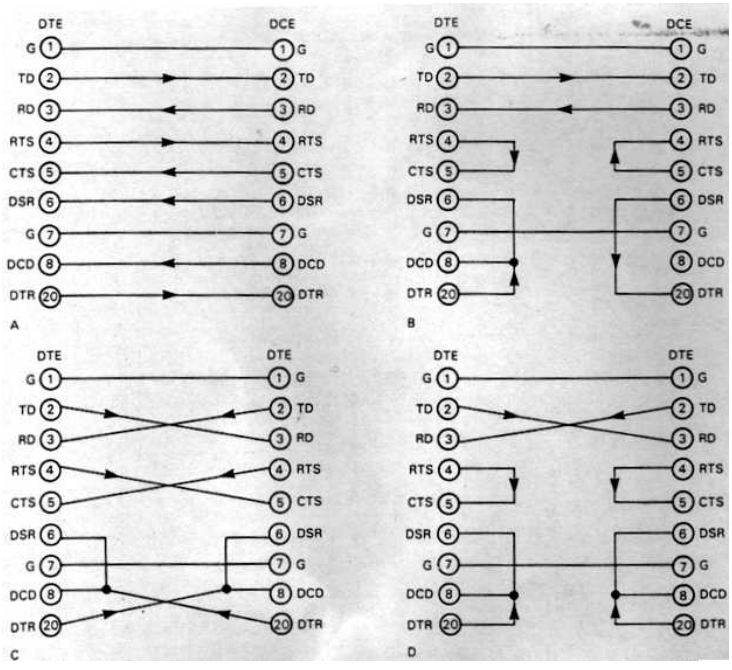
DTR (Data Terminal Ready) – Computador ligado
DSR (Data Set Ready) – Modem ligado

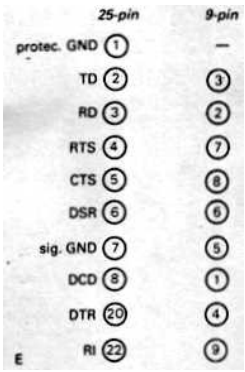
DCD (Data Carrier Detected) – Modem detecta portadora na linha telefónica
RI (Ring Indicator) – Modem detecta *ring*

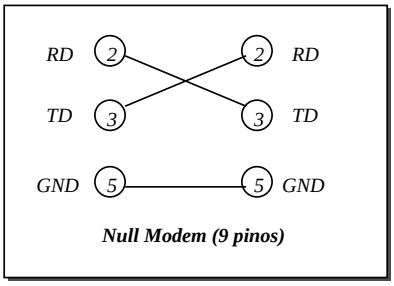
RTS (Request to Send) – Computador pronto a comunicar
CTS (Clear To Send) – Modem pronto a comunicar

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

Ligações entre Equipamentos







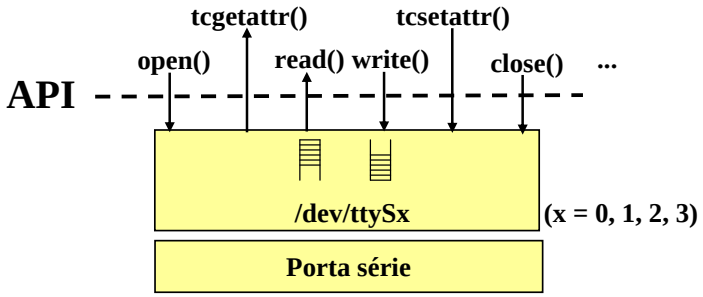
Null Modem (9 pinos)

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*
  - Carácter
    - 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, desactiva o eco e não envia
            sinais ao programa*/
newtio.c_lflag = ICANON;
```


Tipos de Recepção 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 caracteres
  - » Adequada para leitura de grupos de caracteres
- Assíncrona
  - » *read*( ) retorna imediatamente e envia um sinal à aplicação quando termina
  - » Utilização de um *signal handler*

Exemplos de programas

Recepção 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);
}
```

Recepção 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

Recepção 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; }
```

Recepção 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();
     }
  }
```