Protocolo de Ligação Lógica

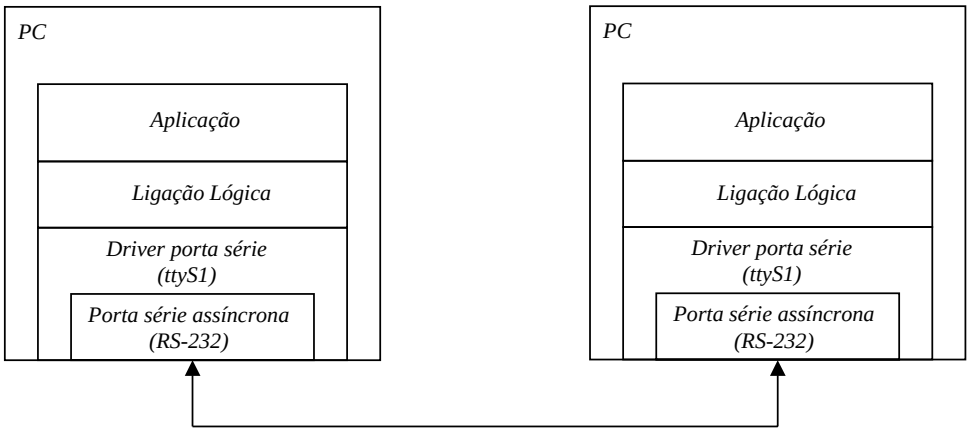
(Trabalho Laboratorial)

FEUP/DEEC/CDRC I – 2002/03
MPR/JAR

Descrição do Trabalho

- Objectivos
  - » Implementar um protocolo de ligação lógica
  - » Testar o protocolo com uma aplicação de transferência de ficheiros
- Ambiente de desenvolvimento
  - » PC com Unix (LINUX)
  - » Linguagem de programação - C
  - » Portas série RS-232 (comunicação assíncrona)
- Avaliação
  - » Contínua
  - » Demonstração
  - » Relatório

Configuração de Teste



Código ASCII

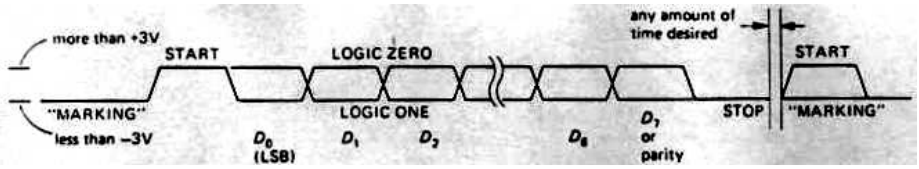
American
Standard
Code for
Information
Interchange

TABLE 10.3. ASCII CODES

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

Transmissão Série Assíncrona

- » Cada caracter é delimitado por
  - – Start bit
  - – Stop bit (1 ou 2)
- » Paridade
  - – Par – número par de 1s
  - – Ímpar – número ímpar de 1s
  - – Nenhuma (bit D7 usado para dados)
- » 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*

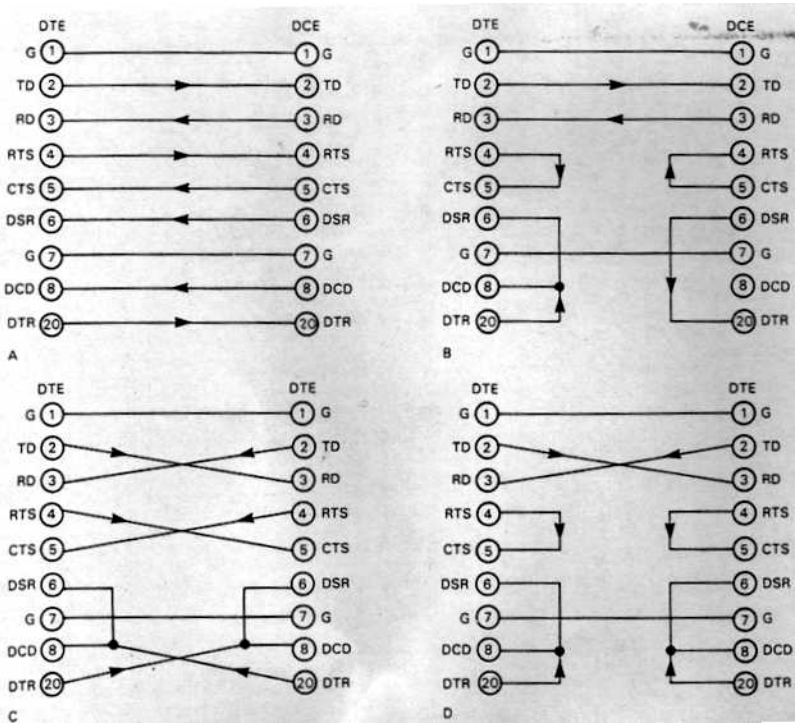
RTS (Request to Send) - Computador pronto a comunicar

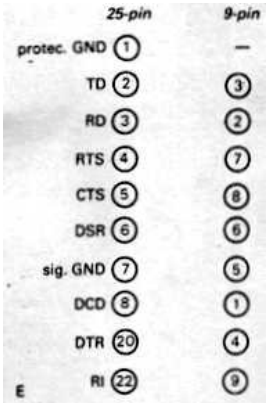
CTS (Clear To Send) - Modem pronto a comunicar

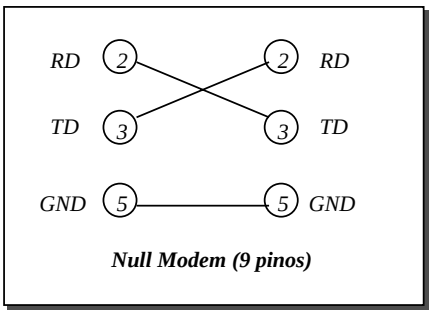
TD (Transmit data) - Transmissão de dados

RD (Receive data) - Recepção de dados

Ligações entre Equipamentos







Drivers Unix

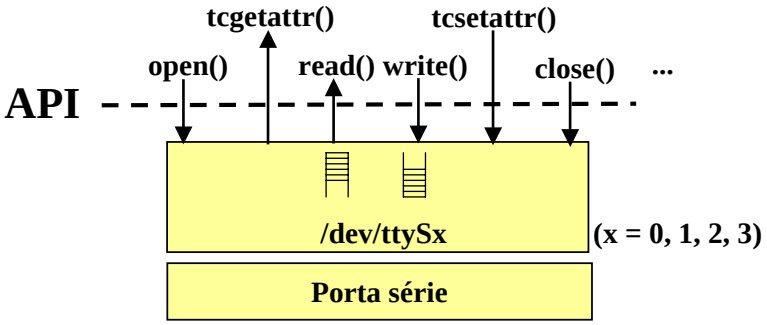
» Características
- *Software* que gere um controlador de *hardware*
- Conjunto de rotinas de baixo nível com execução privilegiada
- Residentes em memória (fazem parte do *kernel*)
- Interrupção de *hardware* associada

» 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, 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();
    }
  }
```


Protocolo de Ligação Lógica

- Objectivo
  - » Fornecer serviços à camada protocolar superior
    - – Exemplo: serviço confirmado (fiável) orientado às ligações
- Funções
  - » Sincronismo de trama – dados organizados em tramas (*framing*)
    - – Alternativas: caracteres / *flags* de início e fim
    - – Tamanho dos dados implícito ou indicado explicitamente
  - » Estabelecimento / terminação da ligação
  - » Confirmação de recepção
  - » Controlo de erro
    - – Confirmação negativa / pedido de retransmissão
    - – Temporizadores
    - – Números de sequência permitem detectar omissões e duplicados
      - Caso mais simples – números de sequência 0 e 1
  - » Controlo de fluxo (exemplo: *Stop-and-Wait*, Janela)

Protocolo de Ligação Lógica

» Formato das tramas de Informação (I)

SYN	SYN	SOH	C	L	BCC	D_1	Dados	D_N	BCC

C – Campo de Controlo **0 0 0 N(s) 0 0 0 0**
L – Comprimento do campo de dados D_1….D_N (N octetos)

» Formato das restantes tramas

SYN	SYN	SOH	C	L	BCC

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 0 N(r) 0 0 0 1**
REJ (reject / negative ACK)	**0 0 0 N(r) 0 1 0 1**
RNR (receiver not ready) – opcional	**0 0 0 N(r) 1 0 0 1**

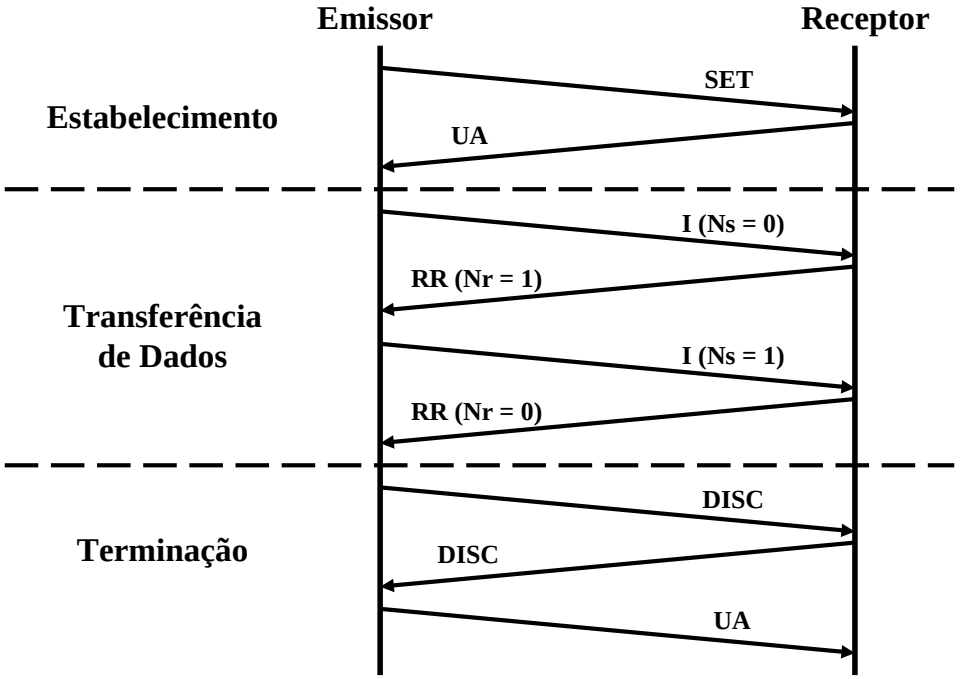
L – Comprimento do campo de dados (0)

Protocolo de Ligação Lógica

» O Emissor deve gerar tramas iniciadas com dois ou mais caracteres SYN
» O Receptor reconhece o início de uma trama após receber um ou mais caracteres SYN, seguido(s) de um caracter SOH
» Todas as tramas têm um cabeçalho com formato comum
  – C (Campo de Controlo) – define o tipo de trama e transporta números de sequência em tramas I e em tramas de Supervisão (RR, REJ e RNR)
  – L (Campo de Comprimento) – define o tamanho do campo de dados (não nulo nas tramas I, nulo nas restantes)
  – BCC (*Block Check Character*) – detecção de erros baseada em paridade par sobre cada um dos bits de C, L e BCC
» As tramas I têm um campo de dados protegido por um BCC próprio (paridade par sobre cada um dos bits dos octetos de dados e do BCC)
» Tramas com cabeçalho errado são ignoradas, sem qualquer acção
» Tramas I com cabeçalho correcto mas com erros nos dados são descartadas e confirmadas negativamente (pedido de retransmissão, antes de *time-out*)
» Tramas I, SET e DISC são protegidas por um temporizador; em caso de *time-out*, devem ser efectuadas duas tentativas de retransmissão

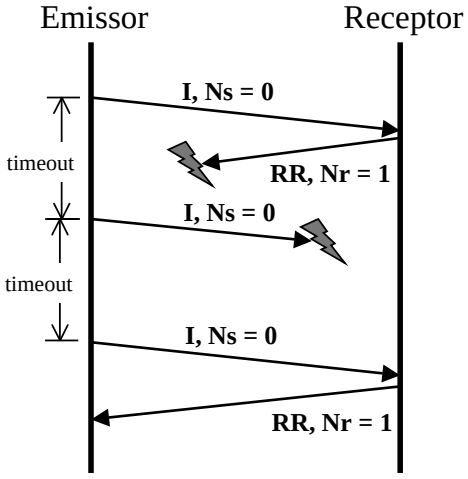
Protocolo de Ligação Lógica

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

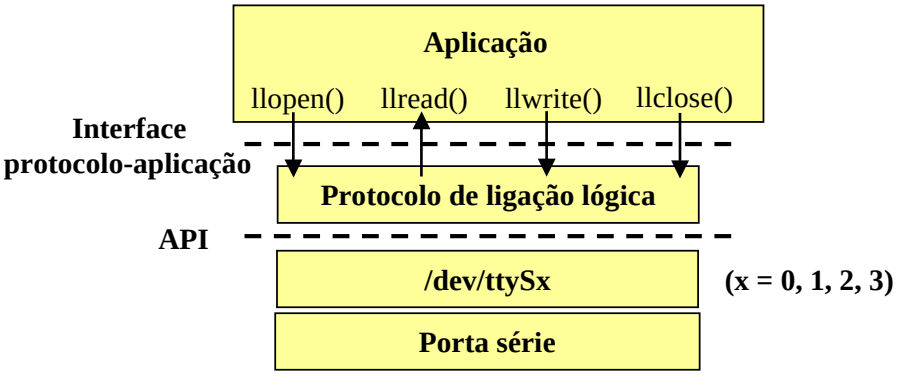


Protocolo de Ligação Lógica

- Controlo de fluxo
  - » *Stop-and-Wait*
- Temporizador
  - » activado após o envio de uma trama I, SET ou DISC
  - » desactivado após recepção de uma resposta sem erro
  - » quando excedido (*time-out*) obriga a retransmissão
- Retransmissão
  - » Recepção de confirmação negativa
  - » Temporizador excedido
    - – perda da trama enviada ou da sua confirmação
  - » Duas tentativas de retransmissão
- Protecção do cabeçalho e dados
  - » O emissor calcula o OU exclusivo dos octetos a proteger; o resultado é colocado no octeto de protecção (BCC)
  - » O receptor faz a verificação



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

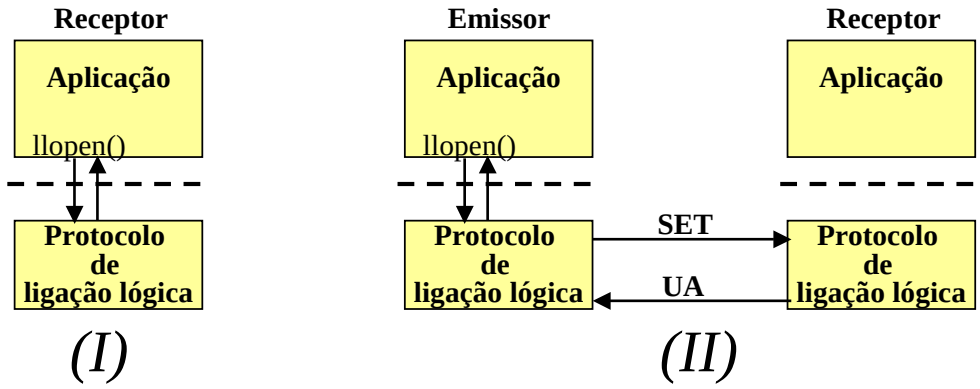
int llopen(int porta, TRANSMITTER | RECEIVER)

argumentos

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

retorno

- identificador de ligação lógica
- valor negativo em caso de erro



Interface Protocolo-Aplicação

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

argumentos

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

retorno

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

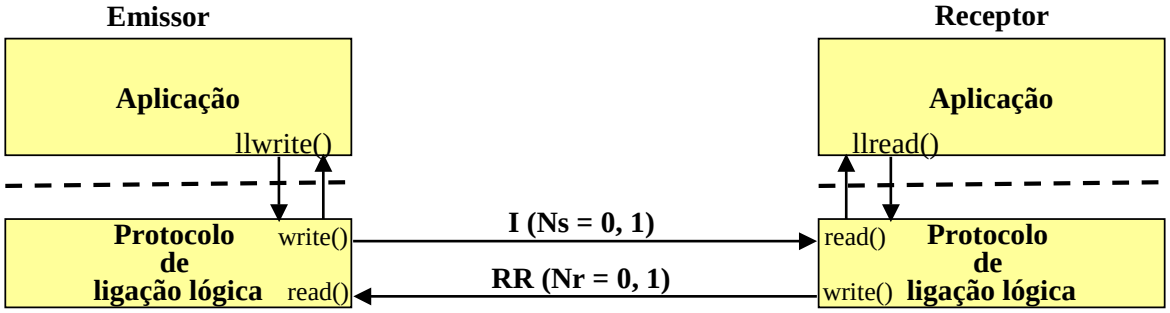
int llread(int fd, char * buffer)

argumentos

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

retorno

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



Interface Protocolo-Aplicação

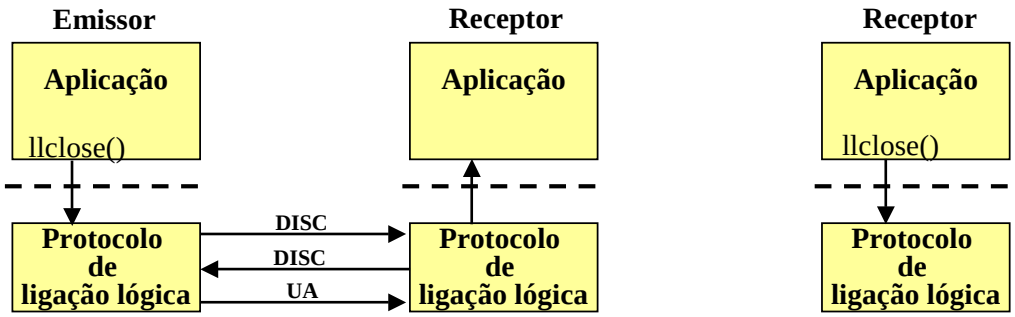
int llclose(int fd)

argumentos

- fd: identificador da ligação lógica

retorno

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



Aplicação de Teste