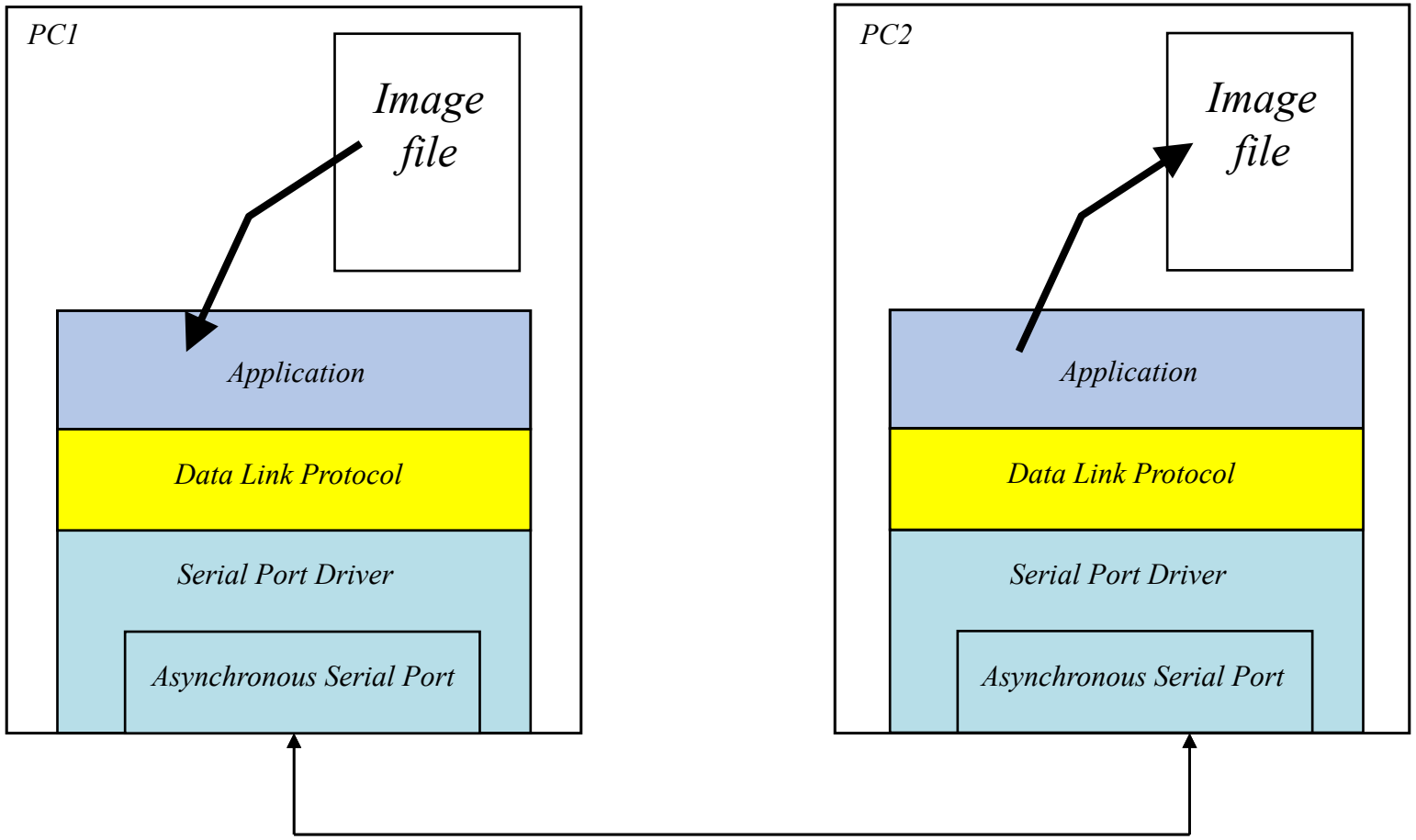
Redes de Computadores
Lab 2
2023-24, 2º S

Ana Aguiar

Lab 3 – Goals & Competences

- Goals
  - Develop and test a data communication protocol
    - Control frames vs data frames
    - Frame synchronisation, error robustness, error correction, ...
    - Stop & wait ARQ protocol
    - Modularity and layering
    - Performance evaluation
  - Consolidate knowledge about layering, functionality separation, state machines, link layer functionality
  - Develop C programming skills
  - Develop competences in Linux
- Competences
  - Understand principles of data communication
    - layering, interfaces, functionality separation, state machines
  - Compiling and debugging a distributed application in Linux environment
    - Basic Linux commands, gdb, make
  - Developing a distributed application in a team

Communication System Overview

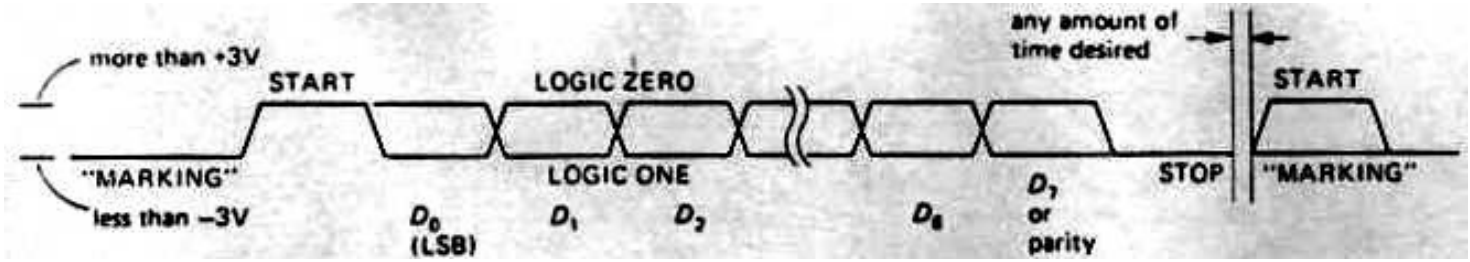


Data Link Protocols–Typical Functionality

- Goal
  - Provide reliable communication between two systems connected by a communication medium – in this case, the serial cable
  - Service provided to applications using the communication medium
- Typical Functionality
  - Frame synchronisation – data organised in frames
    - Characters/ flags to delimit start and end of frame
    - Data size may be implicit or indicated in the frame header
  - Connection establishment and termination
  - Frame numbering
    - Useful for frame acknowledgements, error control, flow control
  - Error control (e.g.: Stop-and-Wait, Go-back-N, Selective Repeat)
    - Acknowledgements after reception of a correct and ordered frame
    - Timers (time-out) – transmitter decides on retransmission
    - Negative acknowledgement (out of sequence frames) – receiver requests retransmission
    - Retransmissions may originate duplicates, which should be detected and eliminated
  - Flow control

Synchronous Serial Transmission

- Each character is delimited by
  - Start bit
  - Stop bit (typically 1 or 2)
- Each character consists of 8 bits (D0 – D7)
- Parity
  - Even – even number of 1s
  - Odd – uneven number of 1s
  - Inhibited (bit D7 used for data) – option adopted in this assignment
- Transmission rate: 300 to 115200 bit/s



RS-232 Signals

- Physical payer protocol between computer or terminal (DTE) and modem (DCE)
  - DTE (Data Terminal Equipment)
  - DCE (Data Circuit-Terminating Equipment)

TABLE 10.4. RS-232 SIGNALS

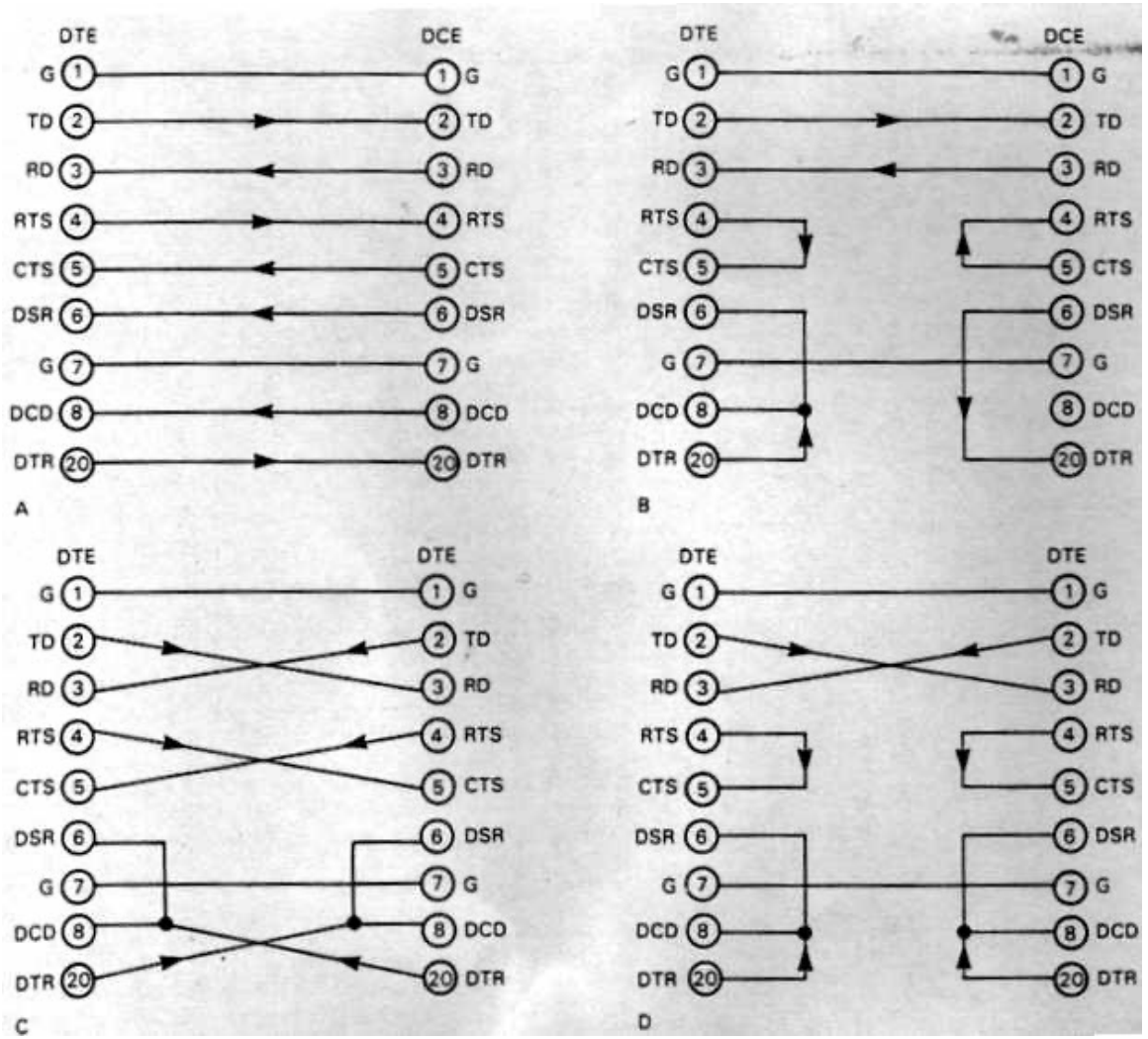
Name	Pin number 25-pin	Pin number 9-pin	Direction (DTE↔DCE)	Function (as seen by DTE)	
TD	2	3	→	transmitted data	} data pair
RD	3	2	←	received data	
RTS	4	7	→	request to send (= DTE ready)	} handshake pair
CTS	5	8	←	clear to send (= DCE ready)	
DTR	20	4	→	data terminal ready	} handshake pair
DSR	6	6	←	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	

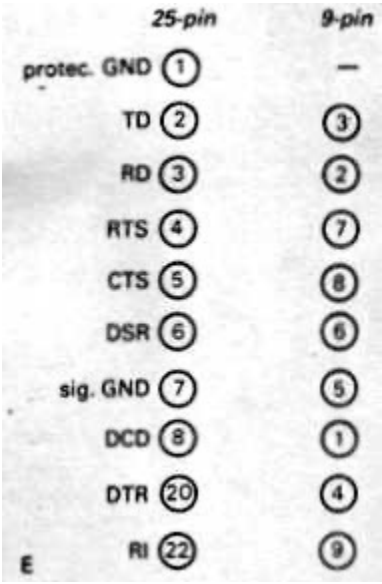
RS-232 Signals

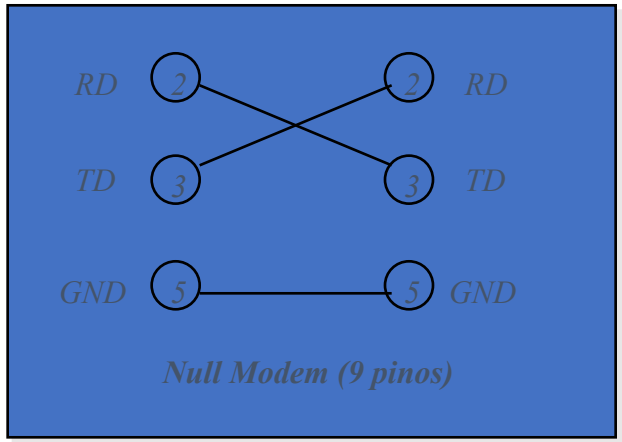
DB25 and DB9 connectors

- **Active signal**
  - Control signal (> + 3 V)
  - Data signal (< - 3 V)
- **DTR (Data Terminal Ready) –** Computer on
- **DSR (Data Set Ready) –** Modem on
- **DCD (Data Carrier Detected) –** Modem detects carrier on phone line
- **RI (Ring Indicator) –** Modem detects ringing
- **RTS (Request to Send) –** Computer ready to communicate
- **CTS (Clear To Send) –** Modem ready to send
- **TD (Transmit data) –** Data transmission
- **RD (Receive data) –** Data reception

Equipment Connections



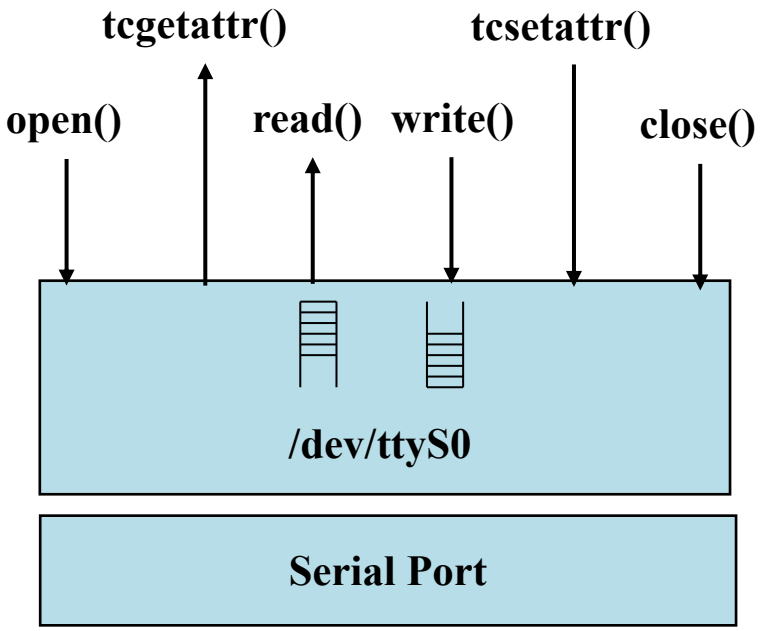




Unix Drivers

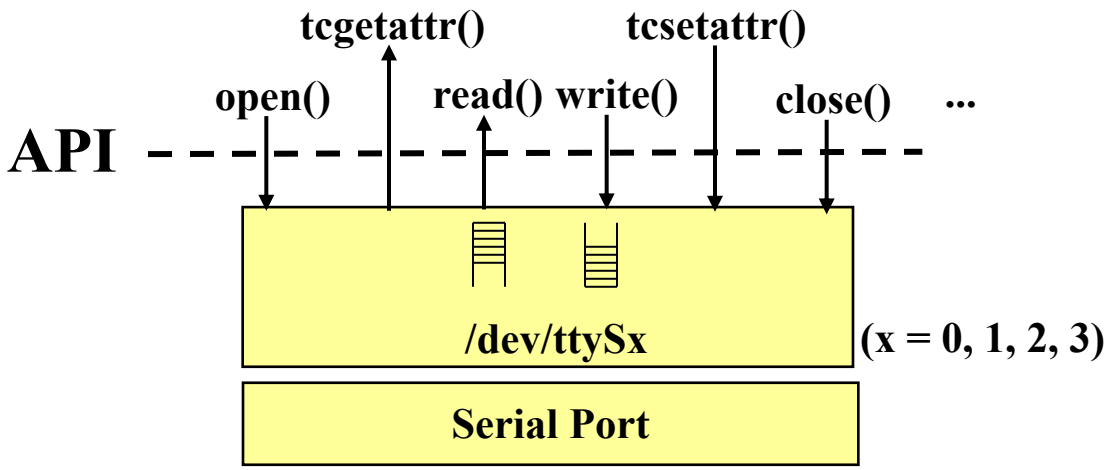
- Software that manages a hardware controler
  - Low level routines with previleged access
  - Reside in memory (part of the kernel)
  - Have associated hardware interrupt
- Access method
  - Mapped into Unix file system (/dev/hda1, /dev/ttyS0)
  - Services are similar to files (open, close, read, write)
- Driver types
  - Character
    - Read and write carried out multiples of a character
    - Direct access (data is not stored in buffers)
  - Block
    - Read/ write as multiples of a block of octets
    - Data stored in random access buffer
  - Network
    - Read and write variable size data packets
    - Socket interface

Serial Port Driver – API



Serial Port Driver API

API – ***Application Programming Interface***



Some API Functions

int **open** (DEVICE, O_RDWR); /* returns a file descriptot */

int **read** (int fileDescriptor, char * buffer, int numChars); /* returns nr characters read */

int **write** (int fileDescriptor, char * buffer, int numChars); /* returns nr characters written */

int **close** (int fileDescriptor);

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);

Serial Port Driver API

- termios data structure – enables configuring and saving all serial port parameters

```
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*/
};
```

Serial Port Driver API

- 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;
```

Serial Port Reception Types

- Canonic
  - read( ) returns only full lines (ended by ASCII LF, EOF, EOL)
  - Used for terminals

- Non-canonic
  - read( ) returns up to a maximum number of characters
  - Enables configuration of maximum number of characters
  - Adequate for reading groups of characters

- Asynchronous
  - read( ) returns immediately and send a signal to the application on return
  - Requires the use of a signal handler

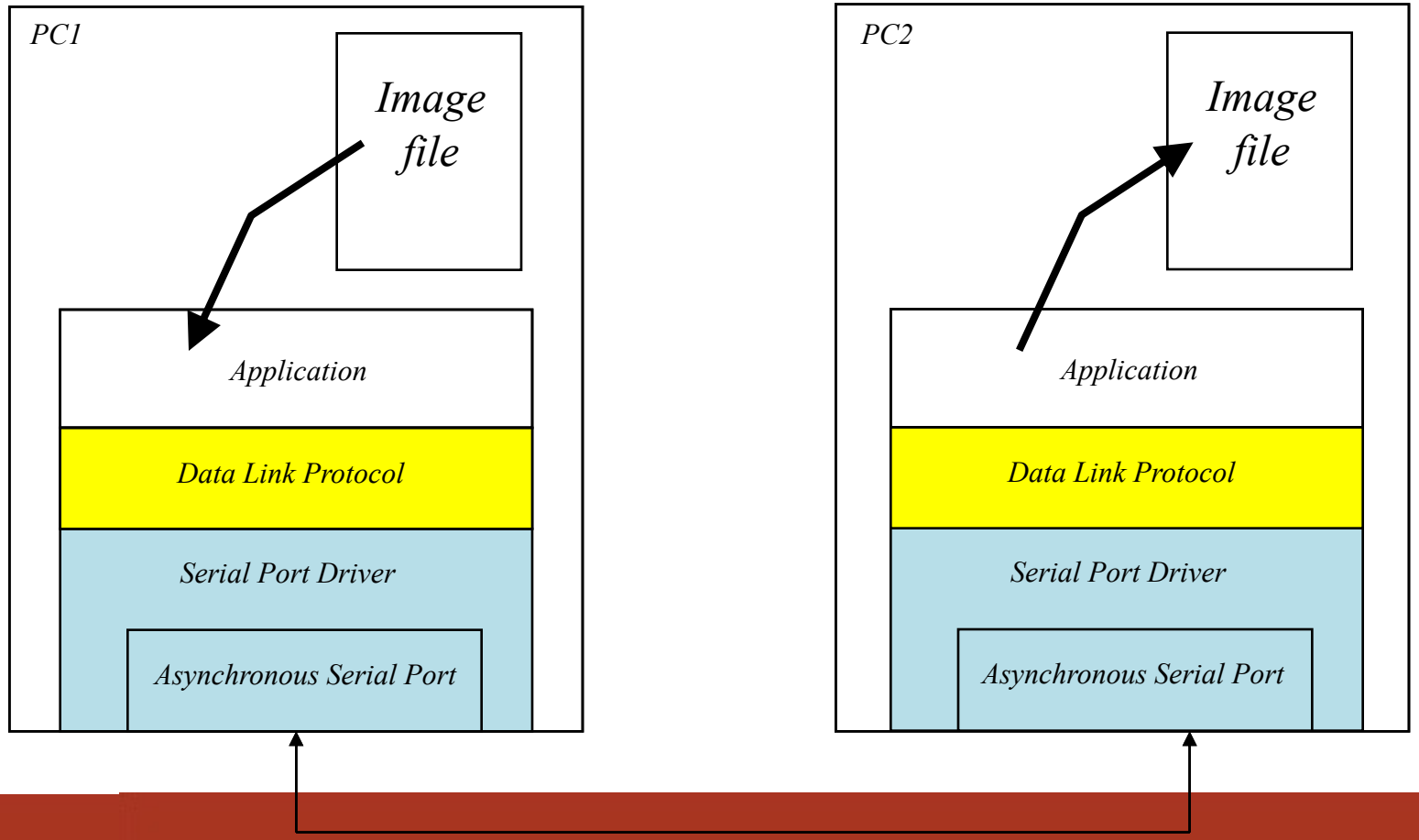
Code Example – Non-canonical Reception

```
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);
}
```

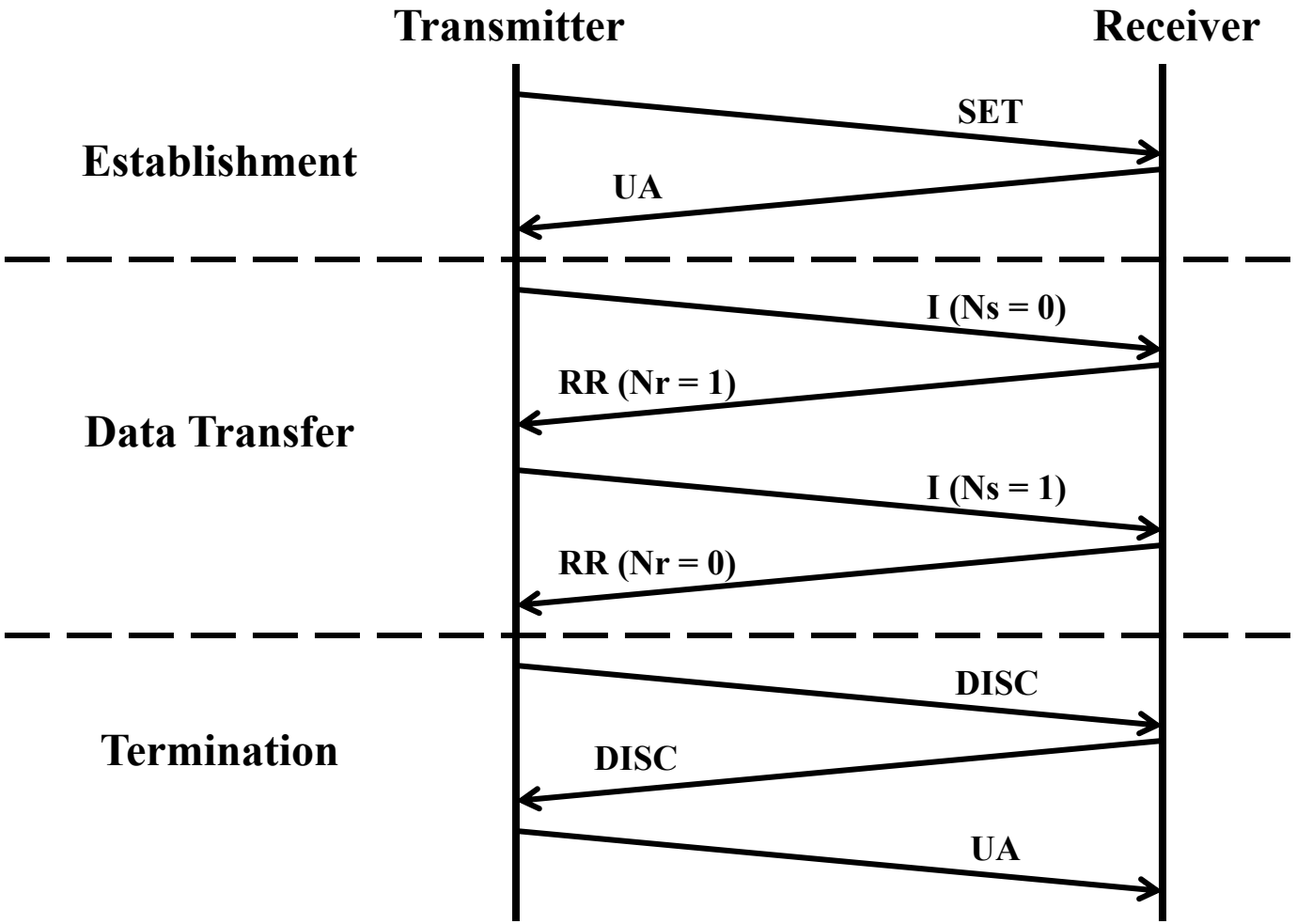
Layer Independence

- Layered architectures are based on layer independence
- This has following implications in this assignment
  - In the data link layer, no processing is done on the headers of the packets to be carried – this information is considered inaccessible to the data link protocol
    - At the the data link level there is no distinction between application control and data packets, nor is the numbering taken into account
  - The application does not know details of the data link protocol, only how to access its service
    - The application protocol does not know the frame structure or the delimitation mechanism, the existence of stuffing, the frame protection mechanism, eventual retransmissions, etc
    - All these functions are implemented exclusively at the data link layer

Data Link Protocol



Data Link Protocol Phases



Data Link Protocol – Specification

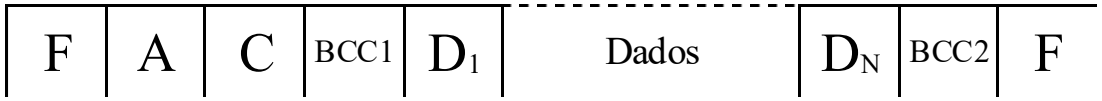
- Transmission organised into 3 types of frames – Information (I), Supervision (S) e Unnumbered (U)
  - Frames have a header with common format
  - Only I frames have a field for data communication that carries application data without interpreting it – **layer independence / transparency**
- Frame delimitation is obtained by a special 8 bit sequence (FLAG)
  - Transparency must be guaranteed through bit stuffing
- Transparent data transmission, i.e. independent of the code used for transmission
- Frames are protected by an error correction code
  - In S and U frames, there is simple frame protection, since they do not carry data
  - In I frames, there is double independent protection for header and data fields, enabling the use of the header even if there are errors in the data field
- Stop and Wait error control, unit window and modulo 2 numbering

Frame Specification & Header Delimitation

- All frames are delimited by flag: **01011010 (0x5c)**
- A frame may be initialised with one or more flags
  - This must be taken into consideration by the reception mechanism
- I, SET e DISC frames are designated commands and UA, RR and REJ frames replies
- All frames have a header with a common format
  - A (Address field)
    - **00000001 (0x01)** in Commands sent by the Transmitter and Answers sent by the Receiver
    - **00000011 (0x03)** in Commands sent by the Receiver and Answers sent by the Transmitter
  - C (Control fiels) – Defines the type of frame and carries the sequence numbers N(s) in I frames and N(r) in S frames (RR, REJ)
  - BCC (Block Check Character) – Provides error control based on an octet that guarantees that there is an even pair of 1's (even parity) for each bit position, considering all octets protected by the BCC (header or data) and the BCC (before stuffing)

Frame Formats

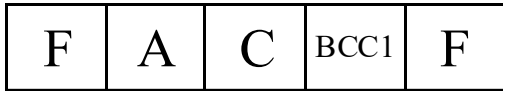
- Information Frames



(before *stuffing* and after *destuffing*)

F	**Flag**		
A	**Address field**		
C	**Control field**	**1 S 0 0 0 0 0 0**	**S = N(s)**
D_1 ... D_N	**Data field (contains data generated by the application)**		
$BCC_{1, 2}$	**Independent protection fields (1 – header, 2 – data)**		

- Supervision and Unnumbered Frames



F	**Flag**		
A	**Address field**		
C	**Control field**		
	SET (set up)	**0 0 0 0 0 1 1 1**	
	DISC (disconnect)	**0 0 0 0 1 0 1 0**	
	UA (unnumbered acknowledgment)	**0 0 0 0 0 1 1 0**	
	RR (receiver ready / positive ACK)	**0 0 0 R 0 0 0 1**	
	REJ (reject / negative ACK)	**0 0 0 R 0 1 0 1**	**R = N(r)**
BCC_1	**Protection field (header)**		

Transparency – Why?

- This work uses asynchronous communication
  - This technique is characterised by the transmission of characters (short sequence of bits, whose number can be configured) delimited by a Start and a Stop character
- Some protocols use characters (words) of a code (e.g. ASCII) to delimit and identify the frame fields and support protocol mechanisms
- For transparent communication, i.e. communication independent of the code used for transmission, it is necessary to use escape mechanisms
  - To identify the occurence of delimiting characters in the data and replace them so that they can be correctly interpreted at the receiver

Transparency – Why?

- The protocol to be implemented here is not based on any code, so the transmited/ received characters should be interpreted as plain octets (bytes), where any of the 256 possible combinations can occur
- To avoid that a character inside a frame is wrongfuly recognised as a delimiting flag, you need to use a mechanism that provides transparency
  - You shall use a mechanism called bit stuffing, used in common link protocols like PPP or HDLC
  - Your protocol shall use the escape byte **01011101** (**0x5d**)

Transparency – Byte Stuffing

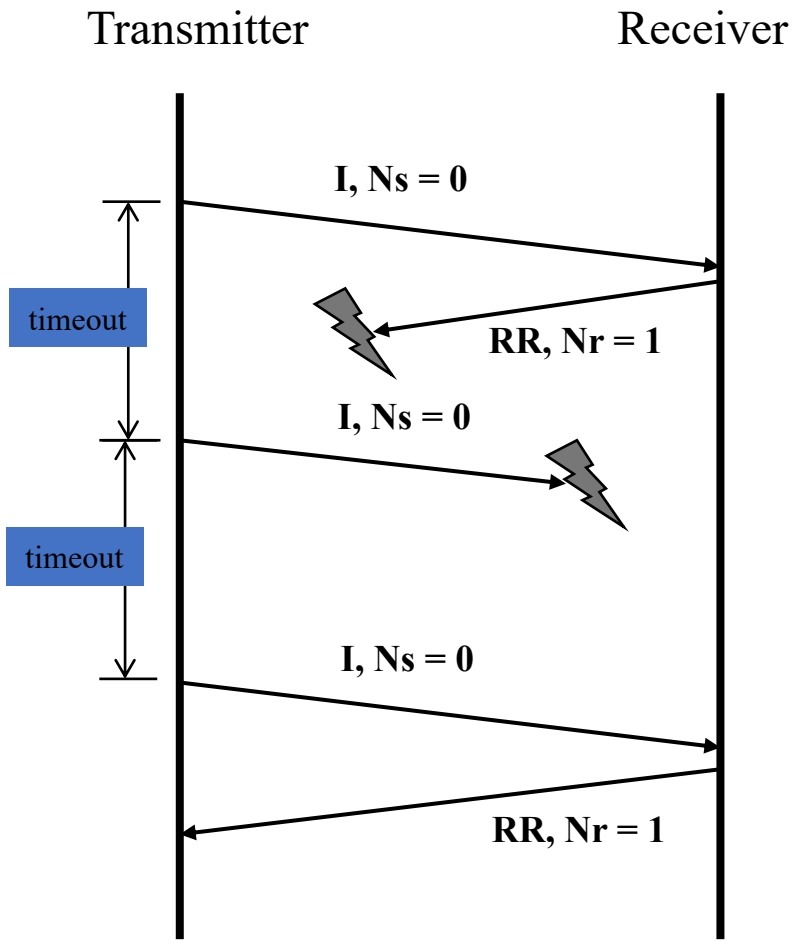
- If the octet 01011010 (0x5c) occurs inside a frame, i.e. the pattern corresponding to a flag, the octet is replaced by the sequence 0x5d 0x7c (escape octet followed by the result of the exclusive or of the replaced octet with octet 0x20)
- If the octet 01111101 (0x5d) occurs inside a frame, i.e. the escape octet pattern, the octet is replaced by the sequence 0x5d 0x7d (escape octet followed by the result of the exclusive or of the replaced octet with octet 0x20)
- The generation of BCC considers only the original octets (before stuffing), even if any octet must be replaced by the escape sequence
- The BCC verification is performed on the original octets, i.e. after the inverse operation (destuffing) in case there had been any substitution of the special octets by the escape sequence

Frame Reception – Error Control

- I, S or U frames with a wrong header are ignored without action
- Header is protected by BCC1 field
- The data field of an I frame is protected by an own BCC (BCC2)
  - Even parity on each bit of the data and BCC2
- I frames received without errors on the header and data field are accepted
  - If it is a new frame, the data field is passed to the application and the frame is confirmed with RR
  - If it is a duplicate, the data field is discarded, but the frame is confirmed with RR anyway
- I frames without detected errors on the header but with errors detected on the data field: data field is discarded, but control field can be used to trigger an action
  - If it is a new frame, a retransmission request can be issued with a REJ request, triggering a faster retransmission that waiting for a timeout
  - If it is a duplicate, the frame should be confirmed with RR
- I, SET e DISC frames are protected by a timer
  - If a time-out occurs, transmission should be repeated a maximum number of times, e.g. 3, which should be configurable

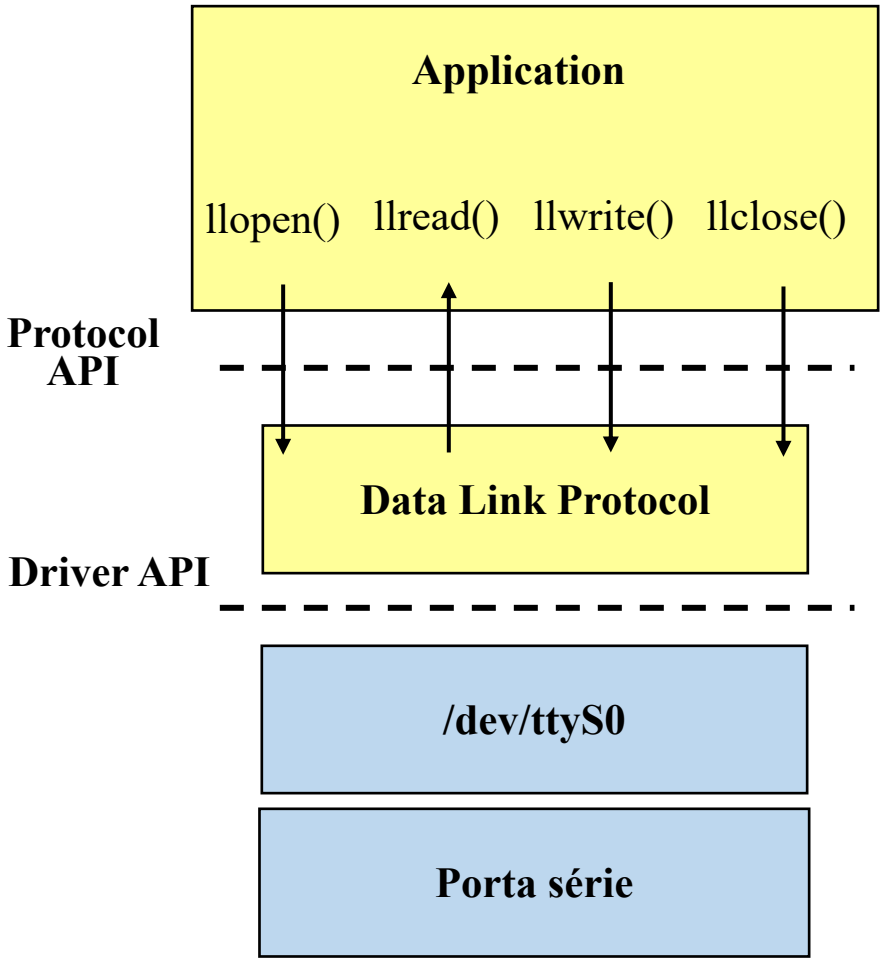
Data Transfer – Retransmissions

- Acknowledgement/ Error Control
  - *Stop-and-Wait*
- Timer
  - Set after an I, SET or DISC frame
  - De-activated after a valid acknowledgement
  - If exceeded (*time-out*), forces retransmission
- I frame retransmissions
  - After *time-out*, due to loss of frame or acknowledgement
    - Configurable maximum number of retrials
  - After negative acknowledgement (REJ)
- Frame protection
  - Generation and verification of the protection fields (BCC)

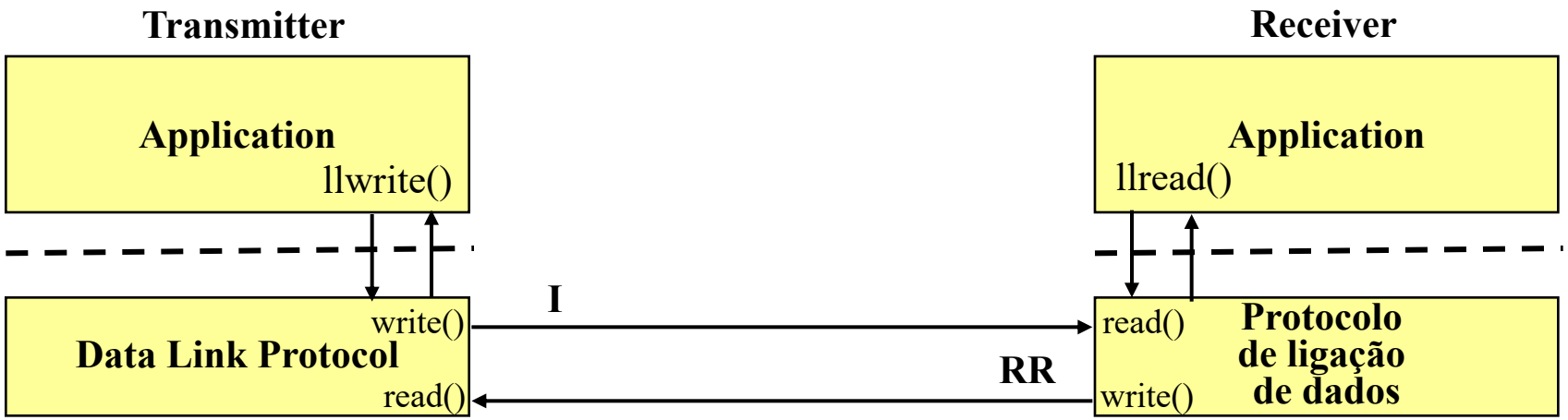


Protocol Interface

- Implement the protocol API in an .h file to be included by the applications
- The application should only use the functions in that file

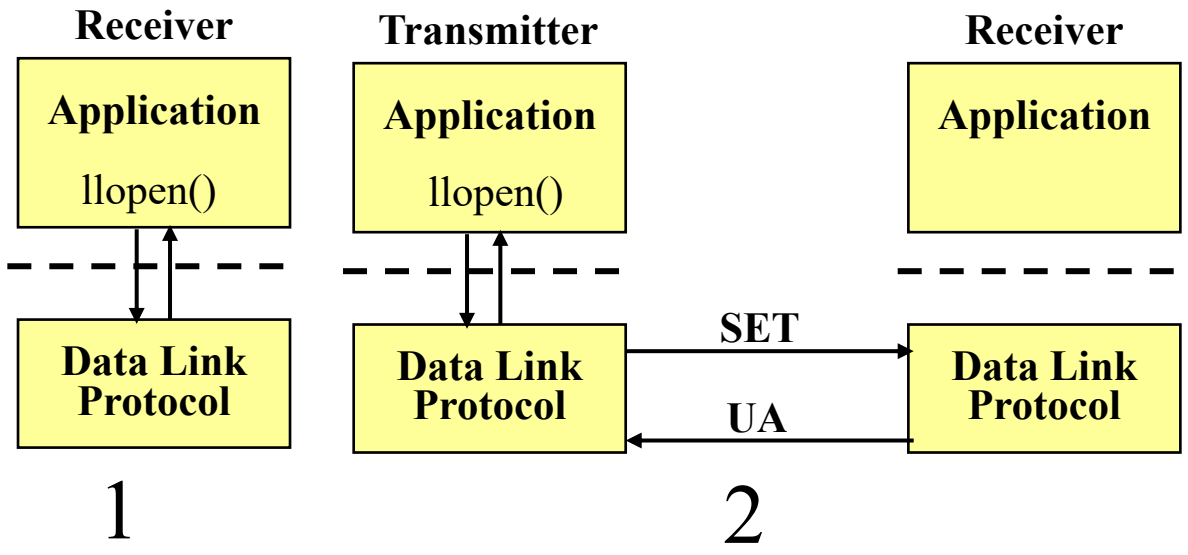


Protocol API: read / write



Protocol API – open

- int llopen(linkLayer connectionParameters)
  - arguement
  - return value
    - Data connection id
    - Negative value in case of failure/ error




```
typedef struct linkLayer{
   char serialPort[50];
   int role; //defines the role of the program: 0==Transmitter,
1=Receiver
   int baudRate;
   int numTries;
   int timeOut;
} linkLayer;
```

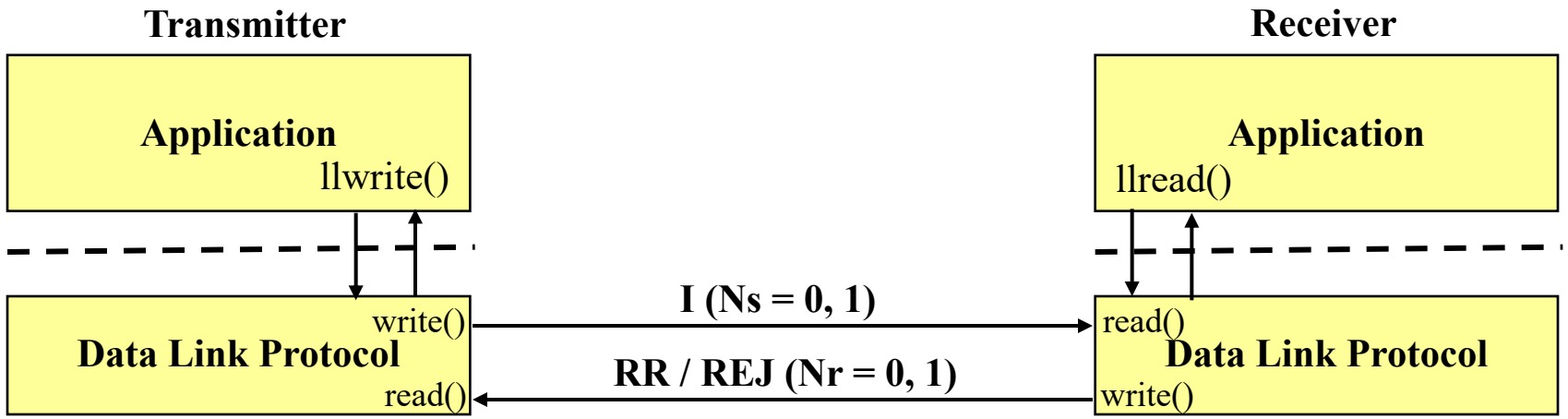
```
//ROLE
#define NOT_DEFINED -1
#define TRANSMITTER 0
#define RECEIVER 1
```

```
//SIZE of maximum acceptable payload; maximum number of bytes
that application layer should send to link layer
#define MAX_PAYLOAD_SIZE 1000

//CONNECTION deafault values
#define BAUDRATE_DEFAULT B38400
#define MAX_RETRANSMISSIONS_DEFAULT 3
#define TIMEOUT_DEFAULT 4
#define _POSIX_SOURCE 1 /* POSIX compliant source */
```

Protocol API – read / write

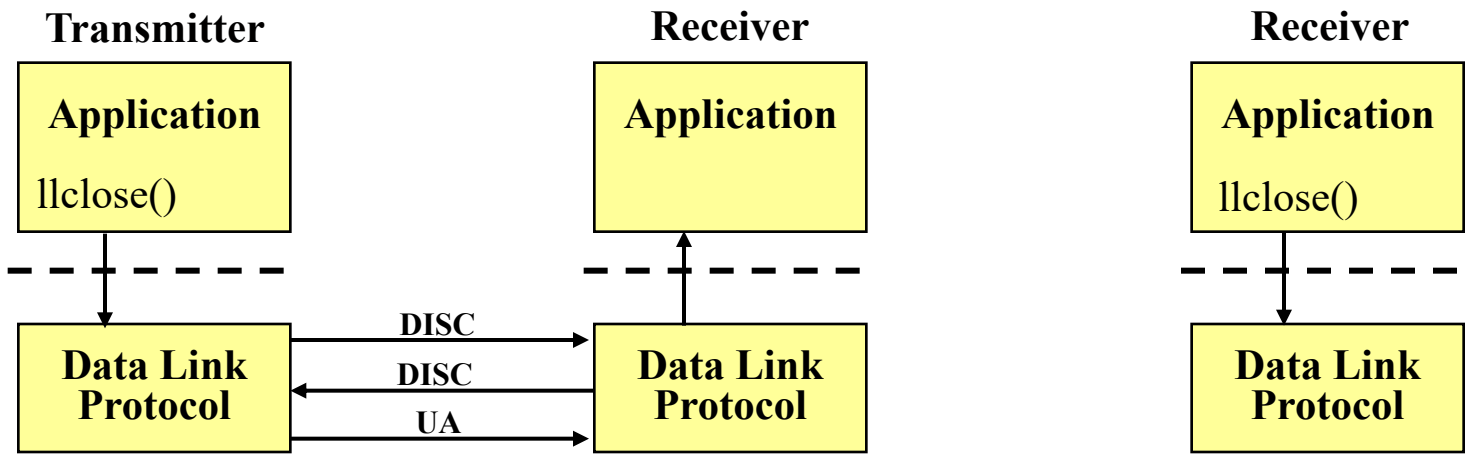
- int llwrite(int fd, char * buffer, int length)
  - Arguments
    - buffer: array of characters to transmit
    - length: length of the character array
  - Return value
    - Number of written characters
    - Negative value in case of failure/ error

- int llread(char * buffer)
  - Arguments
    - fd: data link identifier
    - buffer: received character array
  - Return value
    - Array length (nurmber of characters read)
    - Negative value in case of failure/ error

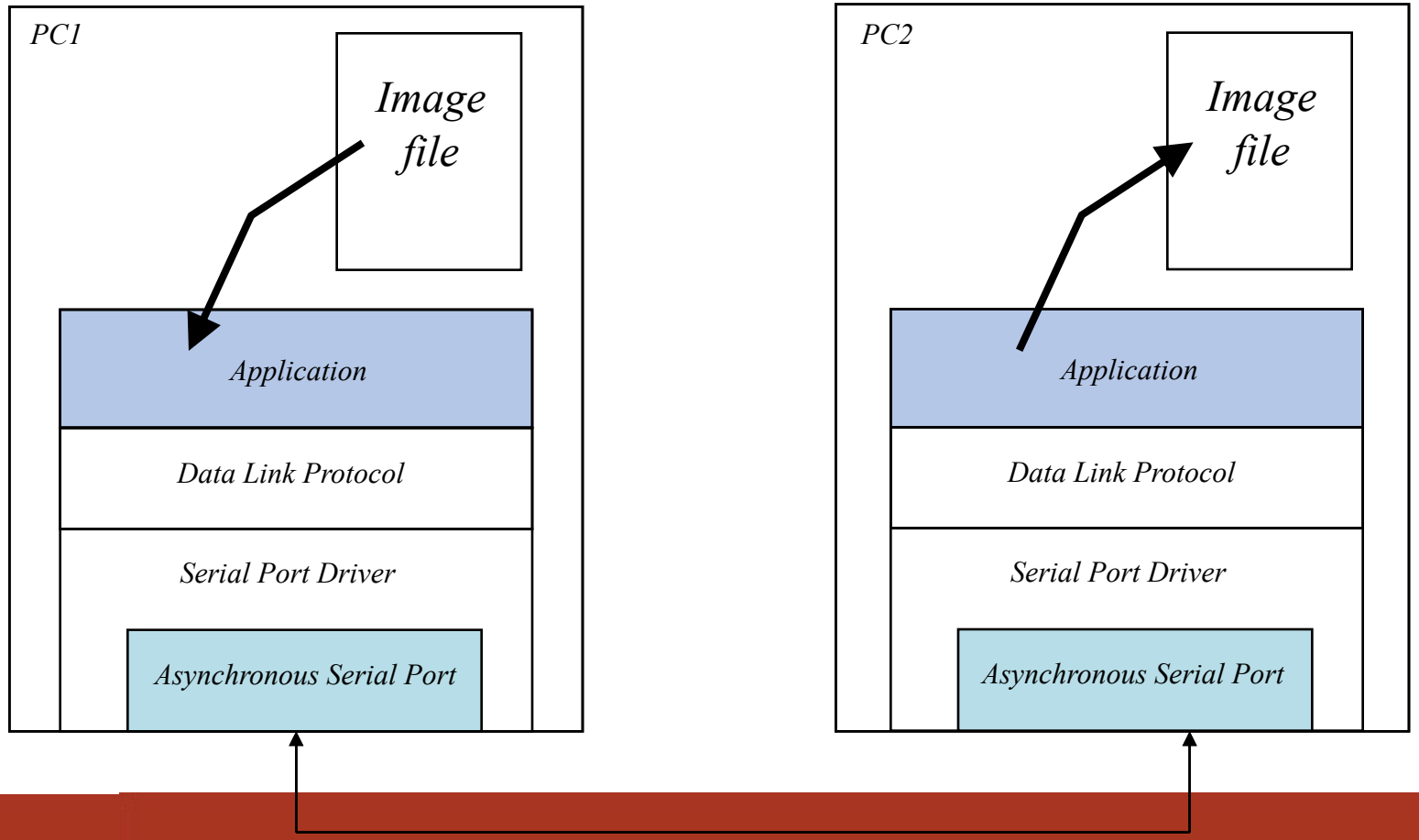


Protocol API – close

- int llclose(linkLayer connectionParameters, int showStatistics);
  - arguments
    - fd: data link identifier
    - showStatistics: true or false to show statistics collected by link layer for performance evaluation
  - Return value
    - Positive value in case of failure/ error
    - Negative value in case of failure/ error



Application



Application

- Simple file transfer application
  - Application uses its own protocol
- Application includes the .h file that specifies the protocol API functions and structures
- A version of the application is available in moodle
- You will need to compile it with your protocol

Performance Evaluation

- Random error generation in data frames for performance evaluation
  - Suggestion: for each correctly received I frame, simulate at the receiver the occurence of errors in header and data field according to pre-defined and independent probabilities, and proceed as if they were real errors
- Protocol event logging
  - Errors, Number of retransmitted/ received I frames, number of time-outs, number of sent/ received REJ, ...

Lab 3 – Link Layer Protocol Workplan

- Students have access to full assignment from the start
  1. Recall C programming; Exchange strings over serial connections; tools; github
  2. Sending and receiving control frame; state machine
  3. Implement stop & wait protocol in llwrite and llread
     1. Message reception
     2. Sequence number logic
     3. Byte stuffing
     4. REJ (negative ACK)
     5. Timer and retransmission
  4. Structure code llopen, llclose, llwrite, llread; separate header and implementation files; integrate with application code
  5. Performance evaluation

Lab 3 – Link Layer Protocol Workplan

01-Apr	02-Apr	03-Apr	04-Apr	Lab 3: Introdution to the serial port; communicating over serial port, tools
08-Apr	09-Apr	10-Apr	11-Apr	Lab 3: Framing byte arrays, state machine
15-Apr	16-Apr	17-Apr	18-Apr	Lab 3: Stop & Wait protocol; data frame transmission & reception state machines
22-Apr	23-Apr	24-Apr	25-Apr	Lab 3: Stop & Wait protocol; data frame transmission & reception state machines
29-Apr	30-Apr	01-May	02-May	Lab 3: Reception state machine; Alarm and re-transmission
06-May	07-May	08-May	09-May	
13-May	14-May	15-May	16-May	Lab 3: llopen, llclose, llwrite, llread, code structure and interface to higher layer
20-May	21-May	22-May	23-May	Lab 3: Debugging and performance evaluation

Lab 3 – Link layer protocol

- Evaluation
  - In class by instructor: preparation , development, understanding, evolution
  - As milestones are achieved
    1. Exchange strings over serial connections, tools, github: 10% (2/20)
       - Establishment of a github code repository
         - i. in week 2: 1/20; ii. In week 3 0.5/20; iii. Later 0/3
    2. Sending and receiving control frame; state machine: 20% (4/20)
    3. Implement stop & wait protocol in llwrite and llread: 20% (4/20)
    4. Timer and retransmission: 10% (2/20)
    5. Structure code llopen, llclose, llwrite, llread; separate header and implementation files: 10% (2/20)
  - Individual in-class quizz (15 min): 20% (4/20)
  - Performance evaluation (2 page additional submission): 10% (2/20)

Laboratory

1. Experiments on web and tools
   - 2 weeks
   - 12.45%
2. Configuration of a LAN
   - 5 weeks
   - 42.5%
3. Link layer protocol
   - 6 weeks
   - 45%

L = 12.5% L1 + 42.5% L2 + 45% L3

Assessment

- Lab 1:
  - Moodle lesson (ca be answered at home)
  - Sequence diagram
  - Coarse graded 0 (no completion), 50% (overall NOK), 100% (overall OK)
- Lab 2:
  - Weekly assessment during class by teacher based on questions asked, effort, preparation
  - Demonstration of achievements
  - Final in-class quizz
- Lab 3
  - Weekly assessment during class by teacher based on questions asked, effort, preparation
  - Demonstration of achievements
  - Final in-class quizz

Effort

Contact hours T	26
Contact hours Lab	26: classes (13 x 2)
Lab preparation	28 (~10 for dowload application)
Study	28
Exam preparation	50
	162

- You are supposed to
  - Prepare TP classes (~2h)
    - Then we can discuss
  - Prepare lab classes (~2h)
    - Recall: weekly in-class evaluation!

Very Important

- Ask questions whenever you do not understand something

- This is one of the most important factors in academic success