Data Link Protocol

(1st Lab Work)

FEUP

Computer Networks

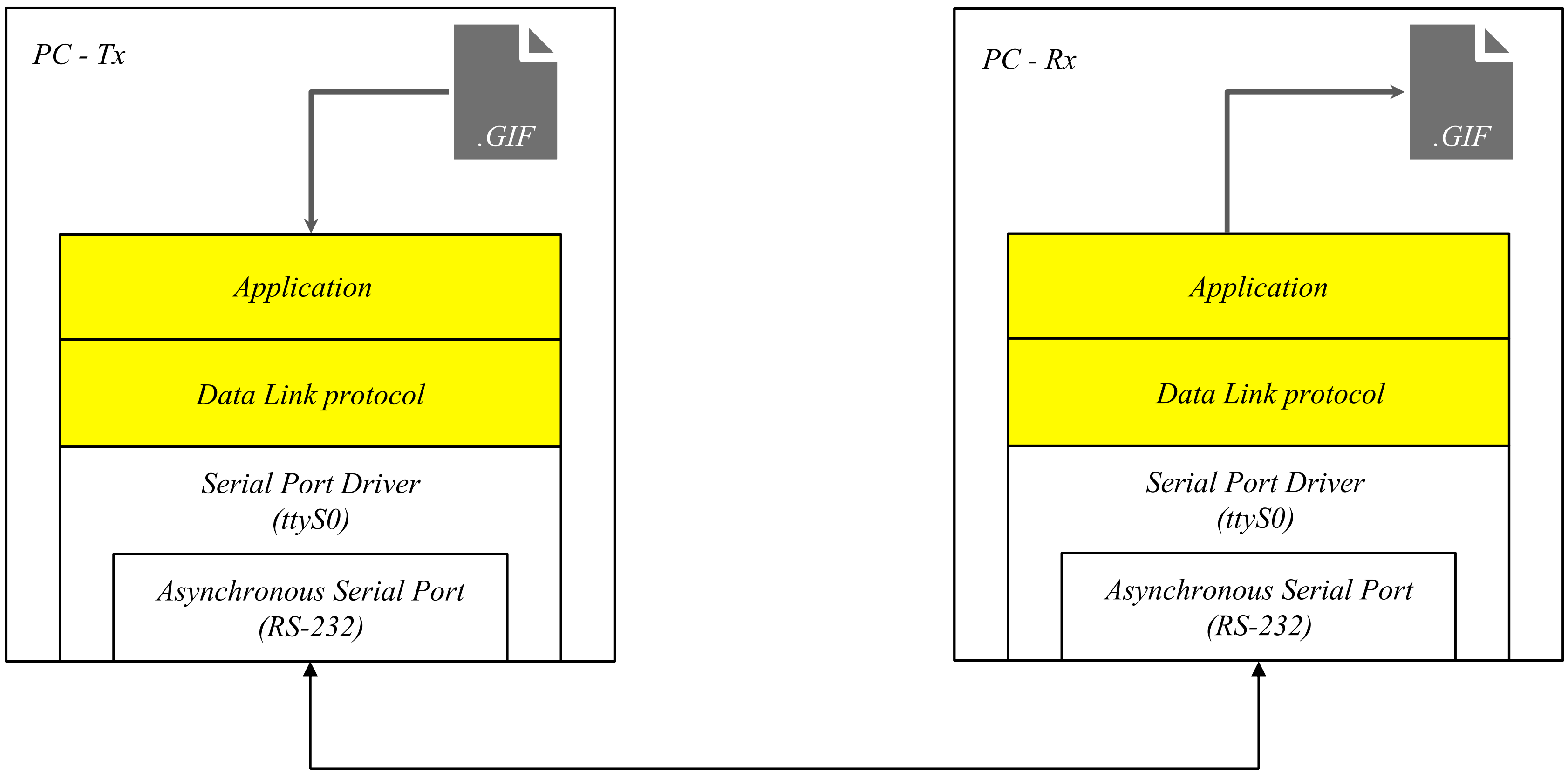
Work overview

- Goals
  - » Implement a data link layer protocol, according to the specification provided in this document
    - » this protocol implements transmitter and receiver functionality to transfer a file stored on a computer hard disk between computers connected through a RS-232 serial cable
  - » Develop a simple transmitter and receiver data transfer application to test the protocol, according to the specification provided in this document
    - » the application to be developed uses/invokes the functions implemented by the data link layer protocol
      - » the data link layer protocol thus offers/exposes an API to the upper layer
- Development environment
  - » PC running LINUX
  - » Programming language – C
  - » Serial port RS-232 (asynchronous communication)

Evaluation

- Organization
  - » Groups of 2 elements
  - » Each group develops the transmitter and receiver
- Evaluation criteria
  - » Participation during class (continuous evaluation)
  - » Presentation and demonstration of the work on 5 milestones
    - – M1: Exchange strings over serial connections;
    - – M2: Sending and receiving control frame (SET/UA) and state machine in llopen;
    - – M3: Implement the Stop & Wait protocol in llwrite and llread;
    - – M4: Timer and retransmission;
    - – M5: Application layer implementation and correct API operation of llopen, llclose, llwrite, llread;
  - » Individual 15-minute quiz to be answered in the classroom, on the last class before the presentation
  - » Final report

Test configuration



Data Link Protocol - Goal and General Functionality

- Goal of the Data Link Layer Protocol
  - » Provide reliable communication between two systems connected by a communication medium – in this case, a serial cable
- Generic functions of data link protocols
  - » Framing
    - » Packaging and synchronisation/delimitation
  - » Connection establishment and termination
  - » Frame numbering
  - » Acknowledgement
  - » Error control (e.g.: Stop-and-Wait, Go-back-N, Selective Repeat)
  - » Flow control

Data Link Protocol - Functionality - Framing

- Packaging - data coming from the upper layer is packed into frames
    - frames have an header, a body and a trailer
    - user/application data goes into the body section
    - these frames are designated as information frames
- Frame synchronisation (delimitation)
  - start and end of frames are uniquely identified so that data reception can become synchronised
    - main alternative is to use a special character/flag at the beginning and end of frames
    - need to make sure that its value does not occur elsewhere inside the frame
      - transparency or stuffing mechanism (explained in slides 16 and 17)
  - the size of frames may be implicitly determined
    - by counting the number of bytes in between synchronisation flags
  - or explicitly indicated
    - in one field of the header

Data Link Protocol - Functionality

- Connection establishment and termination
  - exchange of specific messages sent in fixed-length frames
    - designated of <u>supervision frames</u> having only control fields (no user data)
- Frame numbering
  - a counter module-n in the header of frames to allow to verify the correct sequence of information frames and/or the occurrence of duplicates
- Positive Acknowledgement
  - every time a frame is received without errors and in the right sequence a positive acknowledgement is sent back to the sender
- Error control (e.g.: Stop-and-Wait, Go-back-N, Selective Repeat)
  - use of timers (time-out) to enable re-transmission of un-acknowledged frames
  - use of negative acknowledgement to request the retransmission of out-of-sequence or errored frames
  - verification of duplicates which may occur due to re-transmissions

Data Link Protocol - Specification

- The protocol to implement combines characteristics of existing real-world data link protocols
  - » agnostic to the type of user data to be transferred (independence and transparency)
  - » transmission organised into frames, which can be of three types
    - » Information (I), Supervision (S) and Unnumbered (U)
  - » Frames have a header with a common format
    - » only Information frames have a field for user data transport
      - » a field to transport a packet generated by the application, which content is not processed by the data link protocol
  - » Frame delimiting is done by means of a special eight-bit sequence (flag) and a byte stuffing technique ensures that this value will not occur inside the frame (explained in slide 17)
  - » The frames are protected by an error detection code
    - – In frames S and U there is simple frame protection (since they do not carry data)
    - – In I frames there is double and independent protection of the header and the data field (which allows to use a valid header, even if an error occurs in the data field)
  - » The Stop and Wait variant is used (unit window and modulo 2 numbering)

Format and types of frames

» Supervision (S) and Unnumbered (U) Frames

F	A	C	BCC1	F

F	**Flag**
A	**Address Field**
C	**Control Field to indicate the type of supervision frame/message**
	SET (set up)
	DISC (disconnect)
	UA (unnumbered acknowledgment)
	RR (receiver ready / positive ACK)
	REJ (reject / negative ACK)
BCC_1	**Protection Field to detect the occurrence of errors in header**

Format and types of frames

» Supervision (S) and Unnumbered (U) Frames

F	A	C	BCC1	F

Field	Value	Meaning
F	01111110 / 0x7E	Synchronisation: start or end of frame
A	00000011 / 0x03	Address field in frames that are commands sent by the Transmitter or replies sent by the Receiver
	00000001 / 0x01	Address field in frames that are commands sent by the Receiver or replies sent by the Transmitter
C	00000011 / 0x03	SET frame: sent by the transmitter to initiate a connection
	00000111 / 0x07	UA frame: confirmation to the reception of a valid supervision frame
	10101010 / 0xAA	RR0 frame: indication sent by the Receiver that it is ready to receive an information frame number 0
	10101011 / 0xAB	RR1 frame: indication sent by the Receiver that it is ready to receive an information frame number 1
	01010100 / 0x54	REJ0 frame: indication sent by the Receiver that it rejects an information frame number 0 (detected an error)
	01010101 / 0x55	REJ1 frame: indication sent by the Receiver that it rejects an information frame number 1 (detected an error)
	00001011 / 0x0B	DISC frame to indicate the termination of a connection
BCC1	A *XOR* C	Field to detect the occurrence of errors in the header

Format and types of frames

» Information Frames (I)

F	A	C	BCC1	D_1	Data	D_N	BCC2	F

(before stuffing and after destuffing)

F **Flag**
A **Address Field**
C **Control Field to allow numbering information frames**
D_1 … D_N **Information Field (packet generated by the Application)**
$BCC_{1, 2}$ **Independent Protection Fields (1 – header, 2 – data)**

0 S 0 0 0 0 0 0 **S = N(s)**

N(s) implements a module-2 counter, enabling to distinguish frame 0 and frame 1 successively throughout transmission

Field	Value	Meaning
F	01111110 / 0x7E	Synchronisation: start or end of frame
A	00000011 / 0x03	Address field in frames that are commands sent by the Transmitter or replies sent by the Receiver
	00000001 / 0x01	Address field in frames that are commands sent by the Receiver or replies sent by the Transmitter
C	00000000 / 0x00	Information frame number 0
	10000000 / 0x80	Information frame number 1
BCC1	A *XOR* C	Field to detect the occurrence of errors in the header
BCC2	D_1 *XOR* D_2 *XOR* D_3 … *XOR* D_N	Field to detect the occurrence of errors in the data field

Packets and frames

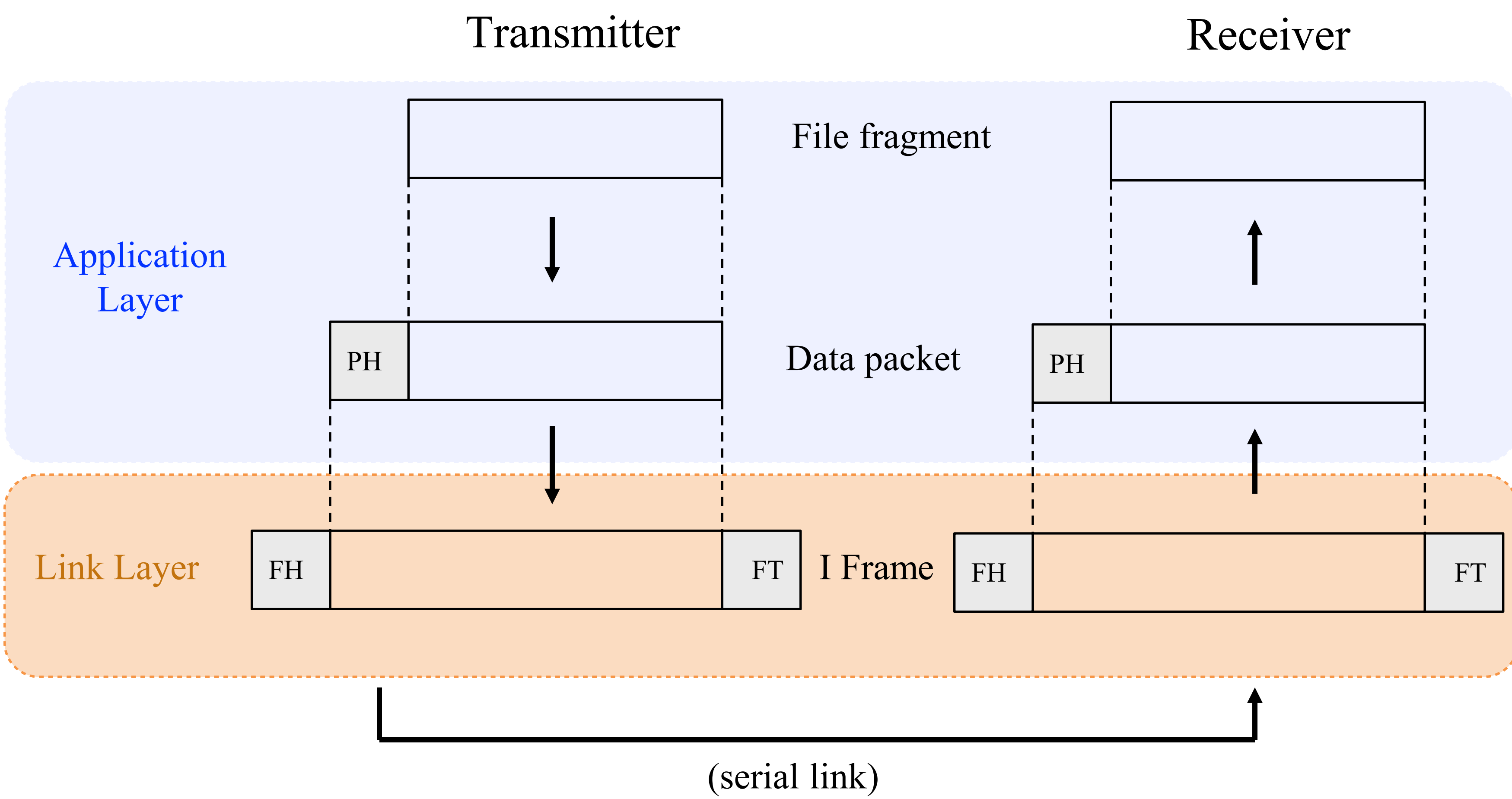
- At the application layer
  - the file to be transmitted is fragmented - the fragments are encapsulated in data packets which are passed to the link layer one by one
  - in addition to data packets (which contain file fragments), the Application protocol uses control packets
  - the format of the packets (data and control) is defined ahead (slide 27)

- At the link layer each packet (data or control) is carried in the data field of an I frame

- The Transmitter is the machine that sends the file and the Receiver is the machine that receives the file
  - thus, only the Transmitter transmits packets (data or control) and therefore only the Transmitter transmits I frames

- Both the Transmitter and the Receiver send and receive frames (write or read frames into/from the serial line)

Data packets and I frames



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

Control packets are also transported in I frames

Frames - Delimitation and header

» All frames are delimited by flags (**01111110**)

» A frame can be started with one or more flags, which must be taken into account by the frame reception mechanism

» Frames I, SET and DISC are designated Commands and the rest (UA, RR and REJ) are called Replies

» Frames have a header with a common format

– A (Address Field)

- **00000011** (**0x03**) in Commands sent by the Transmitter and Replies sent by the Receiver
- **00000001** (**0x01**) in Commands sent by the Receiver and Replies sent by the Transmitter

– C (Control Field) – defines frame type and carries sequence numbers N(s) in I frames and N(r) in Supervision frames (RR, REJ)

– BCC (Block Check Character) - error detection based on the generation of an octet (BCC) such that there is an even number of 1s in each position (bit), considering all octets protected by the BCC (header or data, as appropriate) and the BCC itself (before stuffing)

Reception of frames - Procedures

- I, S or U frames with wrong header are ignored without any action
- The data field of the I frames is protected by its own BCC (even parity on each bit of the data octets and the BCC)
- I frames received with no errors detected in the header and data field are accepted for processing
  - If it is a new frame, the data field is accepted (and passed to the Application), and the frame must be confirmed with RR
  - If it is a duplicate, the data field is discarded, but the frame must be confirmed with RR
- I frames with no header error detected but error detected (by the respective BCC) in the data field – the data field is discarded, but the control field can be used to trigger an appropriate action
  - If it is a new frame, it is convenient to make a retransmission request with REJ, which allows to anticipate the occurrence of time-out in the transmitter
  - If it is a duplicate, it must be confirmed with RR
- I, SET and DISC frames are protected by a timer
  - In the event of a time-out, a maximum number of retransmission attempts must be made (the value must be configurable; for example, three)

Transparency/stuffing - Necessity

- The transmission between the two computers is, in this work, based on a technique called asynchronous transmission
  - This technique is characterised by the transmission of "characters" (short string of bits, whose number can be configured) delimited by Start and Stop bits
  - Some protocols use characters (words) of a code (for example ASCII) to delimit and identify the fields that constitute the frames and to support the execution of the protocol mechanisms
    - In these protocols, the transmission of data transparently (regardless of the code used by the protocol) requires the use of escape mechanisms
- The protocol to be implemented is not based on the use of any code, so the transmitted characters (consisting of 8 bits) must be interpreted as simple octets (bytes), and any of the 256 possible combinations can occur
- To avoid the false recognition of a flag inside a frame, a mechanism that guarantees transparency is needed

Transparency – Byte stuffing mechanism

» In the protocol to be implemented, the mechanism used in PPP is adopted, which uses the escape octet **01111101 (0x7d)**

– If the octet **01111110 (0x7e)** occurs inside the frame, i.e., the pattern that corresponds to a flag, the octet is replaced by the sequence **0x7d 0x5e** (escape octet followed by the result of the exclusive or of the octet replaced with the octet 0x20)

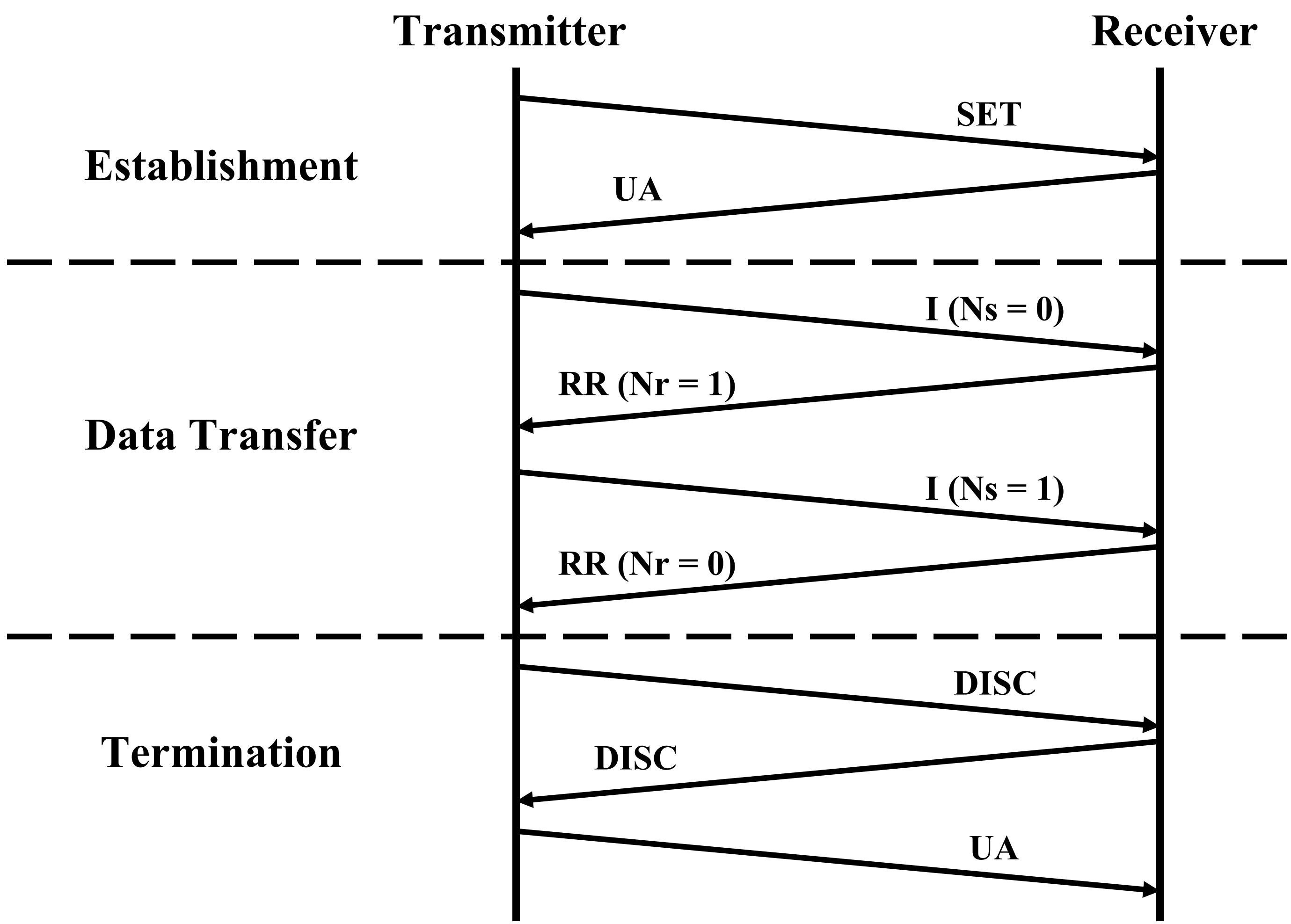
– If the octet 01111101 (0x7d) occurs inside the frame, i.e., the pattern that corresponds to the escape octet, the octet is replaced by the sequence **0x7d 0x5d** (escape octet followed by the result of the exclusive or of the octet replaced with the octet 0x20)

– In the BCC generation, only the original octets are considered (before the stuffing operation), even if some octet (including the BCC itself) has to be replaced by the corresponding escape sequence

– The verification of the BCC is carried out in relation to the original octets, i.e., after the inverse operation (destuffing) has been performed, if the replacement of any of the special octets by the corresponding escape sequence has occurred

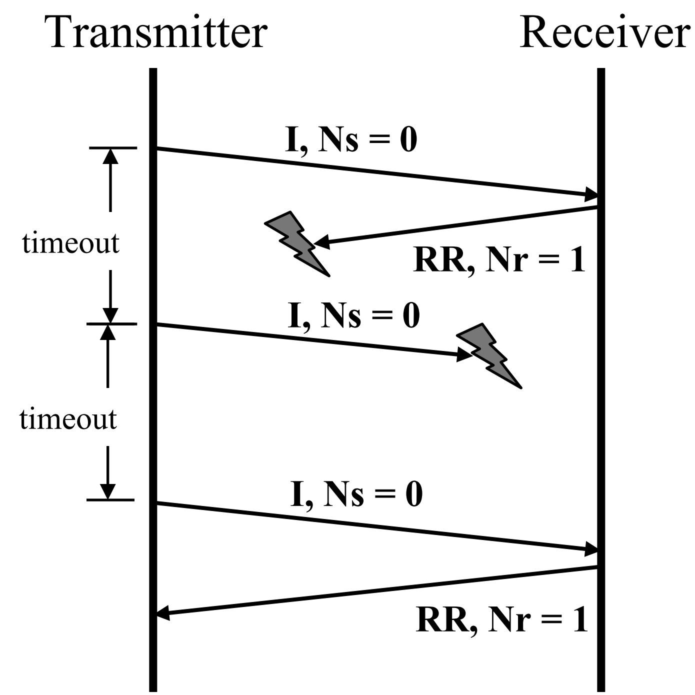
Data Link Protocol phases

» Example of a typical frame sequence (without errors)

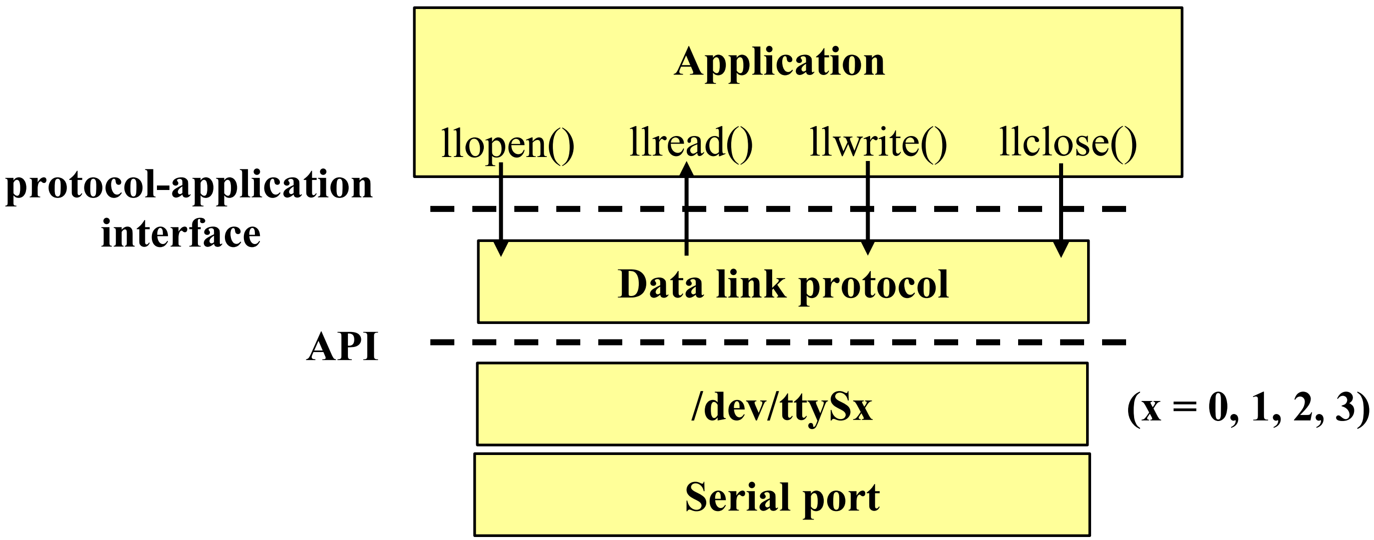


Data transfer – Retransmissions

- Acknowledgement / Error Control
  - » Stop-and-Wait
- Timer
  - » Enabled after an I, SET or DISC frame
  - » Disabled after a valid acknowledgement
  - » If acceded (timeout), forces retransmission
- I frames retransmission
  - » After a time-out, due to loss of the I frame or its acknowledgement
    - – Maximum number of predefined (configured) retransmission attempts
  - » After a receiving a negative acknowledgement (REJ)
- Frame protection
  - » Generation and verification of the protection fields (BCC)



Protocol-Application interface



Protocol-Application interface

- Examples of data structures

  » Protocol

```
struct LinkLayer {
char serialPort[50];                /*Device /dev/ttySx, x = 0, 1*/
LinkLayerRole role;                 /*TRANSMITTER | RECEIVER*/
int baudRate;                       /*Speed of the transmission*/
int nRetransmissions;               /*Number of retries in case of
                                    failure*/
int timeout;                        /*Timer value: 1 s*/
}
```

Protocol-Application interface – open

int llopen(LinkLayer connectionParameters)

arguments

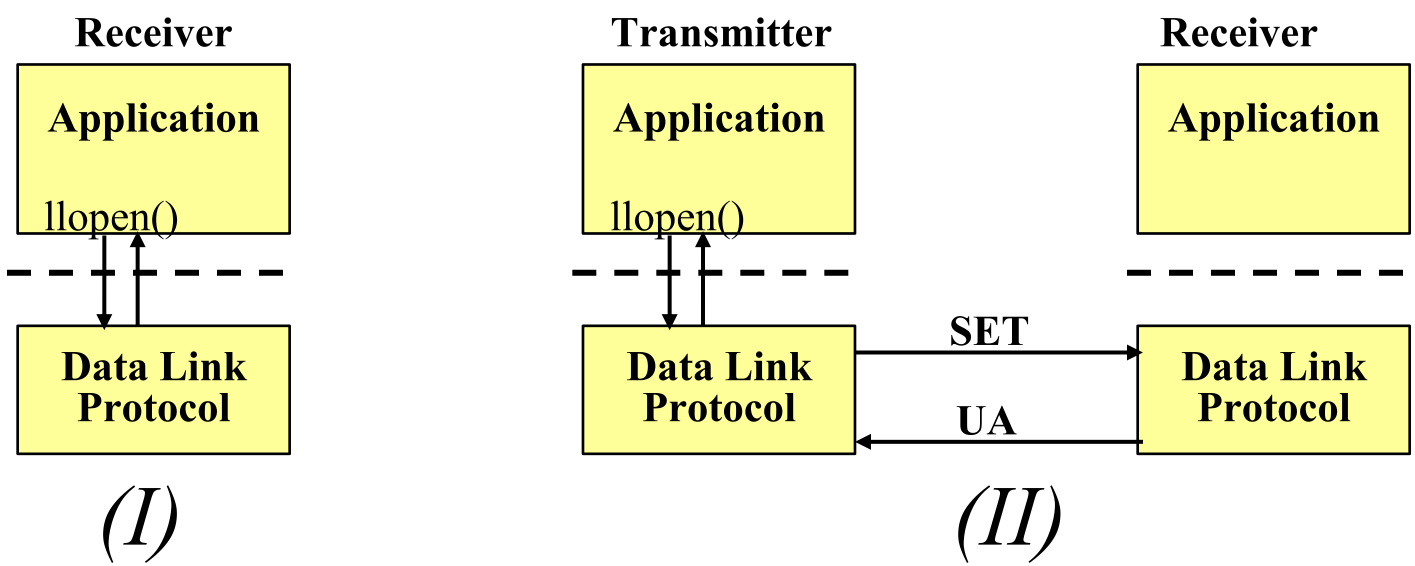
– connectionParameters: LinkLayer structure with the parameters of the connection

return

– data link identifier
– Negative value in case of error

Step I: the Rx application layer invokes llopen(…)

Step II: the Tx application layer invokes llopen(…) that runs at the link layer, exchanging supervision frames



Protocol-Application interface – send / receive

int llwrite(const unsigned char *buf, int bufSize)

arguments
- buf: array of characters to transmit
- bufSize: length of the characters array

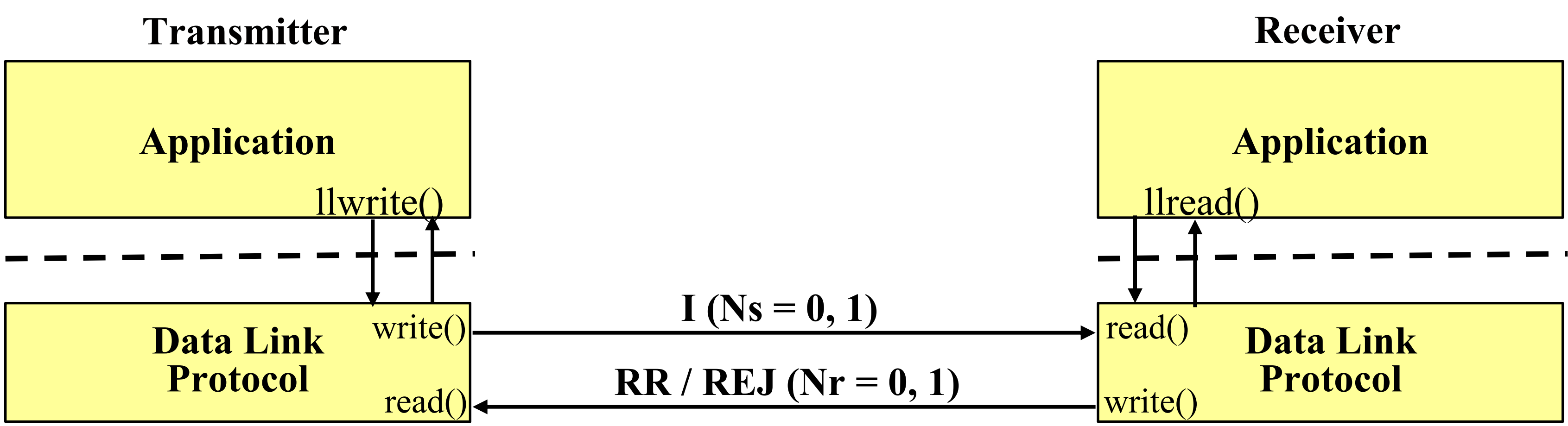
return
- number of writer characters
- negative value in case of error

int llread(unsigned char *packet)

arguments
- packet: array of characters read

return
- array length (number of characters read)
- negative value in case of error

The Tx application layer forms a packet (data or control) and invokes llwrite(…); the Rx application layer invokes llread(…); llwrite(…) and llread(…) exchange I and S frames. When frames are correctly received, both functions return the control to the application layer



Protocol-Application interface – close

int llclose(int showStatistics)

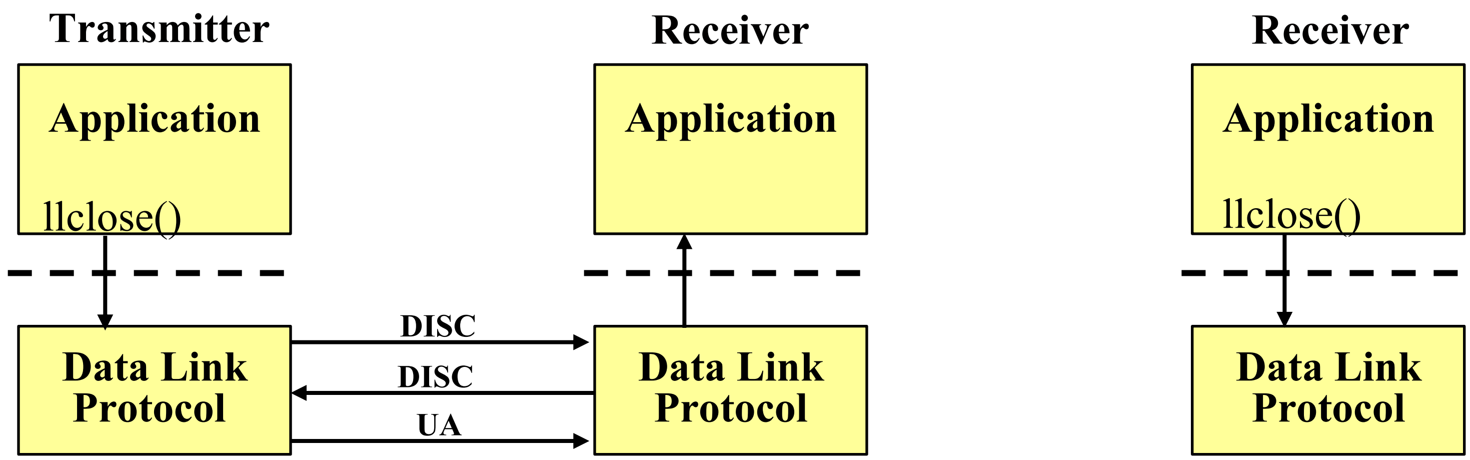
arguments

- ShowStatistics: print Communications statistics (e.g. number of frames, number of retransmissions, number of timeouts, etc.)

return

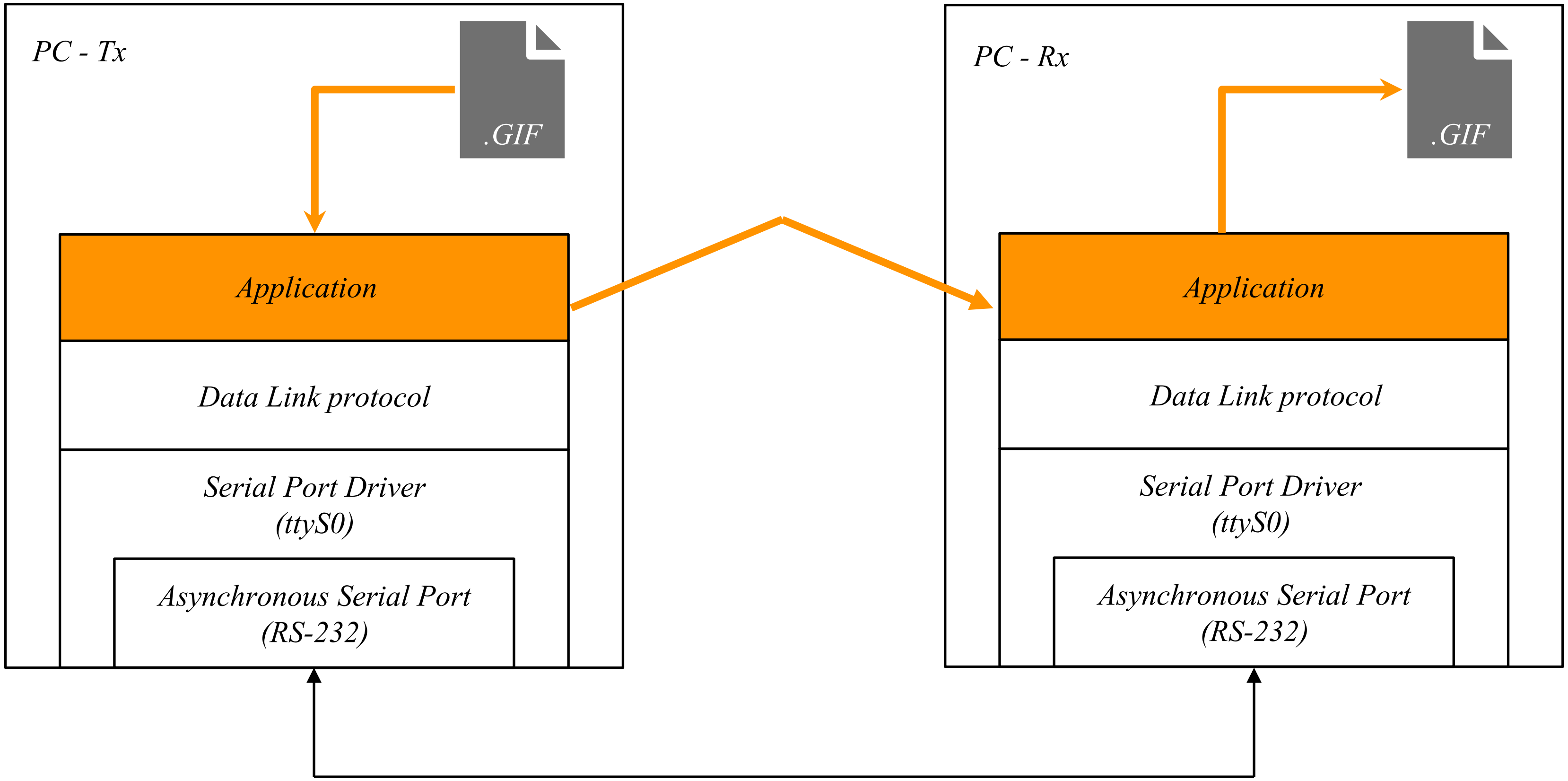
- positive value in case of success
- Negative value in case of error

The Tx and Rx application layer invoke llclose(…) that run at the link layer exchanging appropriate S frames



Test application

Simple application to access/store the file and pack/unpack data into packets



Test application - Specification

- The goal is to develop a very simple application protocol for transferring a file, using the reliable service offered by the data link protocol
- The application must support two types of packets sent by the Transmitter
  - » Control packages to signal the start and end of file transfer
  - » Data packets containing fragments of the file to be transmitted
- The control package that signals the beginning of the transmission (START) must have a field with the file size and optionally a field with the file name (and possibly other fields)
- The control packet that signals the end of transmission (END) shall repeat the information contained in the START packet
- Data packets must contain a field (two octets) that indicates the size of the respective data field ($D_1 \dots D_N$) to allow for additional checks regarding data integrity
  - » This size depends on the maximum size that may be established for the Information field of frames I

Application layer packets

- Data packet

C	S	L_2	L_1	P_1	…	P_k

  » C – control field (value: 2 – data)
  » S – sequence number (0-99)
  » L_2 L_1 – number of octets (K) in the data field
  » P_1 ... P_K – packet data field (K octets)

  $(K = 256 * L_2 + L_1)$

- Control packet

C	T_1	L_1	V_1	T_2	L_2	V_2	

  » C – control field (values: 1 – start; 3 – end)
  » Each parameter (size, file name or other) is coded as TLV (Type, Length, Value)
    – T (one octet) –indicates the parameter (0 – file size, 1 – file name, other values – to be defined, if necessary)
    – L (one octet) – indicates the V field size in octets (parameter value)
    – V (number of octets indicated in L) – parameter value

Layer Independency

- Layered architectures are based on the principle of independence between layers.
- This principle has the following consequences in the scope of this work:
  - » A the data link layer
    - » no processing shall be done that may affect the header of packets passed by the application layer (to be transported in Information frames)
      - » this information is considered inaccessible to the data link protocol
    - » no distinction is made between control and data packets (all packets coming from the application layer are equally considered as user data)
  - » At the application layer
    - » there is no knowledge about the details of the data link protocol, only how its services can be accessed
      - » no knowledge about the structure of the frames and the respective delineation/synchronisation mechanism, the existence of stuffing (and which option is adopted), the protection mechanism of the frames, the numbering of frames, any eventual retransmissions of I frames, etc.
        - » all these functions are exclusively performed in the data link layer

Evaluation

- Data link protocol
  - » Frame synchronisation process
  - » Retransmission process
  - » Robustness to errors
  - » Error control
- Application protocol
  - » Control packets
  - » File integrity
- Code organisation
  - » Interface between layers (functions)
  - » Layer independency
- Statistical characterisation of the protocol efficiency (see next slide)
- Demonstration and report
- Penalties
  - » Delays on the demo and / or report submission

Evaluation – protocol efficiency

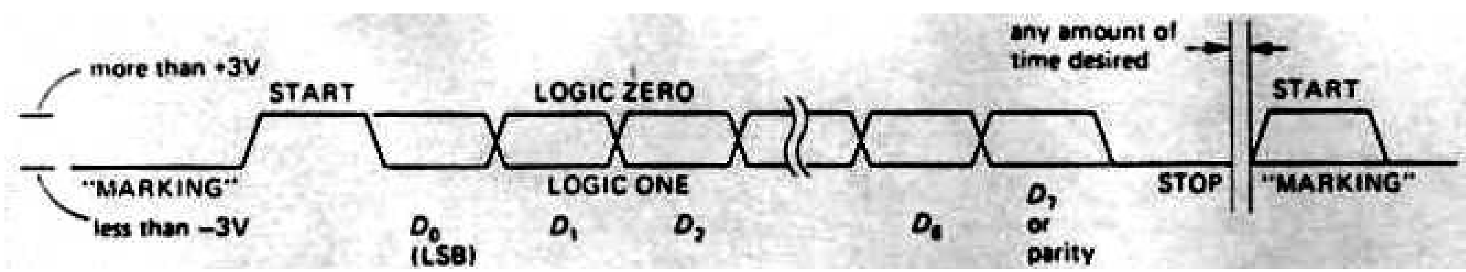
- Statistical characterization of efficiency S (FER, a). Suggestions:
  - » 1) vary FER, T_prop, C, size of I frame (C= link capacity, bit/s)
  - » 2) measure the obtained transference time S = R/C (R=received bitrate, bit/s)
  - » 3) plot S (FER,a) and check the validity of the known formulas for efficiency
  - » 4) repeat measurements
- To vary FER: random error generation on Information frames
  - » Suggestion – for each I frame correctly received, simulate (at the receiver) the occurrence of a header error and on the data field with pre-defined (and independent) probabilities, and proceed as a normal error
- To vary T_prop: generation of a simulated propagation delay
  - » Suggestion – use alarm.c to include a processing delay on each received frame

Note: you can use cable.c to perform the protocol efficiency evaluation.

Annexes

Asynchronous 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 (D7 bit used for data) – option adopted in this link layer protocol
» Transmission rate: 300 a 115200 bit/s



RS-232 Signals

- Physical layer protocol between a 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	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	

Connectors DB25 e DB9

Active signal

Control signal (> + 3 V)
Data signal (< - 3 V)

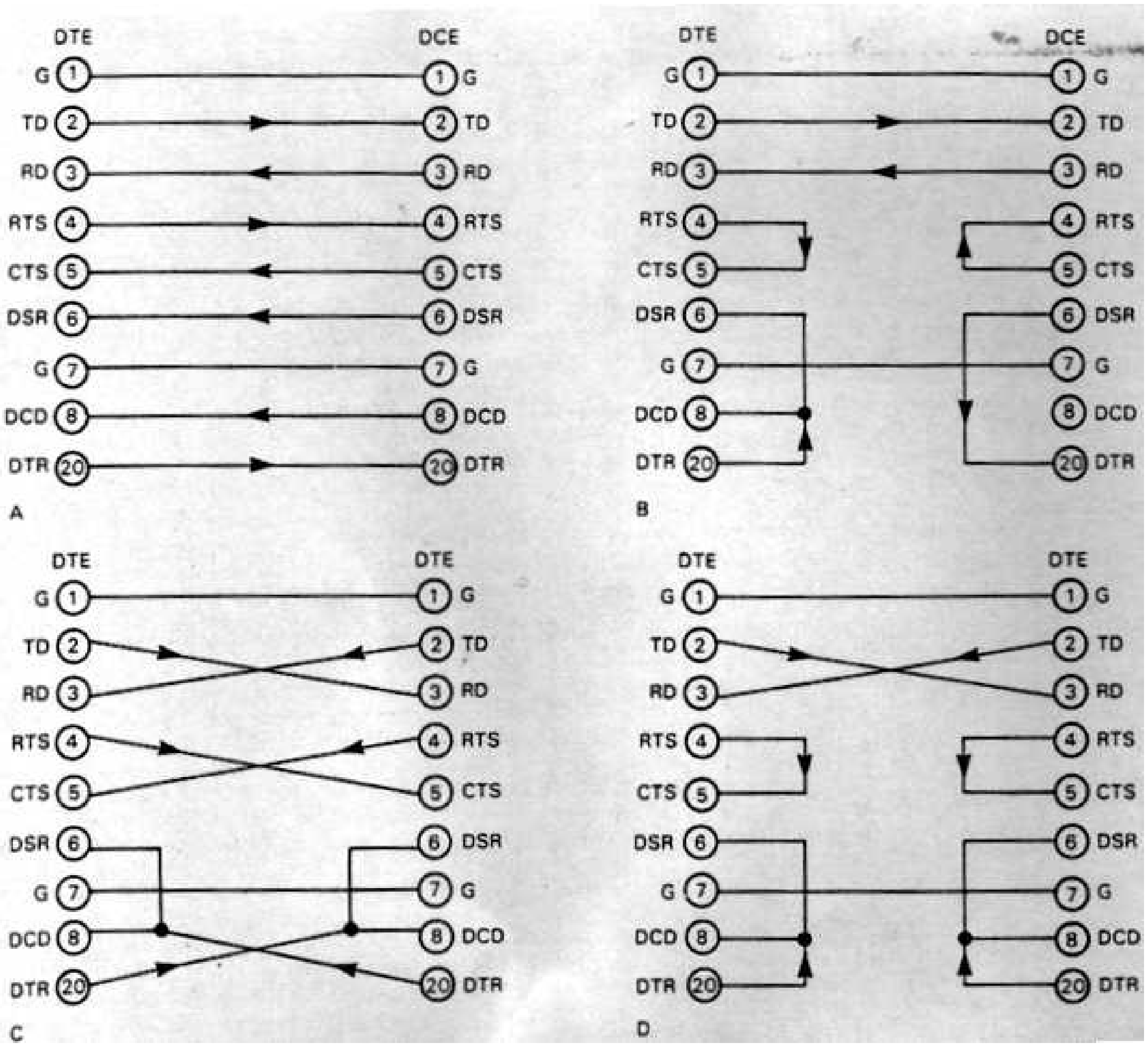
DTR (Data Terminal Ready) – Computer on
DSR (Data Set Ready) – Modem on

DCD (Data Carrier Detected) – Modem detects phone line carrier
RI (Ring Indicator) – Modem detects ring

RTS (Request to Send) – Computer ready to communicate
CTS (Clear To Send) – Modem ready to communicate

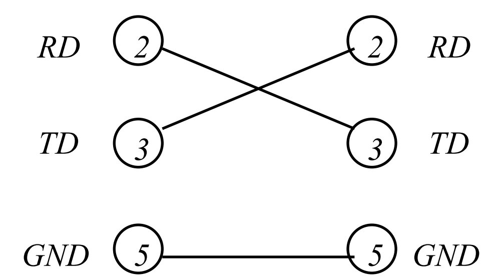
TD (Transmit data) – Data transmission
RD (Receive data) – Data reception

Connections between equipment



	25-pin	9-pin
protec. GND	1	—
TD	2	3
RD	3	2
RTS	4	7
CTS	5	8
DSR	6	6
sig. GND	7	5
DCD	8	1
DTR	20	4
RI	22	9

E



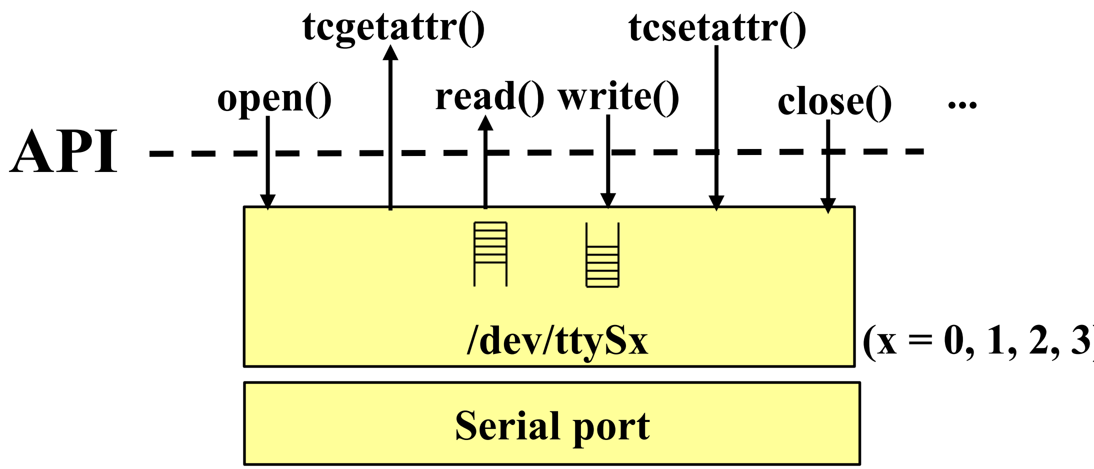
Null Modem (9 pinos)

Unix Drivers

- » Characteristics
  - – Software that manages a hardware controller
  - – Set of low level routines with privileged execution access
  - – Reside in memory (they are part of the kernel)
  - – Hardware interruption associated
- » Access method
  - – Mapped into Unix file system (/dev/hda1, /dev/ttyS0)
  - – Offered services similar to files (*open*, *close*, *read*, *write*)
- » Driver types
  - – Character
    - Read and write from the controlled as multiple characters
    - Direct access (data is not stored in buffers)
  - – Block
    - Read/write as multiples of a block (block = 512 or 1024 octets)
    - Data sorted in buffers and random access
  - – Network
    - Read and write variable size packets
    - Sockets interface

Serial Port Driver – API

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



Some API functions

int **open** (DEVICE, O_RDWR); /*returns a file descriptor*/
int **read** (int fileDescriptor, char * buffer, int numChars); /*returns the number of characters read*/
int **write** (int fileDescriptor, char * buffer, int numChars); /*returns the number of characters written*/
int **close** (int fileDescriptor);

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

trmios data structure – allows to configure and store the serial port configuration parameters

```
struct termios {
    tcflag_t  c_iflag;    /*reception configuration flags*/
    tcflag_t  c_oflag;    /*transmission configuration flags*/
    tcflag_t  c_cflag;    /*control flags*/
    tcflag_t  c_lflag;    /*local configuration flags*/
    cc_t      c_line;     /*not used*/
    cc_t      c_cc[NCCS]  /*control characteres; NCCS = 19*/
};
```

Example:

```
#define BAUDRATE B38400
struct termios newtio;

/* CS8:     8n1 (8 bits, without parity bit,1 stopbit)*/
/* CLOCAL:  local connection, without modem*/
/* CREAD:   enables the reception of characters*/
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

/* IGNPAR:  Ignores parity bits errors*/
/* ICRNL:   Converts CR into NL*/
newtio.c_iflag = IGNPAR | ICRNL;

newtio.c_oflag = 0;  /*Output not processed*/

/* ICANON:  enables the canonic reception */
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 the configuration of a maximum time between each character read
  - » Suitable for reading groups of characters
- Asynchronous
  - » *read*( ) returns immediately
  - » Uses a *signal handler*

Code examples

Canonic Reception

```
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);
}
```

Non-canonic 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; /* timer between characters
newtio.c_cc[VMIN] = 5;  /* block until 5 characters
                           are read */
tcflush(fd, TCIFLUSH);
tcsetattr(fd,TCSANOW,&newtio);

res = read(fd,buf,255); /* at least 5 characters */

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

Code examples

Asynchronous Reception

```
void signal_handler_IO (int status); /* signal
handler definition */
main() {
   /* variables declaration and serial port open */
    saio.sa_handler = signal_handler_IO;
    saio.sa_flags = 0;
    saio.sa_restorer = NULL;    /* obsolete */
    sigaction(SIGIO,&saio,NULL);
    fcntl(fd, F_SETOWN, getpid());
    fcntl(fd, F_SETFL, FASYNC);
   /* serial port configuration through the termios
structure */
  while (loop) {
      write(1, ".", 1);usleep(100000);
      /* after SIGIO signal, wait_flag = FALSE, data
available to read */
      if (wait_flag==FALSE) {
        read(fd,buf,255); wait_flag = TRUE; /*
waiting for new data to be read */
      }
    }
    /* configure the serial port with the initial
values and close */
}
void signal_handler_IO (int status) { wait_flag =
FALSE; }
```

Multiple Reception

```
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();
     }
  }
```