Data Link Protocol - some more details

(1st Lab Work)

FEUP

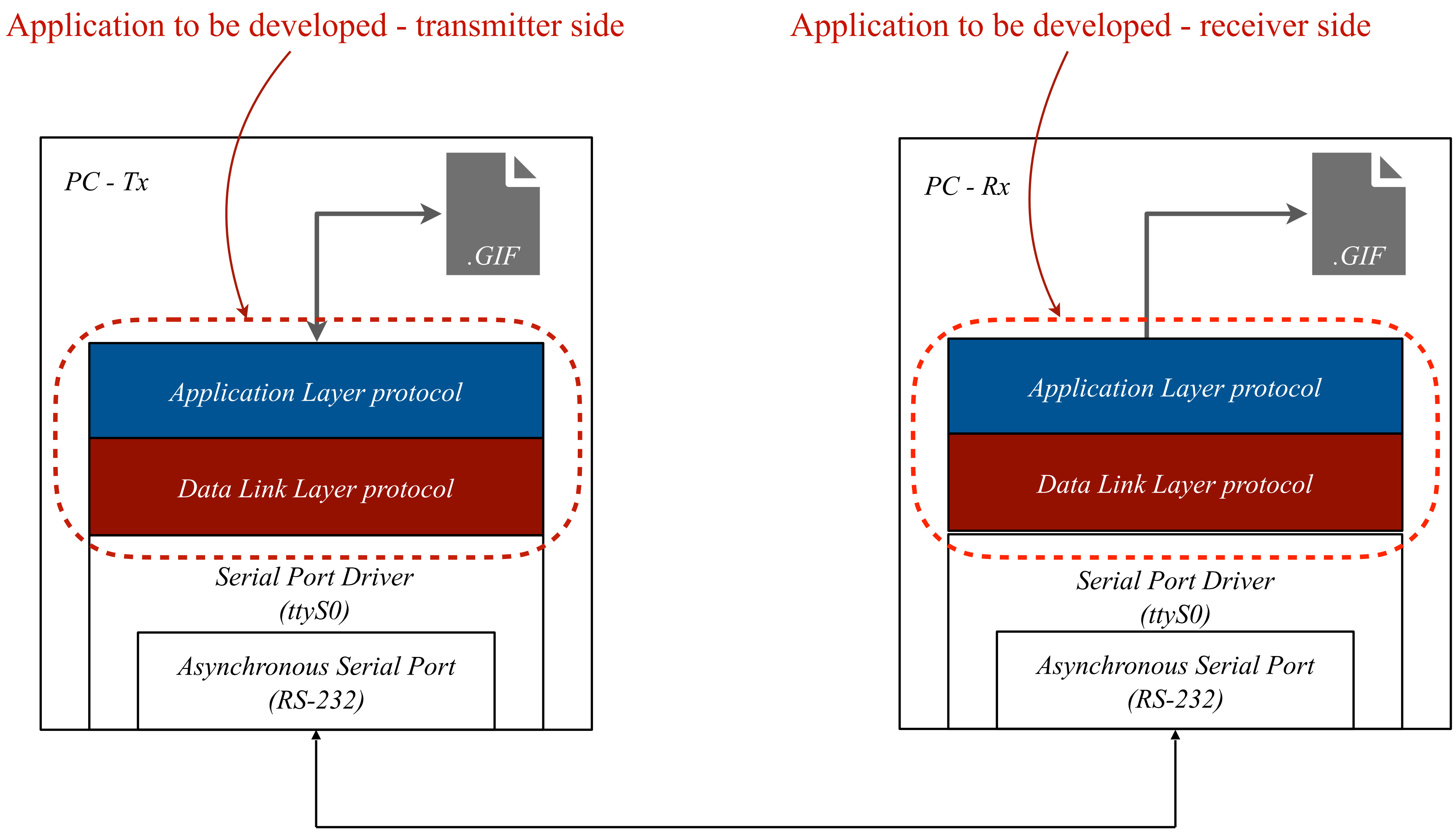
Computer Networks

Lab 1

- STOP-AND-WAIT communication protocol in a point-to-point link
  - application to transfer a file between 2 computers connected directly through a RS-232 serial cable
  - 2 main software components to be developed according to specifications described in the slides with the complete description of the work

1. Data Link Layer (DLL) Protocol for serial connections
   - protocol that specifies the syntax and semantics of messages to be exchanged between DLL transmitter and receiver to allow them to exchange a file using a RS-232 serial cable
2. Application Layer (AL) Protocol for transferring files
   - Protocol running at the AL that specifies messages to exchange data files between AL transmitter and receiver and the procedures to be executed at both ends
     - Uses/invokes the DLL protocol

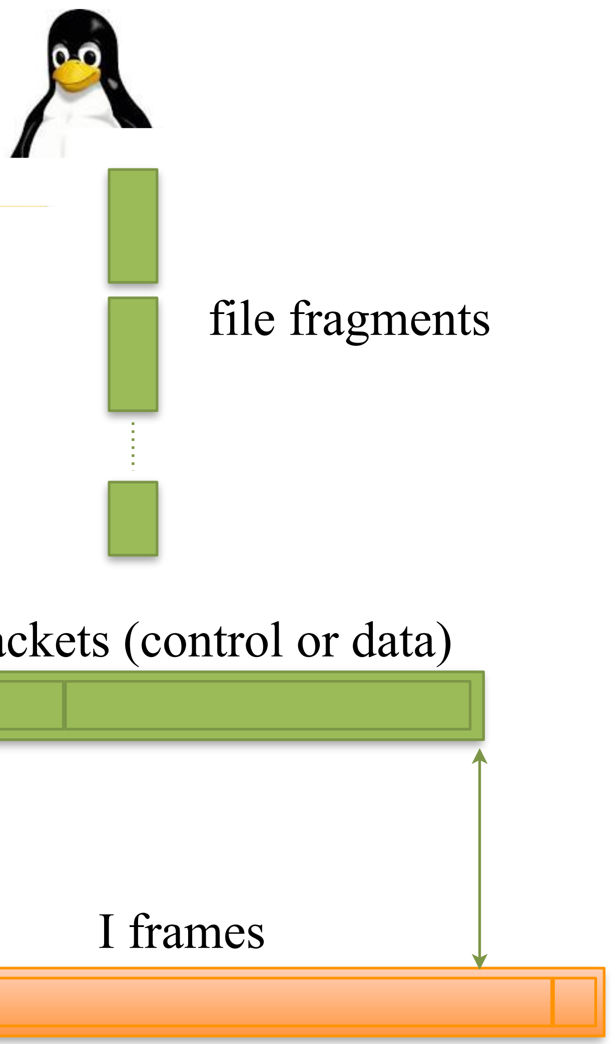
Test configuration



The process to transfer data

- What types of data are there?
    - fragments of bytes from the file to be transferred
    - control or signalling information understood uniquely by the protocol running at the layer that generated it
- How is data transferred?
    - Packed into packets (app layer) and frames (link layer)
- What types of packets/frames are there to send data?
    - » At the application layer
        - » CONTROL <u>packets</u>
        - » DATA <u>packets</u>
    - » At the data link layer
        - » Supervision <u>frames</u> (S) and Un-numbered acknowledgments (U)
        - » Information <u>frames</u> (I)

The process to transfer data (2)

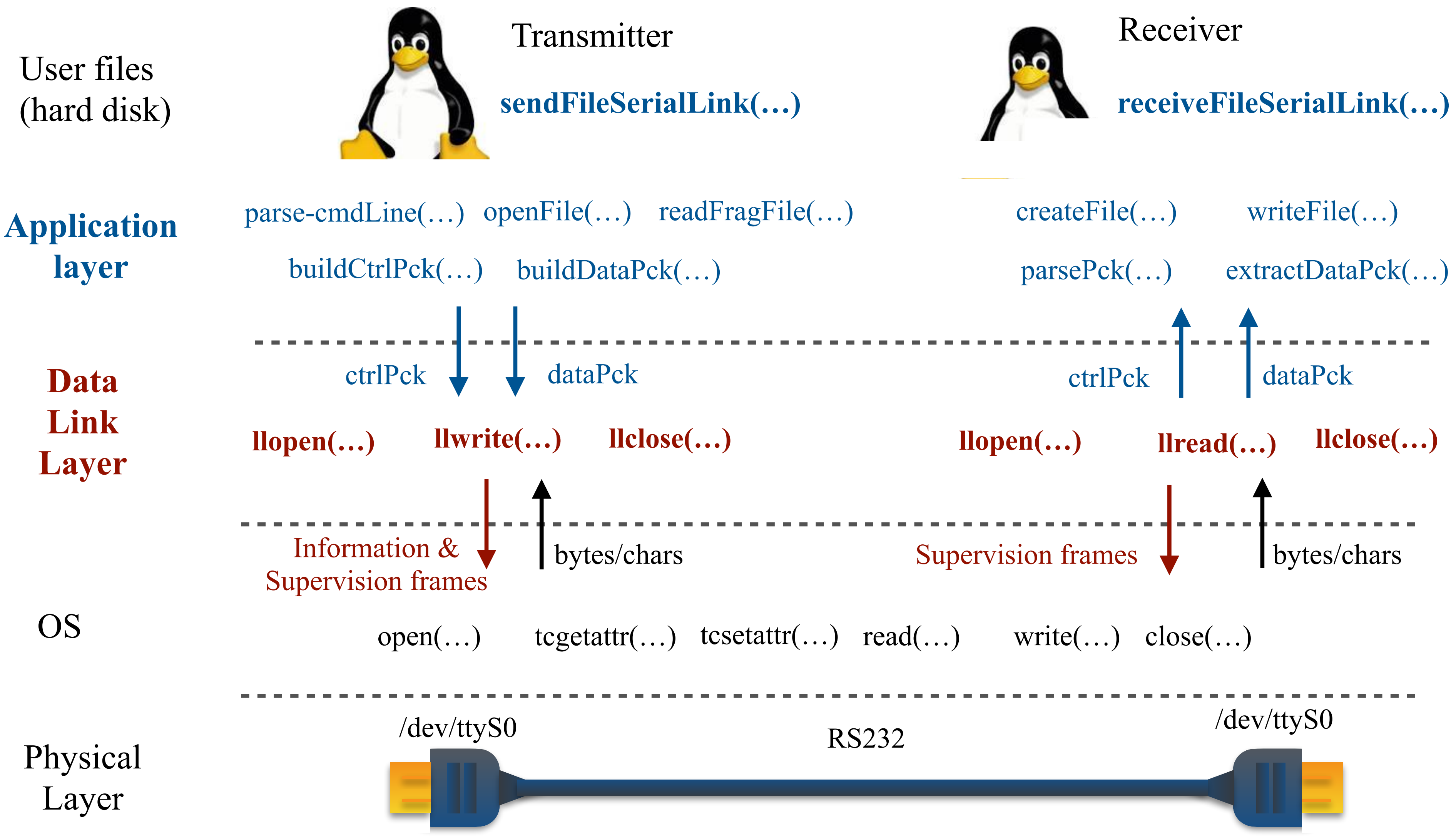


The process to transfer data (3)

- The transmitter application
  - » Announces to the receiver that a file is going to be sent
    - » Sends a START packet with name and size of file
      - » This is a type of CONTROL packet
  - » Takes the file and breaks it into smaller chunks (data fragments)
    - » Packs each segment into a DATA packet by adding an header
    - » Sends each DATA packet
  - » After sending the last DATA packet, announces to the receiver that the transfer is finalised
    - » Sends an END packet (another CONTROL packet)
  - » To send any type of packets, the application uses the API implemented by the Data Link Layer
  - » Packets have a well defined format

Functions and APIs at each layer to be developed

(merely indicative at the Application Layer)



Note: at the application layer, names of functions are only demonstrative

Code already developed that can be used

name	functions	purpose
main.c	`int main(int argc, char *argv[])`	main program: parses command line, defines serial port parameters and invokes the application layer main function. NOTHING TO DO here

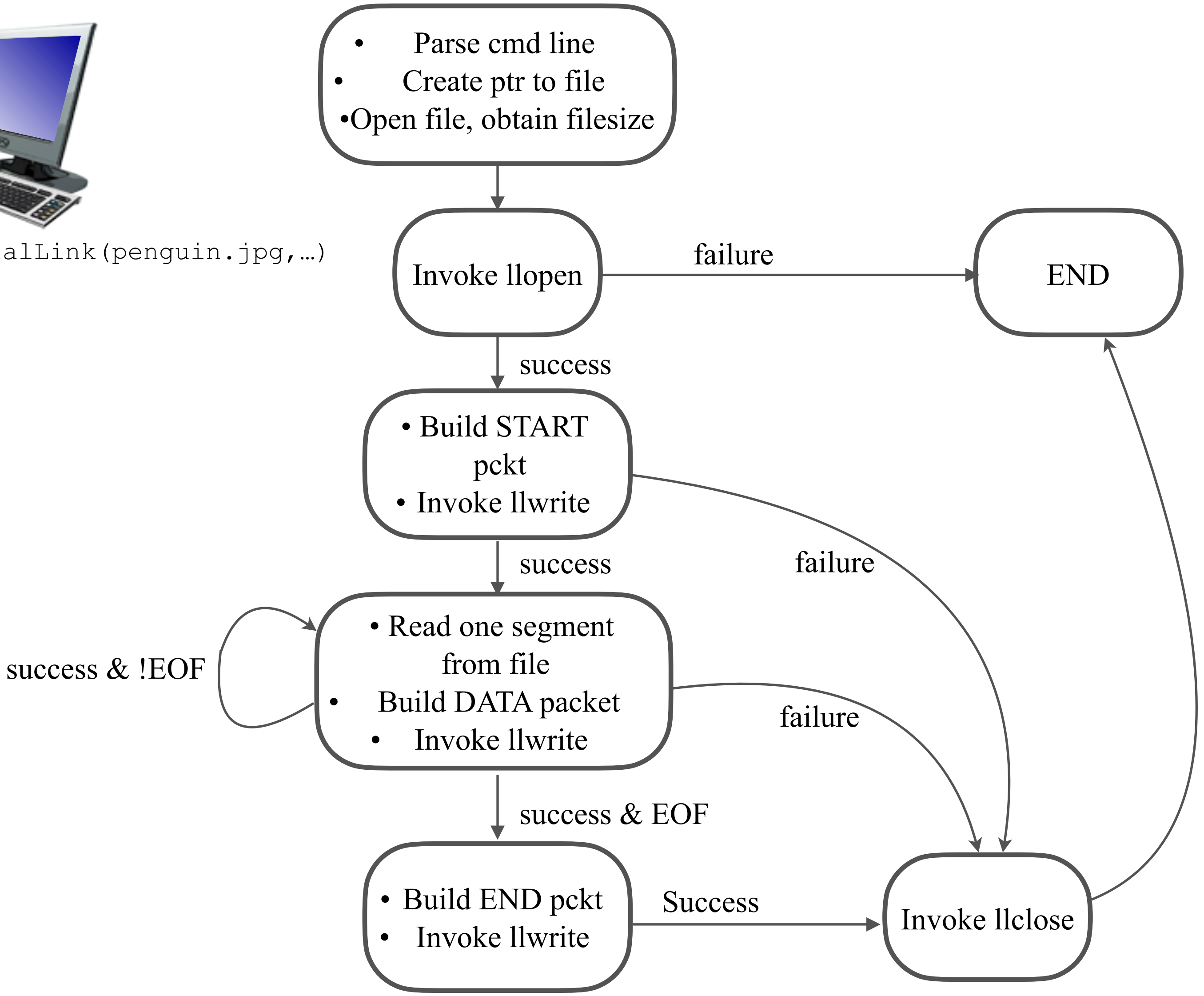
serial_port.c	`int` **`openSerialPort`**`(const char *serialPort, int baudRate)`
	`int` **`readByteSerialPort`**`(unsigned char *byte)`
	`int` **`writeBytesSerialPort`**`(const unsigned char *bytes, int numBytes)`
	`int` **`closeSerialPort`**`()`
		Implements the interface to the serial port: 1) with the serial port parameters it receives, opens and configures the serial port; 2) reads bytes from the serial port for reception (one at the time); 3) writes a specified number of bytes into the serial port for transmission; 4) when transmission is done, closes the serial port. NOTHING TO DO here

application_layer.c	`void` **`applicationLayer`**`(const char *serialPort, const char *role, int baudRate, int nTries, int timeout, const char *filename)`	where it all starts: 1) depending on the role Tx/Rx, implements and performs related file-level operations and invokes functions of the DLL (see slides 19-20 and 22-23). **TO BE DEVELOPED**

link_layer.c	`int` **`llopen`**`(LinkLayer connectionParameters)`
	`int` **`llwrite`**`(const unsigned char *buf, int bufSize)`
	`int` **`llread`**`(unsigned char *packet)`
	`int` **`llclose`**`()`
		where the DLL protocol is implemented: 1) invokes serial_port function to open and configure the serial port; 2) depending on the role Tx/Rx provides functionality for establishing connection; sending/ receiving frames and receiving/sending confirmation/ rejection; ending the connection (see slides 21 and 24). **TO BE DEVELOPED**

General operation of the Tx application layer

```
>sendFileSerialLink(penguin.jpg,…)
```



General operation of the **Tx application layer**

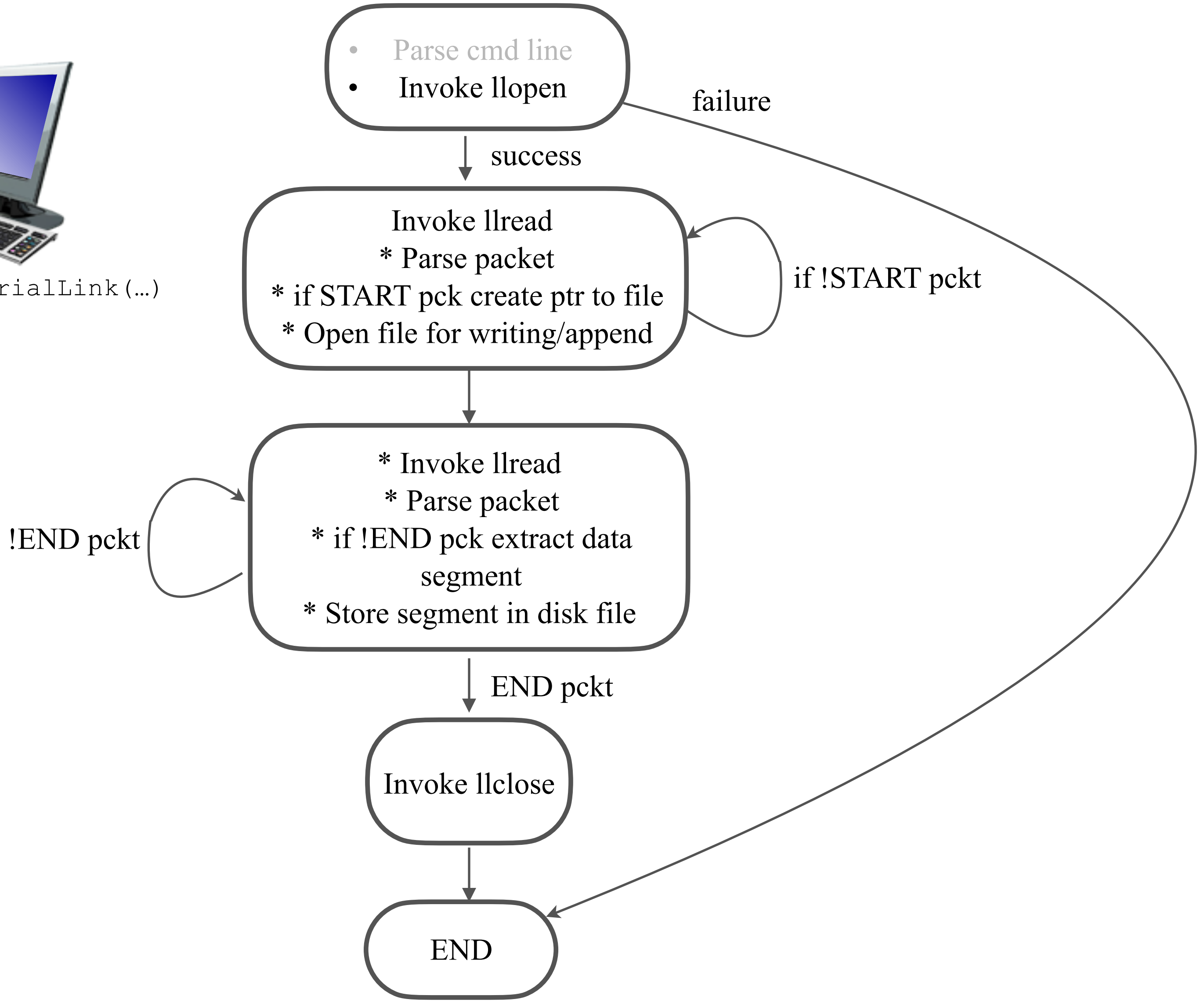
- Parse command line to extract parameters, namely, the name of the file to be transferred
- Invoke llopen
- Open file for reading and inspect its size
- Build control packet (START) with name and size of the file
- Enter a loop until the end of file is reached
  - Read one fragment of data from file
  - Build data packet
  - Invoke llwrite
- Invoke llclose

General operation of the Tx ***data link layer***

- llopen
  - opens serial port (invokes openSerialPort)
  - sends SET frame
  - reads one byte at a time to receive UA frame (see state machine in slide 40)
  - returns success or failure
- llwrite
  - implements error detection (compute BCC over the data packet)
  - parses data packet to implement byte stuffing (transparency)
  - builds Information frame Ns (Ns=0 or 1)
  - sends I frame
  - reads one byte at a time to receive response
  - if negative response (REJ) or no response, resends I frame up to a maximum number of times (retransmission mechanism explained in slide 46)
  - return success or failure
- llclose
  - sends DISC frame
  - reads one byte at a time to receive DISC frame
  - sends UA frame
  - closes serial port

General operation of the Rx application layer

```
>receiveFileSerialLink(…)
```



*General operation of the **Rx application layer***

- <Parse command line to extract parameters, namely, the name of the file to be received>
- Invoke llopen
- Enter a loop until the END packet is received
  - Invoke llread
  - Parse packet
  - if packet==control packet (START)
    - extract name and size of the file to be received
    - create file for writing/appending
  - else if packet== data packet
    - parse packet and extract fragment of data
    - write/append data segment in disk file
- Invoke llclose

General operation of the Rx ***data link layer***

- llopen
  - opens serial port (invokes openSerialPort)
  - read one byte at a time to receive SET frame
  - reply by sending UA frame
- llread
  - Enter the loop until a correct I frame is received
    - read one byte at a time to receive an I frame (see example state machine in slide 42)
    - compute BCC_1 over the frame header
    - if BCC1 correct check Ns
      - if expected Ns (new frame, no duplicate)
        - parse frane to implement byte de-stuffing
        - compute BCC2 over the data
        - if BCC2 correct
          - reply “send me new frame” by sending RR(Ns+1)
          - return data packet
        - else send REJ(Ns)
      - else, ignore or send RR(Ns)
- llclose
  - read one byte at a time to receive DISC frame
  - send DISC frame

Frames - Delimitation and header

» All frames are delimited by the flag F (**01111110 or 0x7E**)

» A frame is a sequence of data bytes enclosed within two flags

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

» Frames I, SET and DISC are designated Commands

» Frames UA, RR (0 or 1) and REJ (0 or 1) are called Replies

» As seen in the two previous slides, all frames have a header with a common format

F	A	C	BCC1

- A (Address) to distinguish whether the frame is a command or a reply
- C (Control) to distinguish between the different types of supervision frames and to carry sequence numbers N(s) in I frames
- BCC (Block Check Character), error detection mechanism

Transparency/stuffing - Necessity

- The transmission between the two computers is based on a technique called asynchronous transmission
    - Frames are transmitted asynchronously as a sequence of “characters”
        - 8-bit chars, that must be interpreted as simple octets or bytes
            - any of the 256 possible combinations can occur
            - inclusive the value of the delimiting flag!
    - To avoid the false recognition of a flag inside a frame, a mechanism that guarantees transparency is needed (stuffing or transparency)
        - If the value of the flag occurs within the data field, it will be replaced by an Escape sequence
- Note: Any data to be transferred needs to be defined as hexadecimals so the data structures must be of type “char” or “array of char” (unsigned to make sure it is only 8 bits)

Transparency/stuffing implementation

» the transmitter reads byte-by-byte the bock of data after computing BCC2 and:

- If the octet **01111110 (0x7e**) occurs inside the frame, it replaces it with the sequence **0x7d 0x5e**
- If the octet 01111101 (0x7d) occurs inside the frame, it replaces it by the sequence **0x7d 0x5d**

» he receiver, reads byte-by-byte and

- if it detects a 0x7d, inspect the next byte
  - if it is 0x5e, the pair 0x7d 0x5e is replaced by 0x7e and stored
  - if it is 0x5d, the pair 0x7d 0x5d is replaced by 0x7d and stored

Note: In the Tx, the BCC2 is competed before stuffing; in the Rx, the BCC2 is computed after de-stuffing

Receiving data (transmitter and receiver)

- » Given that frames are transmitted asynchronously and as a sequence of bytes
  - » delimited by one byte with a specific value (the flag F)
  - » in the header each byte has a special meaning by itself
- » data should be received and read out from the buffer of the serial port one by one
  - » the "read" function needs to be invoked in a way as to return at most one byte
    - » use the read function in non-blocking mode to read at most one byte from a serial port file descriptor (fd) into an unsigned char buffer buf

```
bytes_read = read(fd, &buf, 1);
```

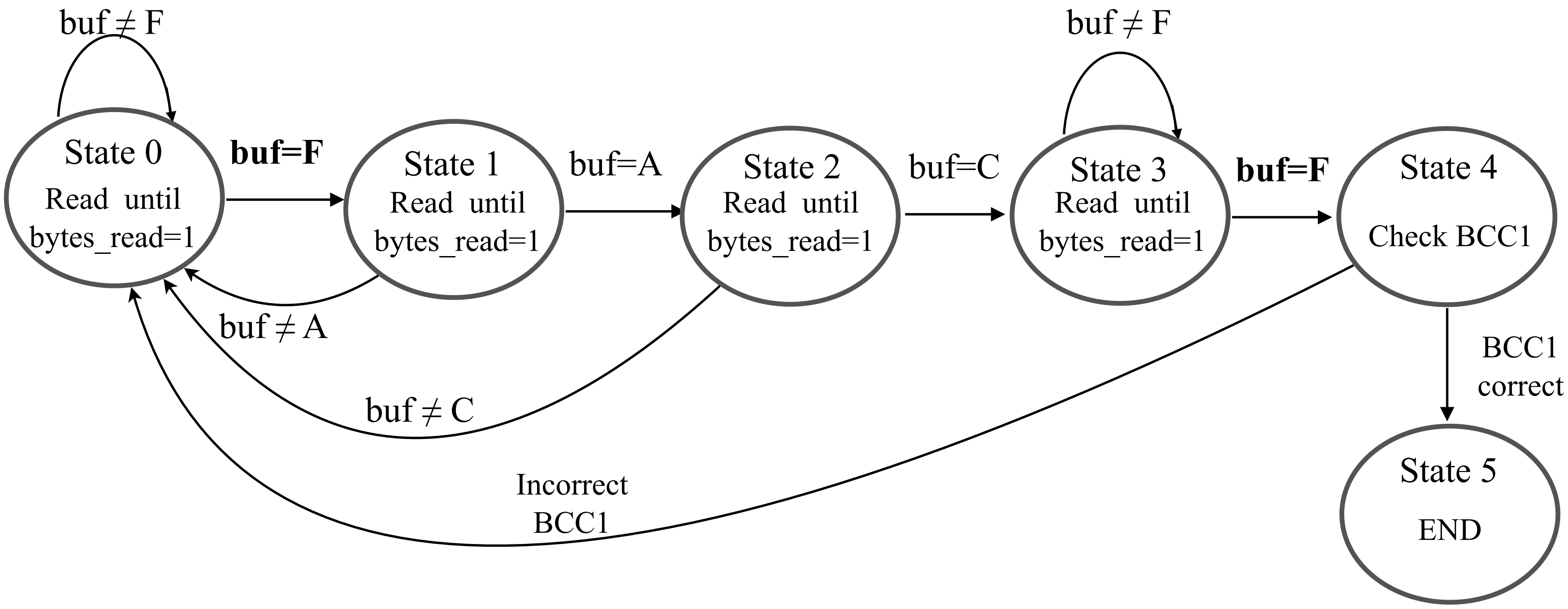
- » Note: given that transmission is asynchronous, there may be no data in the buffer when read is invoked
  - » thus bytes_read may be 0…
  - » Before trying to analyse what's in buf, check if bytes_read is greater than 0

Receiving Supervision frames (transmitter and receiver)

- Given that in the frame header each byte has a special meaning by itself
  - A state machine should be implemented to read the complete frame, byte-by-byte

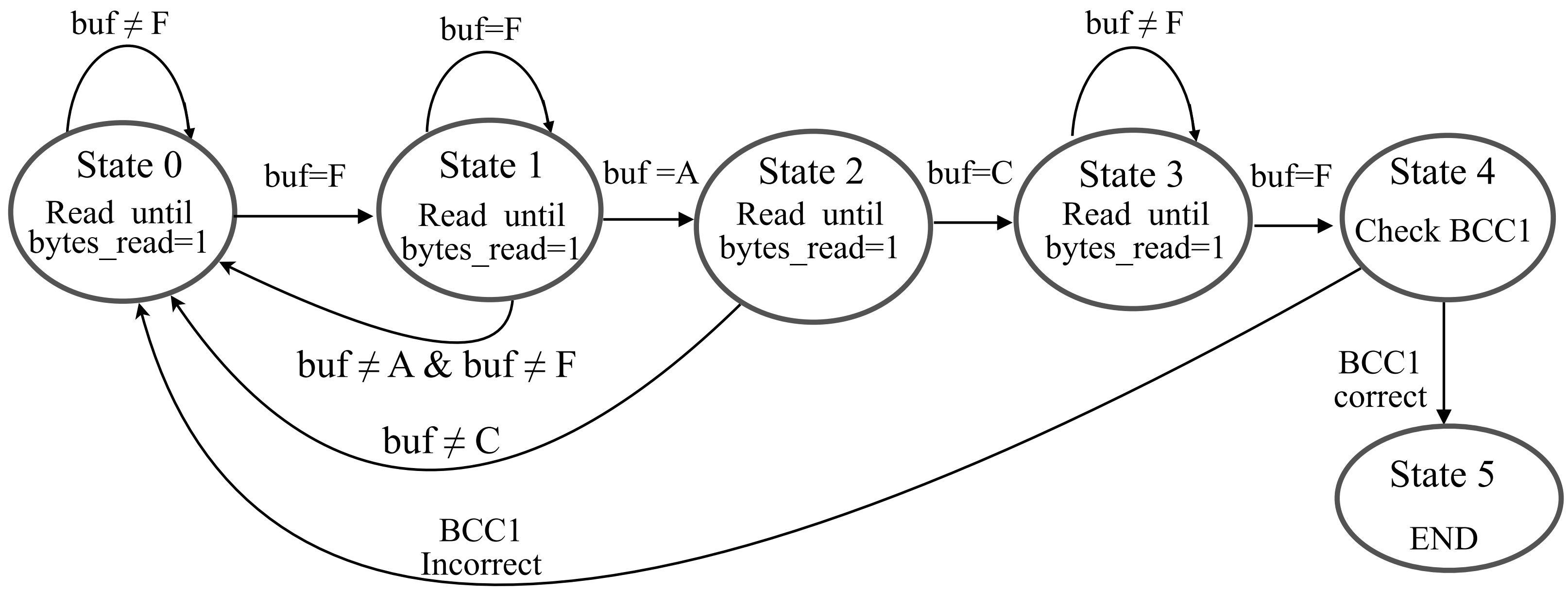
```
bytes_read = read(fd, &buf, 1);
```

    - from the starting Flag up to the closing Flag



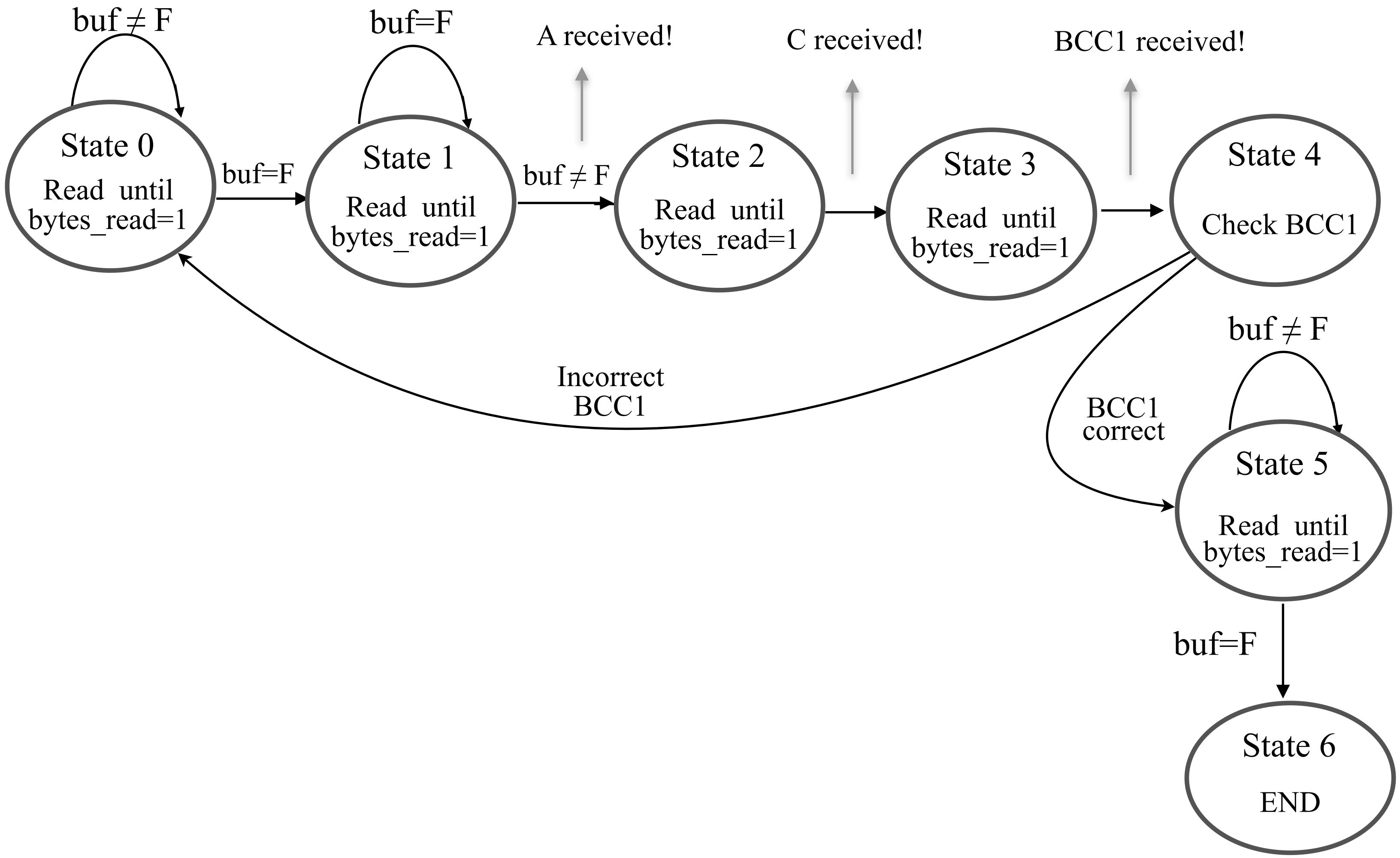
Receiving Supervision frames (transmitter and receiver)

» Variant considering the event that multiple flags are sent in sequence at the beginning



Receiving I frames (receiver)

» A variant of the state machine to receive I frames



Procedures for reception of frames - errors

- » <u>any frame</u> with wrong header (BCC1 incorrect) is ignored without any action
  - » applicable for transmitter and receiver
- » In the receiver, when receiving <u>I frames</u>, if BCC1 is correct
  - » if there is an error in the data field (BCC2 incorrect)
    - » the data field is discarded
    - » if it is a new frame, receiver sends REJ frame
    - » if it is a duplicate, receiver sends RR (0 or 1 according to the frame it expects)
  - » if there are no errors in the data field (BCC2 correct), the frame is accepted for processing
    - » receiver confirms it by sending a RR
    - » if it is a new frame, the data field is processed and passed to the Application layer
    - » if it is a duplicate, the data field is discarded