Report nr 1º of the lab work

RCOM

Nicolae Munteanu – 202202842
José Ramos – 202005460

This lab work was based on making a connection between two computers, through a physical cable, in the hope of sending a file (penguim.gif).
In this report we conclude that the project didn't go as expected and there are a lot of things that need to be fixed or upgraded, but still we are happy with the end result, because we worked hard and did all we could do with the time given.

Introduction

Has it was said previously, the main objective of this lab work was to send a file through a physical connection, but there were several other objectives throughout the project, such as having stages in sending the file (connecting, information transfer, disconnecting), the use of tramas changed in all of the stages so that we were able to distinguish between them and change from one stage to another, while sending the frames we had to check if the trama sent was correctly put together, if not we had to resend the data with the correct trama, there should be a application layer and a data layer, stop-and-wait so that we had time for the receiver to respond to our trama and we also had to stuff the data bytes in the transmitter so that we would not receive a flag by mistake on the receiver side which would have to then unstuff the bytes. Finally with this report we hope in explaining what we did and what we did not, while giving a explanation on how things work.

Architecture

Our project did not have a physical layer, but we have two logical link protocols, one for the receiver and one for the transmitter. In the receiver side we had the code divided by guards so that we could distinguish which state we were in, we had this in a while loop, except for when we disconnect and waiting for the UA. In the transmitter side we also had the code divided by guards so that we could know what state to use.

Code Structure

In the project we used the alarm given in the practical classes, it being used in the transmitter side throughout the whole program, readByByte was also used in the whole program both in the transmitter and receiver, checkSupervision was used to check if the supervision frames were correct and is used in both, although they are slightly different since the receiver doesn't receive some of the transmitter tramas and vice versa, clearBuffer was used to clear the buffer before it was going to get anything written on it, we did this because of some errors we were having, trama was used in to make supervision frames and it was used in both sides, checkData was used only in the transmitter to check if the trama in the information frames was correct, infoTrama does the intial trama for the information frames and we also have a swap which changes the Ns and Nr when necessary.

Main case usages

Both the receiver and transmitter are noncanonical. In the receiver side we allways start by reading the bytes received with readByByte and at the end we clear the buf with clearBuffer, after that, if we

are in the first state, we check if the frame is correct with checkSupervision, if it is then we create a message with trama and send it to the transmitter and go to state 1, else ignore the message and wait for the next message. When we are at state 1 we verify if the frame is valid for an information frame with checkData, if it is then go to state 2, else go back to state 0. After reaching state 2 we don't change state. In the state 2 we read the information frames until a disconnect appears, which we check at the beginning with checkSupervision, if we didn't disconnect then we clear the message variable with clearBuffer and call checkData and save the message received in message, then we create (with trama), send a response and use swap to change the Ns and Nr accordingly to the output of checkData, if we do disconnect then we clear the buffer with clearBuffer then do readByByte and lastly we do checkSupervision to check if we received the final frame.

In the transmitter side we start by putting and alarm with 5 seconds in which in repeats 3 times until time out. If the state is equal to 0 then we clear the buf with clearBuffer then create the frame with trama and send it with write, after that we clear the buffer again and wait 3 seconds so that we can then read the response with readByByte, if the response passes the checkSupervision it goes to state 1 otherwise it repeats the operation. In state 1 we start by checking if we had a bad connection (no response), if we have a good connection we clear the buf and repeat with clearBuffer, then we call infoTrama so that we have the first bytes of the frame in buf and the we start reading the file by blocks and stuff the content accordingly to specifications, after that we send the frame with write, if we have a bad connection we send the repeat which is a repeat frame that wasn't sent, sleep is then called for one second and we start readding with readByByte, and accordingly with the output of checkSupervision we call swap. When we reach the end of the file, we start the disconnection procedure by clearing the buf with clearBuffer, trama to write the message and then write to send it, after we sleep for one second, we check if the response passes the checkSupervision, if it does we clear the buf with clearBuffer, write the message with trame and send it with write, else we do the procedure again.

Logical Link Protocol

The logical link protocol has the main objective of providing reliable communication between two systems. It does so by splitting the data into chunks called frames, which are then sent in a safe and secure way; that's because all the frames must be acknowledged after they're received, and so no chunk of data is lost. The protocol also manages error detection and can check if something went wrong in the data transmission.
In order to create and delimit frames, some of the bytes are used at the start and at the end to express information. The beginning and end of a frame are delimited by a flag; there's also an address field, a control field that is used for checking the acknowledgement, rejection or other functionalities, and a protection field called BCC that checks for wrongly transmitted bytes in the frame. However, if a byte inside the data is by chance equal to the flag, it could lead to some problems.
That's why we stuff and de-stuff the frames: when sending the frame, we swap the "unintended" flag bytes with an escape character followed by the byte, so that we know it's just a character. If there's an unintended escape character, we do the same process too. The message is then de-stuffed by the receiver.

```
if (buffer[0] == 0x7e){
                    buf[numOfBytes] = 0x7d;
                    numOfBytes++;
                    buf[numOfBytes] = 0x5e;
                }
                else if (buffer[0] == 0x7d){
                    buf[numOfBytes] = 0x7d;
```

```
            numOfBytes++;
            buf[numOfBytes] = 0x5d;
        }
        else{
            buf[numOfBytes] = buffer[0];
        }
    }
```

For the retransmissions, there's a bunch of cases. If a message is sent but no acknowledgement is received, then an alarm will make the sender send the message again after some time passes. However, if the acknowledgement has incongruent Ns/Nr, the message has to be sent again too. The same thing happens if the message gets rejected.

```
if (checkSupervision(buf, 500, C_RR_NR0) == 1){
        printf("Connection good for now");
        alarmCount = 0;
        cycle = 0;
        swap();
    }
    else if (checkSupervision(buf, 500, C_RR_NR0) == 2){
        printf("Message repeated");
        alarmCount = 0;
        cycle = 0;
        count = count + 497;
        swap();
    }
    else if (checkSupervision(buf, 500, C_REJ_NR0)){
        printf("Message rejected by transmitter");
        alarmCount = 0;
        count = count + 497;
    }
```

Ns and Nr are global variables and, using the values 0 and 1, can check whether the message is the one that was expected, and can ask for the new one.
The error control is done with the BCC byte which is the XOR of all the precedent bytes, and can thus be used by the receiver to see if the BCC it calculates is correct.

Application Protocol

Because of the principle of layer independency, the application protocol doesn't know the internal details of the logical link protocol, it only accesses its service. It manages to give a more abstract interface to work with the logical layer, and thus it doesn't know anything about the internal delimitations of the frame, or of byte stuffing. However, having functionalities above the logical link layer, it can count and number data packets, and distinguish between which frames contain Data and which are for Control.
In this lab project we didn't develop two separate protocols, but we wrote a combination of both, with just a file for the sender and one for the receiver. The application knows that the writer only gets Supervision frames, and the receiver checks for Information frames or Supervision frames accordingly.

Validation

We tested the program both with virtual serial ports and with real physical connected computers, and it worked in both cases. Disconnecting the physical port and connecting it again during the transmission still guaranteed correct results. Working with the virtual serial ports meant having instant transmission; when we had to test it on real machines we had to adjust our work for it to read byte by byte, since the transmission speed was giving us problems.

Data-Link Protocol Efficiency

Our code works with a baud rate of 38400, which means that it can send 4800 bytes/s.
Our frames have a size of 500 bytes, and the computers are distant at most 1 meter from each other.

L = 500 byte
R (data rate) = 4800 bytes/s
d = 0.001 km
V = 4800 bytes/s
Tf = L/R = 0,104 s
Tp = d/V = 2,08e-7 s
a = Tp/Tf = 0,000002

Efficiency S = 1/(1+2a) = 0,9999960000159999
Which makes sense since the distance is very small and the speed is high, so this result matches our expectations.

Conclusions

Working on the development of this protocol was very insightful for many reasons. We got to experience first-hand how to implement communication in a functional way between two computers, and this helped consolidate the ideas studied in theory, since we now better understand the reasoning behind the implementation choices. Writing the code from scratch made the ideas come more naturally, and we had to get fully immersed into the subject in order to understand what worked and what didn’t. The implementation is not perfect since we didn’t separate the application and link layer, but for the rest we are satisfied with what we developed and what we learnt along the way.