

Faculdade de Engenharia da Universidade do Porto

**Data Link Protocol**

Instructor:

Eduardo Nuno Moreira Soares de Almeida

Redes de Computadores

Turma 15 – Grupo 2

- Clarisse Maria Teixeira de Carvalho (up202008444@fc.up.pt)

- Maria Eduarda Pacheco Mendes Araújo ([up202004473@fc.up.pt](mailto:up202004473@fc.up.pt))

Porto, 31 de outubro de 2024

**Summary:**

This project was carried out as part of the Computer Networks course and focuses on implementing a data communication protocol for file transfer over na RS-232 serial port.

foi realizado no âmbito da Unidade Curricular de Redes de Computadores

**Introduction:**

The goal of this project is to implement a data link layer protocol based on the provided specifications. This protocol enables a transmitter and receiver to transfer files stored on the hard disk between two computers connected via na RS-232 serial cable.

We developed and tested a data link protocol, in line with the specifications provided, for file transfers through a serial port.

The report is divided as follows:

* **Architecture:** Functional blocks and interfaces.
* **Code Structure:** APIs, main data structures, key functional and their relation to the architecture.
* **Main Use Cases:** Identification of core project functionalities, including function cal sequences.
* **Logical Link Protocol:** Logical connection functionality and implementation strategies.
* **Application Protocol:** Application layer functionality and implementation strategies.
* **Validation:** Tests performed to assess implementation correctness.
* **Data Link Protocol Efficiency:** Evaluation of the Stop & Wait protocol efficiency in the data link layer.
* **Conclusions:** Summary of information presented in previous sections and refletion on learning outcomes.

**Architecture:**

- Functional Blocks:

The project is structured around two primary layers: the **Link Layer** and the **Application Layer**

* *LinkLayer*: This layer is responsible for implementing the data link protocol and handling low-level communication over the serial port. Found in link\_layer.h and link\_layer.c, it manages essential tasks such as establishing and terminating connections, framing data, error detection, and handling acknowledgments. Key functions in this layer include llopen() for connection setup, llwrite() for data transmission, and llclose() for disconnection.
* Application Layer: Implemented in application\_layer.h and application\_layer.c, this layer provides higher-level functionality to facilitate file transfer operations. It leverages the Link Layer API to send and receive data packets through the serial port, handling file preparation and ensuring reliable transmission. Users interact with this layer to set parameters such as frame size, transfer speed, and retry limits.

- Interfaces:

Uma imagem com texto, captura de ecrã, Tipo de letra, documento

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, número, documento

Descrição gerada automaticamente The program is designed to operate on two computers, each connected via an RS-232 serial cable. It uses two terminals—one on each computer—to enable communication between a **transmitter** and a **receiver**. Each terminal runs the program in a specified mode.

* **Transmitter Mode**: Sends data frames to the receiver, ensuring they are correctly received and acknowledged.
* **Receiver Mode**: Receives, validates, and stores data frames, acknowledging successful reception or requesting retransmission if errors are detected.

**Estrutura do código:**

*LinkLayer:*

Uma imagem com texto, captura de ecrã, Tipo de letra

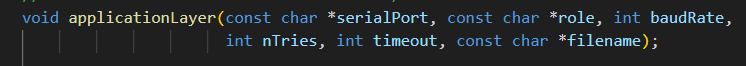
Descrição gerada automaticamente Two auxiliary data structures were used in this layer: LinkLayer, where the parameters associated with data transfer are characterized, and LinkLayerRole, which identifies whether the computer is a transmitter or receiver

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

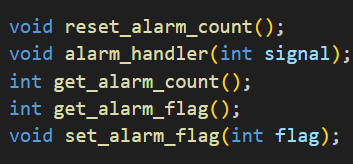
There were the functions implemented

*ApplicationLayer*

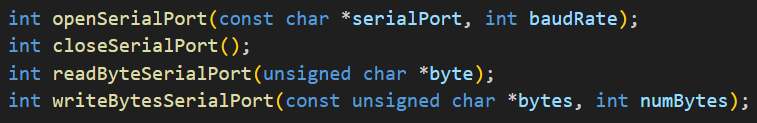


In implementing this layer, there was no need to create auxiliary data structures

*Alarm*



These were the functions implemented

*Serial Port*

There were the functions implemented

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente*State*

**Main use cases:**

The program can be run in transmitter and receiver modes. The functions to be used and the sequence of calls will differ depending on the choice made.

- Transmiter:

1. **Connection Setup (llopen)**:

* The transmitter initiates the connection by calling llopen(), which configures the serial port parameters.
* Then, start\_transmissor() sends a SET supervisory frame to request a connection.
* The transmitter waits for an acknowledgment (UA) from the receiver. If it receives the UA frame, the connection is established; otherwise, it retries.

1. **Data Transmission (llwrite)**:

* The transmitter reads the data to be sent and prepares an I-frame, including a sequence number and error-checking BCC fields.
* The frame is then "stuffed" to escape special characters before transmission.
* Using transmit\_information\_frame(), the frame is sent, and the transmitter waits for an acknowledgment.
* If the acknowledgment is positive (matching sequence number), the transmitter proceeds with the next frame; if negative, it retries up to a maximum of three attempts.

1. **Disconnection (llclose)**:

* Once the file transfer is complete, the transmitter initiates disconnection by sending a DISC frame.
* After receiving a DISC acknowledgment, it sends a UA frame to confirm termination and then closes the serial port

- Receiver:

The program can be run in transmitter and receiver modes. The functions to be used and the sequence of calls will differ depending on the choice made.

1. **Connection Setup (llopen)**:

* The receiver waits for a SET frame from the transmitter.
* Upon receiving it, the receiver responds with a UA frame, establishing the connection.

1. **Data Reception (llread)**:

* The receiver waits for incoming I-frames.
* For each received frame, the llread() function destuffs the data and checks the BCC fields for errors.
* If the data is valid and the sequence number matches, the receiver stores the data and sends a positive acknowledgment (PA). If an error is detected, the receiver sends a rejection (REJ) frame.

1. **Disconnection (llclose)**:

* Upon receiving a DISC frame from the transmitter, the receiver responds with a DISC frame.
* Finally, it waits for a UA acknowledgment before closing the serial port connection.

**Logical Link Protocol:**

The data link layer in this project serves as the interface for direct communication between the transmitter and receiver via the serial port, implementing a Stop-and-Wait protocol to ensure reliable data transfer. This layer manages the setup, data transmission, error-checking, and termination of the communication.

* **Connection Establishment**:  
  The connection is initialized by the **llopen()** function, which configures the serial port and initiates communication. In transmitter mode, **llopen()** sends a SET (setup) supervisory frame to request a connection. The receiver responds with a UA (unnumbered acknowledgment) supervisory frame, signaling successful connection establishment. Upon receiving the UA frame, the transmitter proceeds with data transmission.
* **Data Transmission**:  
  File data is transmitted frame-by-frame using the **llwrite()** function. Each data frame includes sequence numbers and error-checking fields (BCC1 and BCC2) for data integrity. The transmitter waits for an acknowledgment (ACK) from the receiver after each frame is sent. If the receiver detects an error (e.g., due to transmission noise or corruption), it sends a REJ (reject) frame instead, prompting the transmitter to retransmit the affected frame. The Stop-and-Wait mechanism ensures only one frame is sent at a time, providing reliable, ordered delivery.
* **Data Reception**:  
  On the receiver side, data is processed by the **llread()** function, which reads each incoming frame from the serial port. It first removes any "stuffed" escape sequences (added to avoid misinterpreting special characters) and then validates BCC1 and BCC2. If validation succeeds and the sequence number matches, the receiver sends a positive acknowledgment, allowing the transmitter to proceed with the next frame. If validation fails, a REJ frame is sent to request retransmission.
* **Connection Termination**:  
  Once all frames are successfully transferred, the transmitter terminates the session by invoking **llclose()**, which sends a DISC (disconnect) frame to initiate the termination process. The receiver responds with a DISC frame, and upon acknowledgment with a final UA frame, the serial port is closed, completing the file transfer.

This structure ensures that each stage of communication—from setup to transmission, error handling, and disconnection—follows a reliable, organized protocol that can effectively handle data errors and connection disruptions.

**Application Protocol:**

The application layer performs three main tasks: managing file operations, creating and interpreting control and data packets, and coordinating the transmission with the data link layer. The functions are described below.

**Main Functions:**

1. **processReceivedPacket()**: Interprets received packets and manages file operations:
   * Handles three packet types: DATA, START CONTROL, and END CONTROL.
   * For DATA packets, verifies size and writes data to the destination file.
   * For START and END CONTROL packets, opens and closes the destination file, respectively.
2. **createStartControlPacket()** and **createEndControlPacket()**: Generate control packets with metadata for the file being transferred:
   * Encapsulate file information, such as size and name, into a start control packet.
   * The end control packet signifies the completion of the file transfer.
3. **createDataPayloadPacket()**: Forms data packets for file content:
   * Adds packet metadata, such as sequence number and data length, ensuring error-checking mechanisms.
4. **applicationLayer()**: Manages the transfer process and coordinates with the link layer:
   * **Transmitter Mode**: Prepares file data, creates packets, and transmits them using llwrite().
   * **Receiver Mode**: Receives packets, validates them, and reconstructs the file using llread() and processReceivedPacket().

**Transmission Flow:**

1. **Connection Setup**: llopen() initializes the serial connection.
2. **File Transfer**: The transmitter reads the file, packetizes it, and sends it sequentially. The receiver validates and stores received packets.
3. **Disconnection**: llclose() closes the connection after transfer completion.

**Validation:**

During the project, several tests were conducted to evaluate the robustness and accuracy of the implemented data link protocol, especially under adverse conditions. These tests were designed to simulate real-world scenarios where data communication may experience interruptions, noise, or errors.

1. **Partial and/or Total Interruption of the Serial Port**:
   * **Objective**: To test how the protocol responds to sudden loss of connection, which may occur due to hardware disconnection or failure.
   * **Procedure**: During file transfer, the serial port connection was intentionally interrupted at random intervals. We monitored how the protocol handled these interruptions, including any reconnection attempts and error messages logged.
   * **Expected Outcome**: The protocol should detect the loss of connection, attempt reconnection (up to the maximum retry limit), and log an appropriate error if reconnection fails.
2. **Introduction of Noise via a Short Circuit**:
   * **Objective**: To test the protocol’s resilience against electrical interference, which is common in real-world environments.
   * **Procedure**: A short circuit was introduced in the serial port during data transmission to simulate noise. We observed the receiver’s response, checking if it could detect and handle corrupted frames.
   * **Expected Outcome**: The protocol should detect data errors through checksum verification (BCC fields) and request retransmission of corrupted frames, ensuring data integrity.
3. **Rejection of Data Frames by Introducing Random Errors**:
   * **Objective**: To test the protocol’s handling of random transmission errors and its ability to manage frame rejection and retransmission.
   * **Procedure**: Errors were randomly injected into data frames to trigger rejections by the receiver. We observed if the receiver sent a REJ (reject) frame and if the transmitter properly retransmitted the affected frames.
   * **Expected Outcome**: The protocol should correctly reject corrupted frames, log the errors, and handle retransmission attempts until the data is accurately received or the maximum retries are reached

**Pending Validation**

Although these tests were conducted independently, they have not yet been reviewed in the presence of the instructor. A formal presentation in the laboratory is planned, where further feedback and validation may be obtained.

**Data Link Protocol Efficiency:**

- *Baudrate* variation

- Frame size variation

- Error rate variation

**Conclusions:**

With this project we were able to consolidate what we had learned in theory classes. This project made us understand that *LinkLayer* was responsible for interacting with the serial port and managing the information webs and that *ApplpicationLayer* was responsible for the layer that interacted directly with the file that was going to be transferred.