IoT Device Communication System Using C++ with Yocto for QEMU

Project Overview:

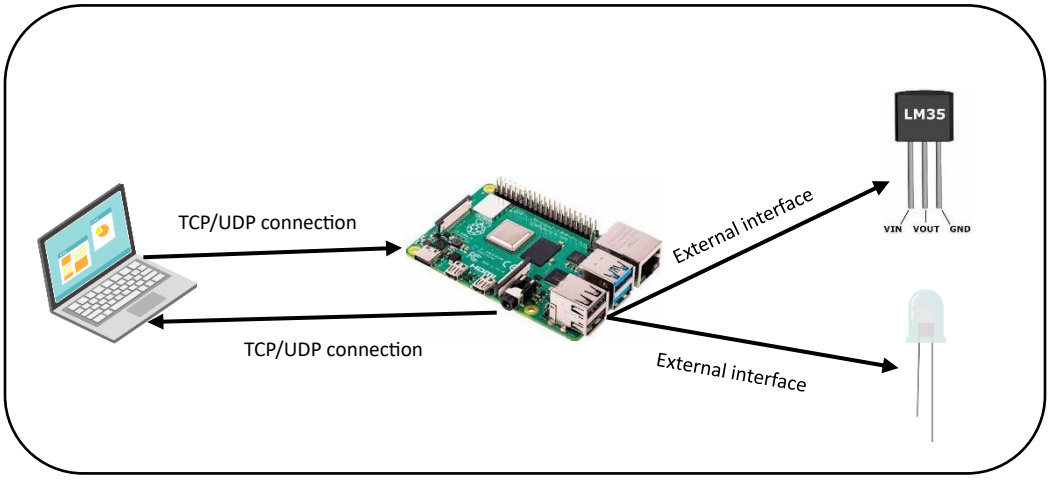
In this version, the system is fully emulated using QEMU instead of deploying on Raspberry Pi 4. The client application will run in a QEMU-emulated terminal, and no physical temperature sensor or LED is used. Instead, the client simulates sensor readings via user input and displays LED status textually. Communication between the client and a central server—running on a laptop with a Qt6 GUI—occurs over both TCP and UDP sockets and is implemented in C++ using object-oriented programming principles. The server periodically checks for threshold changes and instructs the client accordingly.

Project Structure:

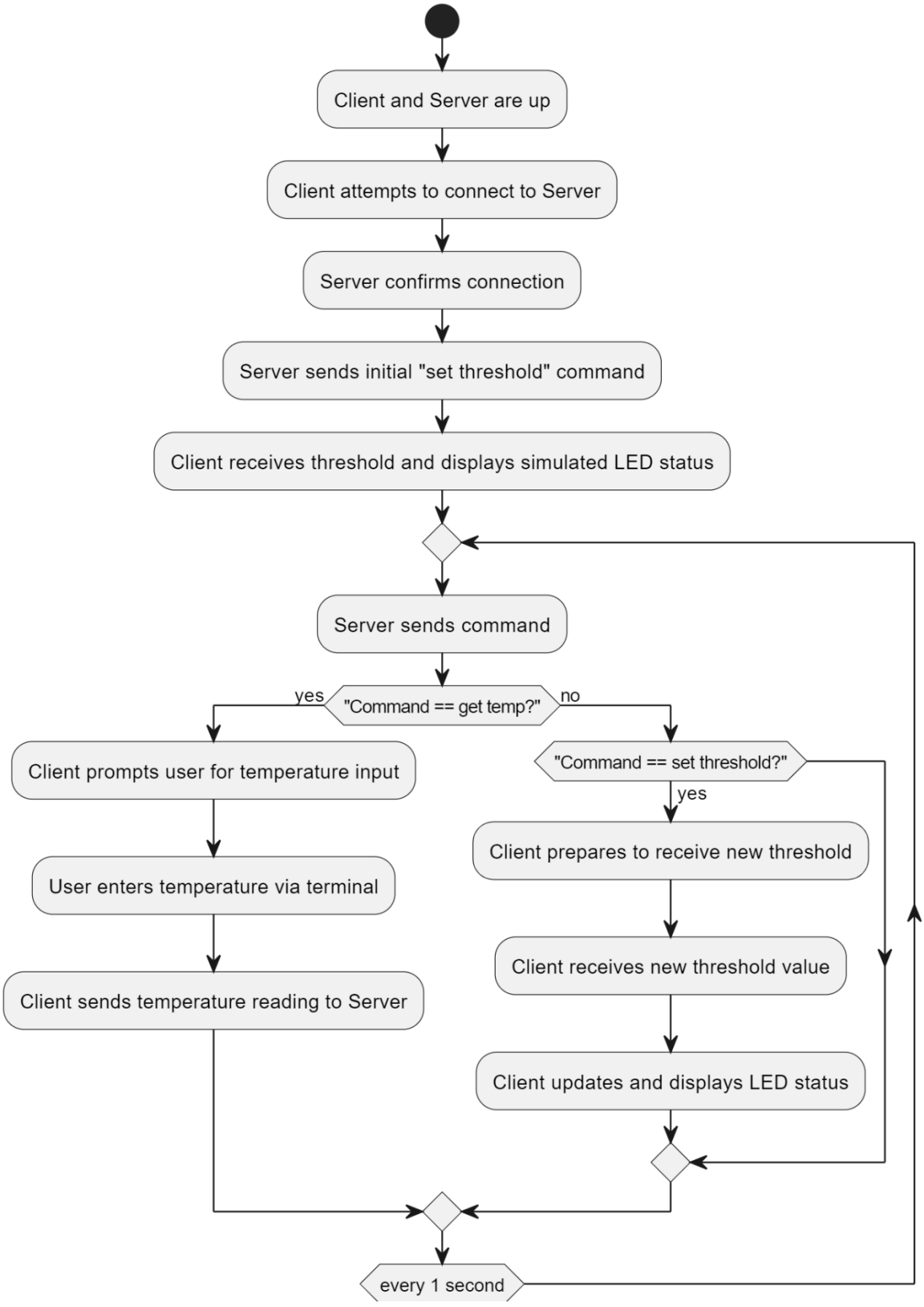
The project is divided into two main parts:

- **Unicast TCP/UDP Server/Client Application**
  - **Server:** Laptop with a Qt6 GUI.
  - **Client:** QEMU-emulated terminal application.

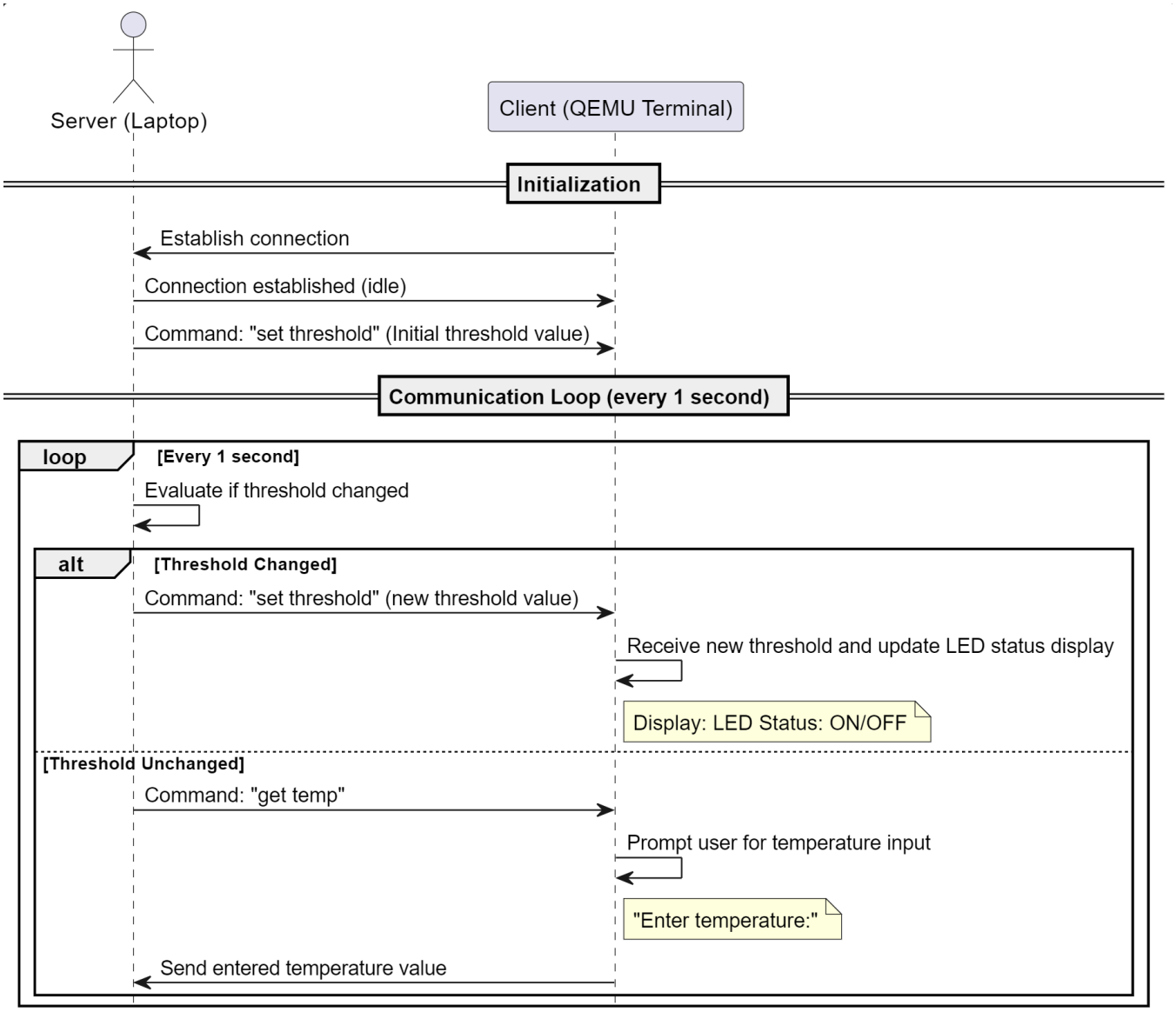
Project Block Diagram:



Project Flow Chart Diagram:



Project Sequence Diagram:



Deliverables:

- Full source code implementing the project.
- Makefile for automating the build process for both the server and client applications.
- Qt6 project files and source code for the server application.
- Project demo **video (is a Must)**

Client Application (QEMU with Terminal Interface):

Task:

- Develop a C++ application to be emulated using QEMU, featuring a terminal interface.
- **Initial Behavior:**
  The client starts up and immediately attempts to connect to the server.
- **Communication Logic:**
  - Once connected, the client waits for the server to set the temperature threshold.
  - Upon receiving the threshold, the client displays the simulated LED status (ON/OFF) on the terminal.
  - Then the client continuously waits for further messages from the server:
    - **If the server sends a "get Temp" command:**
      The client prompts the user via the terminal to enter the current temperature. The entered value is then sent to the server, and the client waits for the next instruction.
    - **If the server sends a "set threshold" command:**
      The client prepares to receive the new threshold value from the server. After receiving it, the client updates and displays the LED status accordingly, then resumes waiting for further commands.

Sample Output (Client Terminal):

- *Client:* "Attempting to connect to server..."
- *Client:* "Connected. Awaiting threshold setting..."
- *Client:* "LED Status: OFF" (simulated status)
- *Client:* "Received command: get Temp"
- *Client:* "Enter temperature: " *(user enters value, e.g., 35°C)*
- *Client:* "Sending temperature: 35°C"
- *Client:* "Received command: set threshold"
- *Client:* "Received new threshold: 40°C – LED Status: ON"

Server Application (Laptop with Qt6 GUI):

Task:

- Develop a C++ server application with a Qt6 GUI that listens for client connections over both TCP and UDP.
- **Communication Logic:**
  The server automatically queries every second to check if the critical temperature threshold has changed:
  - If the threshold has changed, it sends a "set threshold" command to the client so that the new threshold can be communicated.

- Otherwise, it sends a "get Temp" command to request the current temperature from the client.

- The GUI consists of Four tabs, **please refer to Resource.pdf file:**
  - **Tab 1: Real Time Monitor**
    Displays a simulated temperature meter showing the current temperature.
  - **Tab 2: Historical Analysis**
    Presents a 2D graph displaying historical temperature values received from the client.
  - **Tab 3: Configuration**
    Allows the user to set and calibrate the temperature threshold.
  - **Tab 4: Quick Access**
    This tab has 4 Buttons with ICON {Facebook, LinkedIn, Instagram}, upon clicking on it equivalent site will open up

- Additionally, the GUI displays the LED status (simulated on-screen) based on the latest threshold settings.

Communication Protocol:

- Implement both TCP and UDP socket communication for the server/client connection.
- **Server-Driven Interaction:**
  Every second, the server evaluates whether the critical temperature threshold has changed:
  - If **yes**, it sends a "set threshold" command so that the client prepares to receive the new threshold.
  - If **no**, it instructs the client to send the current temperature by issuing a "get Temp" command.
- All socket communication is encapsulated within object-oriented C++ classes.

OOP Design:

Enforce object-oriented programming (OOP) concepts such as **Abstraction, Encapsulation, Inheritance, and Polymorphism** throughout the project.

- **Abstract Class: Socket**
  Responsibilities:
  Define common socket behaviors (send, receive, connect, shutdown) with pure virtual functions.
  Functions:
  - virtual void waitForConnect() = 0;
  - virtual void connect() = 0;
  - virtual void send(const std::string& message) = 0;
  - virtual void receive() = 0;
  - virtual void shutdown() = 0;

- **Derived Classes: TCPSocket and UDPSocket**
  Inherit from *Socket* and implement protocol-specific behaviors for TCP and UDP communication.
- **Abstract Class: Channel**
  Data Member:
  - Socket* channelSocket; – A pointer to a *Socket* object (either *TCPSocket* or *UDPSocket*).
    Responsibilities:
    Handle socket interactions through its *channelSocket* member.
    Functions:
  - virtual void start() = 0;
  - virtual void stop() = 0;
  - virtual void send(const std::string& message) = 0;
  - virtual void receive() = 0;
- **Derived Classes: ServerChannel and ClientChannel**
  Implement specific server and client behaviors by utilizing the *channelSocket* to manage communication.

Development Environment:

- **QEMU Setup:**
  The client application will be emulated using QEMU with a terminal interface. The QEMU image will be produced using YOCTO.
- **Qt6 Setup:**
  Develop the server application using Qt6 to provide an intuitive and interactive GUI.
- **Makefile:**
  Provide a well-structured Makefile to automate the build process for both the server and client applications.

Tools and Technologies:

- **C++ (OOP):**
  For implementing socket communication and GUI logic while enforcing key OOP principles.
- **Qt6:**
  For building the server GUI application.
- **YOCTO:**
  For producing QEMU image include cpp layer added for it.
- **QEMU:**
  For emulating the client application with a terminal interface.
- **Socket Programming:**
  For implementing TCP and UDP communication between the client and server.

Thank You
Edges For Training Team