### **Lab 10:** **Network Programming in Rust**

**Exercise: Simple Echo Server-Client**

In this exercise, We will create a simple echo server and a client application using TCP sockets. The client will send a message to the server, and the server will respond with the same message, acting as an "echo."

1. Create a new Rust project using cargo:
2. Open your terminal/command prompt and run the following command:

cargo new network\_echo

cd network\_echo

1. Open the main.rs file in the src directory of your project. You can use any code editor for this.
2. Implement the echo server:

use std::net::{TcpListener, TcpStream};

use std::io::{Read, Write};

fn handle\_client(mut stream: TcpStream) {

let mut buffer = [0; 512];

loop {

match stream.read(&mut buffer) {

Ok(n) if n > 0 => {

stream.write\_all(&buffer[..n]).expect("Failed to write data");

}

\_ => break,

}

}

}

In the main function, set up the server to listen on a specific IP address and port:

fn main() {

let listener = TcpListener::bind("127.0.0.1:8080").expect("Failed to bind address");

for stream in listener.incoming() {

match stream {

Ok(stream) => {

std::thread::spawn(|| handle\_client(stream));

}

Err(e) => {

println!("Error: {}", e);

}

}

}

}

Save the file and return to your terminal/command prompt.

Build and run your server using cargo run:

cargo run

The server will now be running and listening for incoming connections on 127.0.0.1:8080.

Open another terminal/command prompt window and navigate to the same directory as the project.

Implement the echo client:

Create a new file named client.rs in the src directory with the following content:

use std::net::TcpStream;

use std::io::{Read, Write};

use std::io;

fn main() {

let mut stream = TcpStream::connect("127.0.0.1:8080").expect("Failed to connect to server");

loop {

println!("Enter a message to send (or 'q' to quit):");

let mut input = String::new();

io::stdin().read\_line(&mut input).expect("Failed to read input");

if input.trim() == "q" {

break;

}

stream.write\_all(input.trim().as\_bytes()).expect("Failed to write data");

let mut buffer = [0; 512];

stream.read(&mut buffer).expect("Failed to read response");

println!("Server Response: {}", String::from\_utf8\_lossy(&buffer));

}

}

Save the file and return to your terminal/command prompt.

Build and run your client using cargo run:

cargo run --bin client

The client will now prompt you to enter a message. Type a message and press Enter to send it to the server. The server will echo the message back to the client, and the client will display the server's response.

**To quit the client, type 'q' and press Enter.**

We have now successfully completed the lab exercise on network programming in Rust! We learned how to create a simple echo server and client using TCP sockets. Network programming in Rust enables you to build powerful applications that communicate over networks efficiently.

**Happy coding!**