A close-up of a coin

Description automatically generated with low confidence

**计算机网络实验报告**

实 验 题 目 Socket编程实现客户端与服务器通信

姓名 孙潇桐

专业 软件工程

班级 软工二班

学号 2021117405

西 北 大 学 信 息 学 院

1. 实验目的
   1. 学习Socket编程基础知识；
   2. 理解并学会应用socket API，比如，学习基于C的Socket编程相关函数和数据类型；（实验时，语言不限，代码可以是任意语言）
   3. 熟练掌握TCP的Client/Server模式的通信原理。
   4. 学会建立主机与服务器之间的连接；
   5. 学会在主机和服务器之间收发消息；
2. 实验环境

Windows 11, Rust语言环境

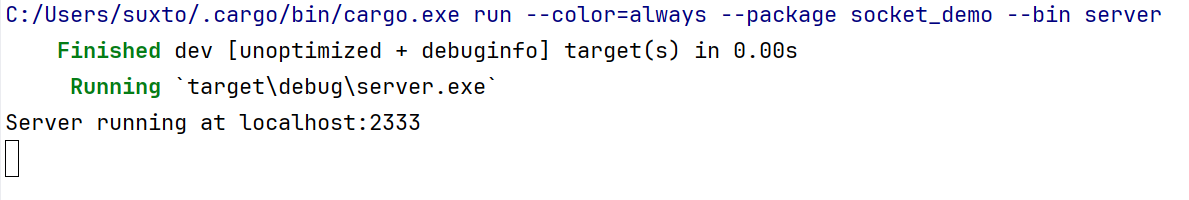
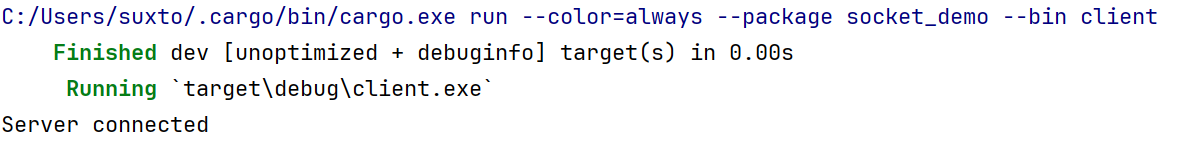
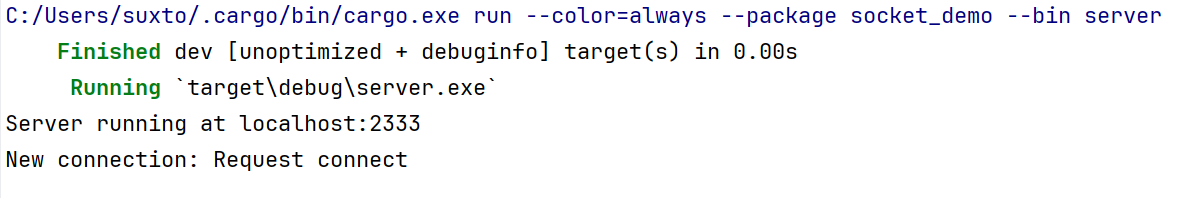
1. 实验内容
   1. 编写服务端代码
   2. 编写客户端代码
   3. 结果演示
2. 实验步骤
   1. 编写服务端代码：Rust代码：server.rs

use std::io::{Read, stdin, Write};  
use std::net::{TcpListener, TcpStream};  
use std::thread;  
  
const *ADDR*: &str = "localhost";  
const *PORT*: &str = "2333";  
  
fn main() {  
 let listener = TcpListener::*bind*(format!("{ADDR}:{PORT}")).expect("Failed to bind address");  
 println!("Server running at {}:{}", *ADDR*, *PORT*);  
  
 for stream in listener.incoming() {  
 let stream = stream.expect("Failed to establish connection");  
 thread::spawn(move || {  
 handle\_connection(stream);  
 });  
 }  
}  
  
fn handle\_connection(mut stream: TcpStream) {  
 let mut buffer = [0; 1024];  
 stream  
 .read(&mut buffer)  
 .expect("Failed to read data from stream");  
  
 *// Parse HTTP request headers* let request = String::*from\_utf8\_lossy*(&buffer[..]);  
 println!("New connection: {request}");  
 let response = "Server connected";  
 stream  
 .write(response.as\_bytes())  
 .expect("Failed to write response to stream");  
 loop {  
 let mut message = [0; 1024];  
 let result = stream.read(&mut message);  
 match result {  
 *Ok*(0) | *Err*(\_) => break,  
 *Ok*(n) => {  
 *// Process received WebSocket message* let message = &message[..n];  
 println!("Received message: {}", String::*from\_utf8\_lossy*(message));  
  
 println!("Type any thing need to sand");  
 let mut input = String::*new*();  
 stdin().read\_line(&mut input).expect("Unable to read from stdio");  
 *// Echo the message back to the client* stream  
 .write(input.as\_bytes())  
 .expect("Failed to write message to stream");  
 }  
 }  
 }  
}

* 1. 编写客户端代码：Rust代码：client.rs

use std::net::TcpStream;  
use std::io::{Read, stdin, Write};  
  
const *ADDR*: &str = "localhost";  
const *PORT*: &str = "2333";  
  
fn main() {  
 let mut stream = TcpStream::*connect*(format!("{ADDR}:{PORT}")).expect("Failed to connect to server");  
  
 let request = "Request connect";  
 stream.write(request.as\_bytes()).expect("Failed to send request to server");  
  
 let mut buffer = [0; 1024];  
 stream.read(&mut buffer).expect("Failed to read response from server");  
  
 *// Parse HTTP response* let response = String::*from\_utf8\_lossy*(&buffer[..]);  
 println!("{response}");  
 loop {  
 let mut input = String::*new*();  
 stdin().read\_line(&mut input).expect("Unable to read from stdio");  
  
 stream.write(input.as\_bytes()).expect("Failed to send message to server");  
  
 let mut response = [0; 1024];  
 let result = stream.read(&mut response);  
 match result {  
 *Ok*(0) | *Err*(\_) => break,  
 *Ok*(n) => {  
 *// Process received WebSocket message* let message = &response[..n];  
 println!("Received message: {}", String::*from\_utf8\_lossy*(message));  
 }  
 }  
 }  
}

* 1. 测试

1. 实验结果
   1. 启动服务器
      1. 
   2. 启动客户端，客户端自动连接服务器，服务器接收到连接请求。
      1. 
      2. 
   3. 消息互传测试
      1. 从客户端发送信息



* + 1. 从服务端发送信息



* + 1. 成功实现客户端和服务端使用web socket沟通

1. 实验总结

这次实验乃是为探究Socket编程之奥秘，曾花费心思学习Socket API，然后将所学知识付诸实践。光阴似箭，不觉中，我已倾注心血，谱写了服务端与客户端的交互之曲。

经过实验，不胜欣喜地宣告，目标已达成。初，撰写了服务端之码（server.rs），它居心地监听特定地址与端口，迎来客户端之连接请求。娴熟地接纳客户端所投递的数据，然后准确地回应客户端之需求，这是服务端之初衷。

继而，编写了客户端之代码（client.rs）。它从容地与服务端握手，秉承礼仪地传递请求，并默默地接收服务端的回应。这客户端，能自由地发送与接收消息，确保双向通信的顺畅进行。

然后，经过了严谨的测试，客户端与服务器之间的通信得以验证。客户端毫不费力地接入服务器，坚韧地发送与接收消息，实现了实验之旨意。

总结之际，这次实验令我感悟颇多。Socket编程的奥妙已渐显于我，TCP通信原理已得以领略，Rust编程语言则成为我驾驭网络通信之得力工具。此实验之所蕴涵，不仅是技术上之成就，更是对网络通信重要性之深刻认知，以及将Socket编程运用于各类网络应用之能力的锤炼。