МИНЕСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧЕРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«Брестский государственный технический университет»

Кафедра «Интеллектуальные информационные технологии»

Лабораторная работа №1

По дисциплине «Аппаратное и программное обеспечение сетей»

Тема: «Организация TCP – сервера и TCP – клиента»

Выполнил:

студент 3 курса

группы ИИ-21(2)

Кирилович А. А.

Проверил:

Степанчук В.И.

Брест 2023

Цель работы: 1) изучить основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрести навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов TCP.

Вариант 5.

|  |  |  |
| --- | --- | --- |
| Вариант задания для бригады студентов | Номер задания для реализации TCP (UDP) сервера (см. табл. 2). | Номер задания для реализации TCP (UDP) клиента (см. табл. 3). |
| 5 | 5 | 4 |

|  |  |
| --- | --- |
| Номер задания | Функция, выполняемая TCP-сервером |
| 5 | После приема каждой цепочки символов, заканчивающейся символом «.», сервер отсылает предложение, в котором слова (цепочки символов, отделенные пробелом) расположены в обратном порядке. Если встречается последовательно два символа «..», то сервер отсылает сообщение об окончании сеанса и разрывает соединение. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Вариант | Ввод символов с отсылкой введенной строки по нажатию на клавишу | Ведение файла протокола событий, включающих\*:  1) время начала и окончания соединения;  2) передаваемую серверу строку и время передачи строки;  3) принимаемую от сервера строку и время приема строки. | Возможность разрыва соединения при помощи команды:  **disconnect <адрес> <порт>** | 1) Задание в программе клиента специальной команды и параметров: подключения к серверу:  **сonnect**  **<адрес> <порт>**  2) Автоматическое подключение к серверу с заданным по умолчанию адресом при запуске клиента |
| 4 | PgDn | 1), 2), 3) | - | 2) |

Реализация TCP-сервера:

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

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

use std::thread;

use std::str;

fn handle\_client(mut stream: TcpStream) {

let mut buffer = [0; 1024];

while let Ok(bytes\_read) = stream.read(&mut buffer) {

if bytes\_read == 0 {

break;

}

let input\_str = str::from\_utf8(&buffer[..bytes\_read]).unwrap();

if input\_str.contains("..") {

stream.write\_all(b"Breaking connection...\n").unwrap();

stream.shutdown(std::net::Shutdown::Both).unwrap();

break;

}

else if input\_str.contains(".") {

let words: Vec<&str> = input\_str.split\_whitespace().collect();

let reversed\_words: Vec<&str> = words.iter().rev().copied().collect();

let reversed\_str = reversed\_words.join(" ");

stream.write\_all(reversed\_str.as\_bytes()).unwrap();

}

else {

stream.write\_all(input\_str.as\_bytes()).unwrap();

}

}

}

fn main() {

let listener = TcpListener::bind("127.0.0.1:6666").expect("Failed to bind");

println!("TCP SERVER DEMO - Listening on port 6666");

for stream in listener.incoming() {

match stream {

Ok(stream) => {

println!("New connection from: {}", stream.peer\_addr().unwrap());

thread::spawn(|| {

handle\_client(stream);

});

}

Err(e) => {

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

}

}

}

}

Реализация TCP-клиента:

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

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

use std::str;

use std::fs::{OpenOptions};

use std::time::SystemTime;

use chrono::{Local, Datelike, Timelike};

const DEFAULT\_FILENAME: &str = "log.txt";

fn write\_to\_file(content: &str) -> io::Result<()> {

let current\_time = SystemTime::now();

let local\_time: chrono::DateTime<Local> = current\_time.into();

let year = local\_time.year();

let month = local\_time.month();

let day = local\_time.day();

let hour = local\_time.hour();

let minute = local\_time.minute();

let second = local\_time.second();

let time\_str = format!("{}-{:02}-{:02} {:02}:{:02}:{:02}", year, month, day, hour, minute, second);

let line = format!("{}\t{}\n", time\_str, content);

let mut file = OpenOptions::new()

.create(true)

.append(true)

.open(DEFAULT\_FILENAME)?;

file.write\_all(line.as\_bytes())?;

Ok(())

}

fn main() -> io::Result<()> {

println!("TCP DEMO CLIENT");

let mut current\_server\_addr: Option<String> = None;

let mut current\_port: Option<u16> = None;

let mut stream: Option<TcpStream> = None;

'main\_loop: loop {

let stdin = io::stdin();

let mut reader = stdin.lock();

let mut buffer = String::new();

loop {

print!("Connect to server. Use 'connect <address> <port>' or 'quit' to exit. \n");

io::stdout().flush()?;

reader.read\_line(&mut buffer)?;

if buffer.trim() == "quit" {

println!("Exit...");

if let Some(s) = stream.take() {

s.shutdown(Shutdown::Both)?;

}

return Ok(());

} else if buffer.starts\_with("disconnect") {

println!("You are not connected.");

} else if buffer.starts\_with("connect") {

let parts: Vec<&str> = buffer.split\_whitespace().collect();

if parts.len() != 3 {

println!("Invalid command format. Use 'connect <address> <port>'.");

} else {

let new\_server\_addr = parts[1].to\_string();

let new\_port = parts[2].parse::<u16>();

match new\_port {

Ok(port) => {

current\_server\_addr = Some(new\_server\_addr.clone());

current\_port = Some(port);

println!("Connection to {}:{}...", new\_server\_addr, port);

if let Some(s) = stream.take() {

s.shutdown(Shutdown::Both)?;

}

match TcpStream::connect((new\_server\_addr.as\_str(), port)) {

Ok(s) => {

stream = Some(s);

let line = format!("CONNECTED to {}:{}",

new\_server\_addr,

port);

match write\_to\_file(line.as\_str()) {

Ok(\_) => {},

Err(e) => eprintln!("Error writing to file: {}", e),

}

}

Err(e) => {

eprintln!("Connection error: {}", e);

stream = None;

}

}

buffer.clear();

break;

}

Err(\_) => {

println!("Invalid port format. Use 'connect <address> <port>'.");

}

}

}

} else {

println!("You must first connect with 'connect <address> <port>'.");

}

buffer.clear();

}

if let Some(s) = stream.as\_mut() {

loop {

reader.read\_line(&mut buffer)?;

if buffer.trim() == "quit" {

println!("Exit...");

s.shutdown(Shutdown::Both)?;

break;

} else if buffer.starts\_with("disconnect") {

println!("Breaking connection...");

if let Some(s) = stream.take() {

s.shutdown(Shutdown::Both)?;

}

let line = format!("DISCONNECTED from {:#?}:{:#?}",

current\_server\_addr.unwrap\_or\_default(),

current\_port.unwrap\_or\_default());

match write\_to\_file(line.as\_str()) {

Ok(\_) => {},

Err(e) => eprintln!("Error writing to file: {}", e),

}

break 'main\_loop;

}

s.write\_all(buffer.as\_bytes())?;

buffer.clear();

let mut response\_buffer = [0; 1024];

match s.read(&mut response\_buffer) {

Ok(n) if n > 0 => {

let response = str::from\_utf8(&response\_buffer[..n]).unwrap();

println!("S=>C: {}", response);

let line = format!("S=>C: {}", response);

match write\_to\_file(line.as\_str()) {

Ok(\_) => {},

Err(e) => eprintln!("Error writing to file: {}", e),

}

}

Ok(\_) | Err(\_) => {

println!("Connection lost");

let line = format!("DISCONNECTED from {:#?}:{:#?}",

current\_server\_addr.unwrap\_or\_default(),

current\_port.unwrap\_or\_default());

match write\_to\_file(line.as\_str()) {

Ok(\_) => {},

Err(e) => eprintln!("Error writing to file: {}", e),

}

break 'main\_loop;

}

}

}

}

}

Ok(())

}

**Вывод:** 1) изучил основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрёл навыки по практическому использованию библиотеки для реализации сетевых приложений в среде С++ на базе протоколов TCP.