|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования  FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 3 | | |
| по дисциплине «Операционные системы и компьютерные сети» | | |
| Разработка приложения интерактивной переписки | | |
|  | | |
|  | Бригада | ВесЕлый Денис |
| №1 | Ворончук Илья |
|  |  |
| Группа | ПМИ-32 |
|  |  |
|  |  |
| Преподаватели | кобылянский в.г. |
|  | сивак м.а. |
| Новосибирск, 2025 | | |

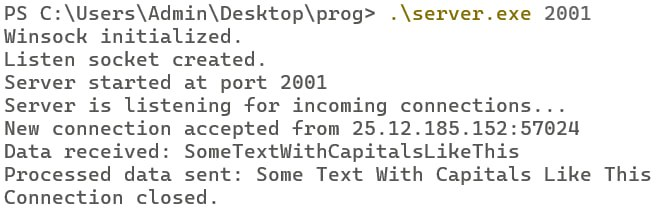
# Цель работы

Изучить основные принципы разработки многопользовательских приложений, построенных на основе технологии клиент-сервер с использованием стека протоколов TCP/IP.

# Описание хода выполнения работы

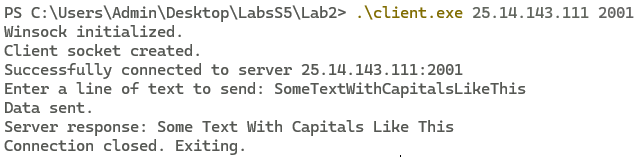
Для выполнения задания были разработаны два приложения: сервер и клиент. Серверная часть ожидает подключения, принимает строку текста, обрабатывает ее согласно варианту №1 и отправляет результат обратно.

Пример работы приложения со стороны сервера:



Запуск сервера происходил с указанием номера порта (2001) протокола TCP.

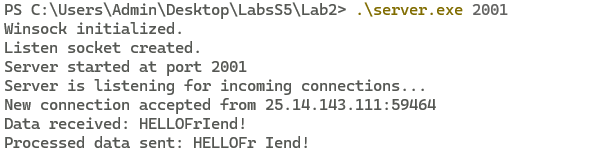
Пример работы приложения со стороны клиента:



Анализ результатов: клиент ввёл IP-адрес сервера (в данном случае 25.14.143.111) и порт (2001). А далее клиент успешно установил связь с сервером и отправил набор данных, введенный пользователем. Сервер получил строку текста «SomeTextWithCapitalsLikeThis» и, согласно заданию, вставил пробелы перед прописными буквами, которым предшествовали строчные. Обработанная строка «Some Text With Capitals Like This» была возвращена клиенту. Клиент, получив ответ с сервера, вывел ответ на экран и прекратил свою работу.

## Теперь рассмотрим пример со строкой «HELLOFrIend!», при этом устройства поменялись местами (видно по IP адресам).

Пример работы приложения со стороны сервера:



Со стороны клиента:



Анализ результата: как видим, строка была обработана верно: сервер получил строку текста «HELLOFrIend!» и, согласно заданию, вставил пробелы перед прописными буквами, которым предшествовали строчные. Обработанная строка «HELLOFr Iend!» была возвращена клиенту. А клиент, получив ответ от сервера, вывел результат на экран и прекратил свою работу.

# Вывод

В результате выполнения лабораторной работы были успешно освоены фундаментальные принципы построения клиент-серверных приложений.

На практике была реализована базовая схема сетевого взаимодействия, включающая следующие ключевые этапы:

1. Создание серверного приложения, способного принимать входящие TCP-соединения на заданном порту.
2. Разработка клиентского приложения, которое инициирует подключение к серверу и отправляет ему данные для обработки.
3. Организация двустороннего обмена данными с использованием низкоуровневых функций API сокетов (Winsock).

Эта работа позволила закрепить теоретические знания о работе сокетов и получить практические навыки, необходимые для разработки сетевого программного обеспечения.

# Приложение

### server.cpp

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

#include <cctype> // для isupper, islower

#include <iostream>

#include <string>

// При компиляции необходимо подключить библиотеку Ws2\_32.lib

#pragma comment(lib, "Ws2\_32.lib")

const int BUFFER\_SIZE = 1024; // Размер буфера

int main(int argc, char\* argv[]) {

// Проверка аргументов командной строки

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <port>" << std::endl;

std::cerr << "Example: " << argv[0] << " 2001" << std::endl;

return 1;

}

// Преобразование порта из строки в число

int port = atoi(argv[1]);

// Проверка диапазона порта

if (port <= 1024 || port > 65535) {

std::cerr << "Invalid port number. Please use a port in the range 1025-65535." << std::endl;

return 1;

}

// Инициализация Winsock

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed." << std::endl;

return 1;

}

std::cout << "Winsock initialized." << std::endl;

// Создание сокета

SOCKET listenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (listenSocket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed. Error: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

std::cout << "Listen socket created." << std::endl;

// Привязка сокета к IP-адресу и порту

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

serverAddr.sin\_addr.s\_addr = INADDR\_ANY; // Принимать подключения на любом IP-адресе

if (bind(listenSocket, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Bind failed. Error: " << WSAGetLastError() << std::endl;

closesocket(listenSocket);

WSACleanup();

return 1;

}

std::cout << "Server started at port " << serverAddr.sin\_port << std::endl;

// Перевод сокета в режим прослушивания

if (listen(listenSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cerr << "Listen failed. Error: " << WSAGetLastError() << std::endl;

closesocket(listenSocket);

WSACleanup();

return 1;

}

std::cout << "Server is listening for incoming connections..." << std::endl;

// Цикл приема клиентов

SOCKET clientSocket;

sockaddr\_in clientAddr;

int clientAddrSize = sizeof(clientAddr);

char buffer[BUFFER\_SIZE];

while (true) {

clientSocket = accept(listenSocket, (SOCKADDR\*)&clientAddr, &clientAddrSize);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Accept failed. Error: " << WSAGetLastError() << std::endl;

continue; // Продолжаем слушать других клиентов

}

std::cout << "New connection accepted from " << inet\_ntoa(clientAddr.sin\_addr)

<< ":" << ntohs(clientAddr.sin\_port) << std::endl;

// Получение данных от клиента

int bytesReceived = recv(clientSocket, buffer, BUFFER\_SIZE, 0);

if (bytesReceived > 0) {

buffer[bytesReceived] = '\0'; // Гарантируем, что строка заканчивается null-терминатором

std::cout << "Data received: " << buffer << std::endl;

// --- Основная логика ---

std::string inputStr(buffer);

std::string resultStr = "";

if (!inputStr.empty()) {

resultStr += inputStr[0]; // Первый символ добавляем как есть

for (size\_t i = 1; i < inputStr.length(); ++i) {

// Если текущий символ - эьл прописная буква, а предыдущий - строчная

if (isupper(static\_cast<unsigned char>(inputStr[i])) && islower(static\_cast<unsigned char>(inputStr[i - 1]))) {

resultStr += ' '; // Вставляем пробел

}

resultStr += inputStr[i];

}

}

// --- Конец блока ---

// Отправка обработанных данных обратно клиенту

send(clientSocket, resultStr.c\_str(), resultStr.length(), 0);

std::cout << "Processed data sent: " << resultStr << std::endl;

}

// Закрытие сокета клиента

closesocket(clientSocket);

std::cout << "Connection closed." << std::endl

<< std::endl;

}

// Очистка (в данном примере до этого кода не дойдет)

closesocket(listenSocket);

WSACleanup();

return 0;

}

### client.cpp

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

#include <iostream>

#include <string>

// При компиляции необходимо подключить библиотеку Ws2\_32.lib

#pragma comment(lib, "Ws2\_32.lib")

const int BUFFER\_SIZE = 1024; // Размер буфера

int main(int argc, char\* argv[]) {

// Проверка аргументов командной строки

if (argc != 3) {

std::cerr << "Usage: " << argv[0] << " <server\_ip> <port>" << std::endl;

std::cerr << "Example: " << argv[0] << " 127.0.0.1 2001" << std::endl;

return 1;

}

const char\* serverIp = argv[1];

int port = atoi(argv[2]);

// Проверка диапазона порта

if (port <= 1024 || port > 65535) {

std::cerr << "Invalid port number. Please use a port in the range 1025-65535." << std::endl;

return 1;

}

// Инициализация Winsock

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

std::cerr << "WSAStartup failed." << std::endl;

return 1;

}

std::cout << "Winsock initialized." << std::endl;

// Создание сокета

SOCKET clientSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Socket creation failed. Error: " << WSAGetLastError() << std::endl;

WSACleanup();

return 1;

}

std::cout << "Client socket created." << std::endl;

sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

serverAddr.sin\_addr.s\_addr = inet\_addr(serverIp);

// Подключение к серверу

if (connect(clientSocket, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Connection to server failed. Error: " << WSAGetLastError() << std::endl;

closesocket(clientSocket);

WSACleanup();

return 1;

}

std::cout << "Successfully connected to server " << serverIp << ":" << port << std::endl;

// Отправка данных

std::string message;

std::cout << "Enter a line of text to send: ";

std::getline(std::cin, message);

if (send(clientSocket, message.c\_str(), message.length(), 0) == SOCKET\_ERROR) {

std::cerr << "Send failed. Error: " << WSAGetLastError() << std::endl;

} else {

std::cout << "Data sent." << std::endl;

// Получение ответа от сервера

char buffer[BUFFER\_SIZE];

int bytesReceived = recv(clientSocket, buffer, BUFFER\_SIZE, 0);

if (bytesReceived > 0) {

buffer[bytesReceived] = '\0';

std::cout << "Server response: " << buffer << std::endl;

} else {

std::cerr << "Receive failed or connection closed. Error: " << WSAGetLastError() << std::endl;

}

}

// Закрытие сокета и очистка

closesocket(clientSocket);

WSACleanup();

std::cout << "Connection closed. Exiting." << std::endl;

system("pause"); // Чтобы консольное окно не закрылось сразу

return 0;

}