**Проектная работа**

**Клиент-сервер**

**Работу выполнили:**

Теплякова Елена (24кнт5)

Щеглакова Дарья (24кнт5)

**Введение:**

Данный проект представляет собой простую клиент-серверную систему для обмена зашифрованными сообщениями между двумя пользователями. Сервер, написанный на C++ с использованием Windows Sockets (Winsock), выполняет роль посредника, пересылая зашифрованные сообщения между клиентами без их расшифровки.

Принцип работы:

1. Сервер запускается и ожидает подключения двух клиентов.
2. Клиенты подключаются, вводят свой ключ шифрования и начинают взаимодействие.
3. Первый клиент отправляет сообщение, оно шифруется и передается серверу.
4. Сервер пересылает сообщение второму клиенту, не расшифровывая его.
5. Второй клиент расшифровывает сообщение, читает его и отправляет ответ.

**Подробный разбор и описание работы сервера и клиентов**

* **Подробное описание работы сервера (server.cpp)**

Этот серверный код написан на C++ с использованием Windows Sockets API (Winsock). Он принимает два клиента, после чего пересылает сообщения между ними. Сам сервер не расшифровывает сообщения, а просто передает данные от одного клиента к другому.

**1. Подключение необходимых библиотек**

* **#include <iostream>** – для работы с вводом/выводом (std::cout, std::cerr).
* **#include <winsock2.h>** – библиотека для работы с сетевыми сокетами в Windows.
* **#include <ws2tcpip.h>** – дополнительные функции работы с сетевыми адресами.
* **#include <vector>** – используется для хранения сокетов клиентов.

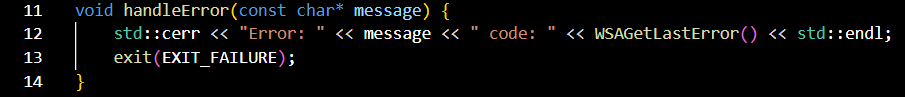
**2. Подключение библиотеки Winsock**

* Директива **#pragma comment(lib, "ws2\_32.lib**") подключает библиотеку Winsock

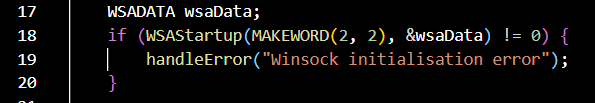
**3. Определение констант**

* **PORT "8080"** – сервер будет работать на порту 8080.
* **BUFFER\_SIZE 1024** – максимальный размер буфера для сообщений (1 КБ).

**4. Функция обработки ошибок**

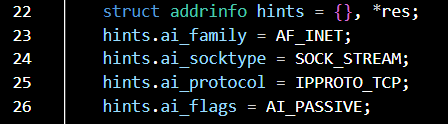
****Как работает эта функция?

1. Выводит сообщение об ошибке (**message**).
2. Получает код ошибки с помощью **WSAGetLastError().**
3. Завершает выполнение программы (**exit(EXIT\_FAILURE**)).
4. **Инициализация библиотеки Winsock**

****

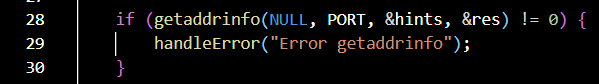
* **WSAStartup(MAKEWORD(2, 2), &wsaData)** – инициализирует Winsock.
* **MAKEWORD(2,2)** – указывает, что мы используем версию 2.2 Winsock.
* Если произошла ошибка, вызываем **handleError().**

1. Настройка информации о сервере



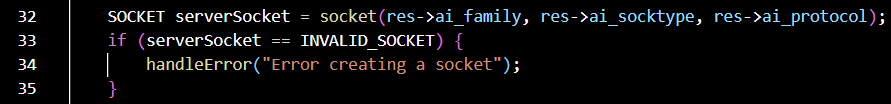
Разбор параметров:

* **hints.ai\_family = AF\_INET**; – используем IPv4.
* **hints.ai\_socktype = SOCK\_STREAM**; – используем TCP (ориентированный на соединение).
* **hints.ai\_protocol = IPPROTO\_TCP**; – указываем, что работаем с TCP.
* **hints.ai\_flags = AI\_PASSIVE**; – сервер принимает подключения на всех доступных сетевых интерфейсах.



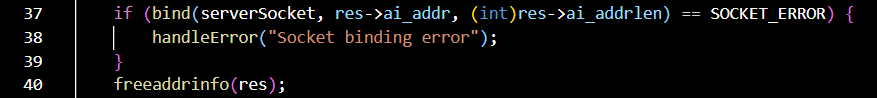
* **getaddrinfo()** получает информацию о сервере.
* Если произошла ошибка, вызываем **handleError().**

**7. Создание серверного сокета**



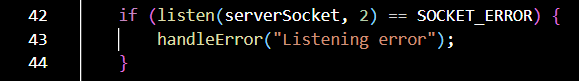
* Создаем сокет с параметрами из **addrinfo**.
* Если произошла ошибка, вызываем **handleError()**.

**8. Привязка сокета к порту**



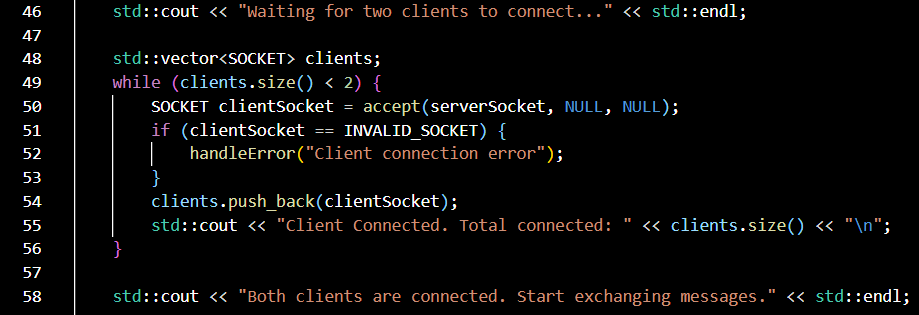
* Привязываем сокет к порту 8080 и освобождаем **addrinfo**.

**9. Ожидание входящих соединений**

****

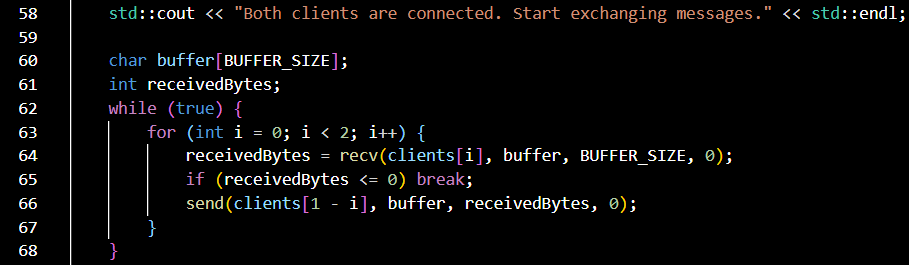
* Сервер начинает слушать входящие соединения.
* 2 – максимальное число ожидаемых клиентов.

**10. Ожидание подключения двух клиентов**

****

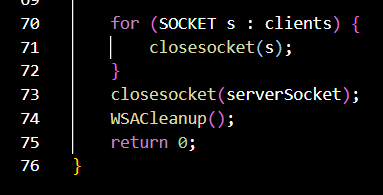
1. Выводим сообщение "**Waiting for two clients to connect...**".
2. Ждем подключения двух клиентов в цикле.
3. Принимаем соединение (**accept()**).
4. Добавляем клиента в вектор **clients**.
5. Выводим количество подключенных клиентов.

**11. Обмен сообщениями между клиентами**



1. Ждем сообщения от одного из клиентов (**recv()**).
2. Пересылаем это сообщение другому клиенту (**send()**).
3. Повторяем процесс.

**12. Завершение работы**

****

* Закрываем все клиентские соединения.
* Закрываем серверный сокет.
* Очищаем ресурсы **Winsock (WSACleanup()).**
* **Подробное описание работы клиентов (client1.cpp, client2.cpp)**

client1.cpp реализует простого клиента для чат-приложения, использующего TCP-сокеты и элементарный XOR-шифр для шифрования. Разберем его поэтапно:

**1. Подключение библиотек и компоновка:**

* **#include <iostream>** - директива, которая подключает библиотеку iostream для операций ввода/вывода (например, вывод в консоль).
* **#include <winsock2.h>** и **#include <ws2tcpip.h>** - директивы, которые подключают заголовочные файлы **Windows Sockets API**, необходимые для сетевого программирования в Windows.
* **#include <string>** - директива, которая подключает библиотеку string для использования типа данных std::string.
* **#pragma comment(lib, "ws2\_32.lib")** – директива, которая указывает компоновщику подключить необходимую библиотеку Windows Sockets (ws2\_32.lib). Без этого код не сможет использовать функции Winsock.

**2. Макроопределения:**

* **#define SERVER\_IP "127.0.0.1"** – директива, которая определяет константу, представляющую IP-адрес сервера. "127.0.0.1" — это адрес loopback, означающий, что сервер работает на той же машине, что и клиент.
* **#define PORT "8080"** – директива, которая определяет номер порта, на котором сервер прослушивает соединения. Он должен совпадать с портом, используемым сервером.
* **#define BUFFER\_SIZE 1024** – директива, которая определяет размер буфера, используемого для отправки и получения данных.

**3. XOR-шифрование/дешифрование:**

* **std::string xorEncryptDecrypt(const std::string& message, char key)** – функция, которая реализует простой XOR-шифр. Каждый символ в message подвергается побитовой операции XOR с символом key. Повторное применение XOR с тем же ключом расшифрует сообщение.

**4. Обработка ошибок:**

* **void handleError(const char message)** – функция, которая обрабатывает ошибки, возникающие во время операций Winsock. Она выводит сообщение об ошибке в консоль, включая код ошибки из **WSAGetLastError(),** а затем завершает программу.

**5. Основная функция (main)**

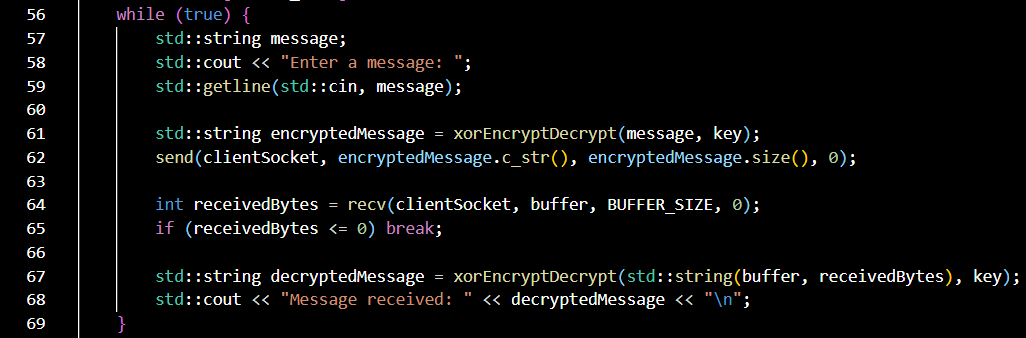
* **WSAStartup** – функция, которая инициализирует библиотеку Winsock. Это необходимо перед использованием любых других функций Winsock.
* **Getaddrinfo** – функция, которая преобразует IP-адрес и номер порта сервера в структуру сокета, которую можно использовать. Это более современный и надежный подход, чем более старые функции, такие как inet\_addr.
* **Socket** – абстракция, которая создает новый сокет. Параметры указывают семейство адресов (IPv4), тип сокета (ориентированный на поток, TCP) и протокол (TCP).
* **Connect** – функция, которая пытается установить соединение с сервером по указанному адресу и порту.
* Ввод ключа: программа запрашивает у пользователя односимвольный ключ шифрования. Этот ключ используется как для шифрования, так и для дешифрования.
* Основной цикл while непрерывно отправляет и принимает сообщения:
* Получает сообщение от пользователя с помощью **std::getline**.
* Шифрует сообщение с помощью функции **xorEncryptDecrypt**.
* Отправляет зашифрованное сообщение на сервер с помощью функции **send**.
* Получает сообщение от сервера с помощью функции **recv**.
* Расшифровывает полученное сообщение с помощью **xorEncryptDecrypt.**
* Выводит расшифрованное сообщение в консоль.
* **Closesocket** – функция, которая закрывает сокет.
* **WSACleanup** – функция, которая очищает библиотеку Winsock.

**client2.cpp:**

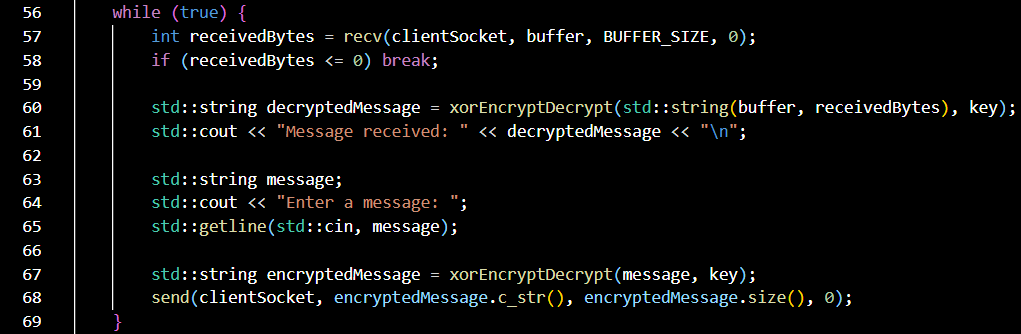
Client2 и client1 заключается в порядке отправки и получения сообщений в основном цикле. Клиент 2 получает сообщение от сервера перед отправкой своего собственного сообщения.

Клиент 1 отправляет сообщение, а затем ожидает ответа. Клиент 2 ожидает сообщение от сервера, прежде чем отправлять ответ.

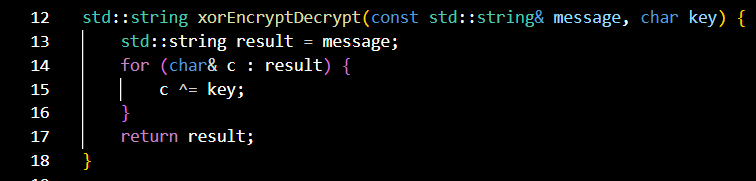
Как это выглядит для client1:



client2:

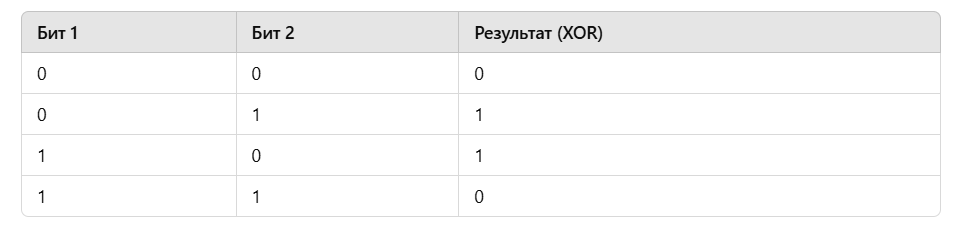


**Используемый шифр:**

****

**XOR (Исключающее ИЛИ) –** это простая побитовая операция, которая работает следующим образом:

* Если два бита разные, результат 1.
* Если два бита одинаковые, результат 0.



XOR-шифрование работает на принципе **обратимости**:

Если применить XOR дважды с одним и тем же ключом, исходный текст восстановится.

Пример:

1. **Шифрование:** A (65) ⊕ key (3) = 66 (B)
2. **Дешифрование:** B (66) ⊕ key (3) = 65 (A)

**2. Как шифрование работает в программе?**

**Шаг 1: Пользователь вводит сообщение**



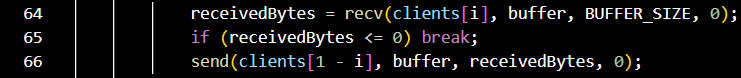
* Клиент запрашивает у пользователя текст для отправки.

**Шаг 2: Шифрование перед отправкой**

****

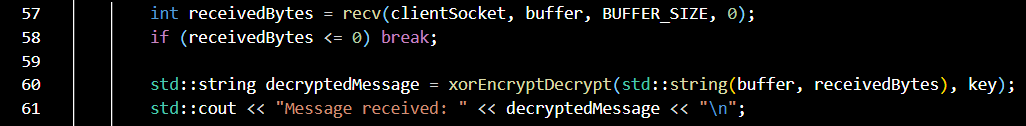
* Введённое сообщение проходит через функцию **xorEncryptDecrypt()**.
* Каждый символ XOR-ится с ключом.
* Зашифрованное сообщение отправляется на сервер.

**Шаг 3: Сервер передаёт сообщение второму клиенту**



* Сервер не расшифровывает сообщение, а просто передаёт его второму клиенту.

**Шаг 4: Второй клиент получает сообщение и расшифровывает**



* Полученное сообщение снова проходит через **xorEncryptDecrypt()**.
* Так как XOR обратим, оно дешифруется и выводится на экран.