**БУ ВО Ханты-Мансийского автономного округа – Югры**

**«Сургутский государственный университет»**

**Политехнический институт**

**Кафедра автоматики и компьютерных систем**

**ОТЧЁТ**

**по лабораторной работе №5**

**по дисциплине: «Операционные системы»**

**Выполнил: студент группы №609-21**

**Шумилов И. Д.**

**Принял: старший преподаватель кафедры АиКС**

**Кривицкая М. А.**

**Сургут**

**2024г.**

**Цель работы:** изучить основные принципы организации удаленного взаимодействия процессов с помощью механизма сокетов и соответствующие функции и структуры данных, предоставляемые интерфейсом библиотеки WinSock, закрепить навыки практического использования прикладного программного интерфейса.

**Задание на лабораторную работу:**

Разработать приложения представляющие серверную и клиентскую стороны, которые реализуют (варианты по выбору):

1. игру «Морской бой»

2. игру «Крестики-нолики»

3. программу передачи текстовых сообщений по сети

**Индивидуальное задание на лабораторную работу:**

Вариант 3. Разработать программу передачи текстовых сообщений по сети.

**Ход работы:**

Были спроектированы серверная и клиентская стороны программы (лист. 1-2).

*Листинг 1. Код программы серверной стороны:*

*#include <iostream>*

*#include <WinSock2.h>*

*#include <WS2tcpip.h>*

*#include <stdio.h>*

*#include <vector>*

*#include <string>*

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

*#define BUFF\_SIZE 1024*

*void processError(std::string message, int errorCode) {*

*std::cout << message << ". Error " << errorCode << std::endl;*

*}*

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

*const char SERV\_IP[] = "127.0.0.1"; // localhost*

*const int PORT\_NUM = 8000;*

*int iResult;*

*WSADATA wsData;*

*SOCKET serverSocket = INVALID\_SOCKET;*

*iResult = WSAStartup(MAKEWORD(2, 2), &wsData);*

*if (iResult != NO\_ERROR) {*

*processError("Couldn't initialize WinSock", WSAGetLastError());*

*return 1;*

*}*

*else*

*std::cout << "WinSock initialization completed" << std::endl;*

*serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);*

*if (serverSocket == INVALID\_SOCKET) {*

*processError("Couldn't initialize socket", WSAGetLastError());*

*WSACleanup();*

*return 1;*

*}*

*else*

*std::cout << "Server socket initialization complete" << std::endl;*

*in\_addr ip\_addr;*

*iResult = inet\_pton(AF\_INET, SERV\_IP, &ip\_addr);*

*if (iResult <= 0) {*

*processError("Couldn't translate IP to numeric format", 1);*

*return 1;*

*}*

*sockaddr\_in serverInfo;*

*ZeroMemory(&serverInfo, sizeof(serverInfo));*

*serverInfo.sin\_family = AF\_INET;*

*serverInfo.sin\_addr = ip\_addr;*

*serverInfo.sin\_port = htons(PORT\_NUM);*

*iResult = bind(serverSocket, (sockaddr\*)&serverInfo, sizeof(serverInfo));*

*if (iResult == SOCKET\_ERROR) {*

*processError("Couldn't bind socket", WSAGetLastError());*

*closesocket(serverSocket);*

*WSACleanup();*

*return 1;*

*}*

*else*

*std::cout << "Socket binding completed" << std::endl;*

*iResult = listen(serverSocket, SOMAXCONN);*

*if (iResult != NO\_ERROR) {*

*processError("Couldn't start listener", WSAGetLastError());*

*closesocket(serverSocket);*

*WSACleanup();*

*return 1;*

*}*

*else*

*std::cout << "Listening..." << std::endl;*

*sockaddr\_in clientInfo;*

*ZeroMemory(&clientInfo, sizeof(clientInfo));*

*int clientInfo\_sz = sizeof(clientInfo);*

*SOCKET clientConn = accept(serverSocket, (sockaddr\*)&clientInfo, &clientInfo\_sz);*

*if (clientConn == INVALID\_SOCKET) {*

*processError("Couldn't connect to client", WSAGetLastError());*

*closesocket(serverSocket);*

*WSACleanup();*

*return 1;*

*}*

*else {*

*std::cout << "Client connection established successfully" << std::endl;*

*char clientIP[22];*

*inet\_ntop(AF\_INET, &clientInfo.sin\_addr, clientIP, INET\_ADDRSTRLEN);*

*std::cout << "Client IP: " << clientIP << std::endl;*

*}*

*std::vector<char> servBuff(BUFF\_SIZE), clientBuff(BUFF\_SIZE);*

*short packetSize = 0;*

*while (true) {*

*packetSize = recv(clientConn, servBuff.data(), servBuff.size(), 0);*

*std::cout << "Client message: " << servBuff.data();*

*std::cout << "Enter host message: ";*

*std::fgets(clientBuff.data(), BUFF\_SIZE, stdin);*

*if (clientBuff[0] == 'x' && clientBuff[1] == 'x' && clientBuff[2] == 'x') {*

*shutdown(clientConn, SD\_BOTH);*

*closesocket(serverSocket);*

*closesocket(clientConn);*

*WSACleanup();*

*return 0;*

*}*

*packetSize = send(clientConn, clientBuff.data(), clientBuff.size(), 0);*

*if (packetSize == SOCKET\_ERROR) {*

*processError("Couldn't send message to client", WSAGetLastError());*

*closesocket(serverSocket);*

*closesocket(clientConn);*

*WSACleanup();*

*return 1;*

*}*

*}*

*closesocket(serverSocket);*

*closesocket(clientConn);*

*WSACleanup();*

*return 0;*

*}*

*Листинг 2. Код программы клиентской стороны:*

*#include <iostream>*

*#include <WinSock2.h>*

*#include <WS2tcpip.h>*

*#include <stdio.h>*

*#include <vector>*

*#include <string>*

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

*#define BUFF\_SIZE 1024*

*void processError(std::string message, int errorCode) {*

*std::cout << message << ". Error " << errorCode << std::endl;*

*}*

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

*const char SERV\_IP[] = "127.0.0.1"; // localhost*

*const int PORT\_NUM = 8000;*

*int iResult;*

*WSADATA wsData;*

*SOCKET clientSocket = INVALID\_SOCKET;*

*iResult = WSAStartup(MAKEWORD(2, 2), &wsData);*

*if (iResult != NO\_ERROR) {*

*processError("Couldn't initialize WinSock", WSAGetLastError());*

*return 1;*

*}*

*else*

*std::cout << "WinSock initialization completed" << std::endl;*

*clientSocket = socket(AF\_INET, SOCK\_STREAM, 0);*

*if (clientSocket == INVALID\_SOCKET) {*

*processError("Couldn't initialize socket", WSAGetLastError());*

*WSACleanup();*

*return 1;*

*}*

*else*

*std::cout << "Client socket initialization complete" << std::endl;*

*in\_addr ip\_addr;*

*iResult = inet\_pton(AF\_INET, SERV\_IP, &ip\_addr);*

*if (iResult <= 0) {*

*processError("Couldn't translate IP to numeric format", 1);*

*return 1;*

*}*

*sockaddr\_in servInfo;*

*ZeroMemory(&servInfo, sizeof(servInfo));*

*servInfo.sin\_family = AF\_INET;*

*servInfo.sin\_addr = ip\_addr;*

*servInfo.sin\_port = htons(PORT\_NUM);*

*iResult = connect(clientSocket, (sockaddr\*)&servInfo, sizeof(sockaddr));*

*if (iResult != NO\_ERROR) {*

*processError("Couldn't connect to server", WSAGetLastError());*

*closesocket(clientSocket);*

*WSACleanup();*

*return 1;*

*}*

*else {*

*std::cout << "Server connection established successfully" << std::endl;*

*char servIP[22];*

*inet\_ntop(AF\_INET, &servInfo.sin\_addr, servIP, INET\_ADDRSTRLEN);*

*std::cout << "Server IP: " << servIP << std::endl;*

*}*

*std::vector<char> servBuff(BUFF\_SIZE), clientBuff(BUFF\_SIZE);*

*short packetSize = 0;*

*while (true) {*

*std::cout << "Enter your message: ";*

*std::fgets(clientBuff.data(), BUFF\_SIZE, stdin);*

*if (clientBuff[0] == 'x' && clientBuff[1] == 'x' && clientBuff[2] == 'x') {*

*shutdown(clientSocket, SD\_BOTH);*

*closesocket(clientSocket);*

*WSACleanup();*

*return 0;*

*}*

*packetSize = send(clientSocket, clientBuff.data(), clientBuff.size(), 0);*

*if (packetSize == SOCKET\_ERROR) {*

*processError("Couldn't send message to server", WSAGetLastError());*

*closesocket(clientSocket);*

*WSACleanup();*

*return 1;*

*}*

*packetSize = recv(clientSocket, servBuff.data(), servBuff.size(), 0);*

*if (packetSize == SOCKET\_ERROR) {*

*processError("Couldn't receive message from server", WSAGetLastError());*

*closesocket(clientSocket);*

*WSACleanup();*

*return 1;*

*}*

*else*

*std::cout << "Server message: " << servBuff.data();*

*}*

*closesocket(clientSocket);*

*WSACleanup();*

*return 0;*

*}*

**Пояснения к программе**

Обе программы начинаются с объявления констант и переменных SOCKET и WSADATA, необходимых для корректной установки соединения между двумя сторонами и инициализации необходимых библиотек. Была выбрана вторая версия сокетов при инициализации библиотек и протокол TCP при создании сокетов. Далее обеих программах происходит заполнение структуры sockaddr\_in, необходимой либо для запроса соединения с сервером по адресу из структуры sockaddr\_in, в случае клиентской стороны, либо для привязки сокета к адресу из структуры sockaddr\_in, в случае серверной стороны. После чего серверной стороне необходимо начать прослушивание соединения на наличие запрашивающих соединение клиентов, в случае появления таковых, с помощью функции accept сервер производит прием соединения и записывает сокет клиента. Запрос соединения необходим, т. к. сокеты используют защищенный протокол TCP, который выполняя данные операции обеспечивает большую надежность, чем подобный ему протокол UDP. На этом этап установления соединения между двумя сторонами окончен.

На следующем этапе обе программы в бесконечном цикле обмениваются сообщениями следующим образом:

* Клиентская сторона запрашивает пользовательский ввод с клавиатуры, после чего делает проверку на прерывание соединения (если было передано сообщение «xxx»), в случае если соединение не было прервано, то происходит отправка сообщения и клиент переходит в состояние ожидания ответного сообщения от сервера (хоста). После получения сообщения от сервера начинается новая итерация цикла.
* Серверная сторона же в начале цикла ожидает сообщение от клиента, а после его получения запрашивает пользовательский ввод с такой же возможностью прервать соединение как и на клиентской стороне. В случае если соединение не прерывается, сообщение отправляется клиенту и начинается новая итерация цикла.

Обмен сообщениями просходит с помощью функций send и recv, все возможные ошибки обрабатываются программами корректно и завершают свою работу предварительно закрыв открытые сокеты и выгрузив библиотеки WinSock.

**Выводы:** были изучены основные принципы организации удаленного взаимодействия процессов с помощью механизма сокетов и соответствующие функции и структуры данных, предоставляемые интерфейсом библиотеки WinSock, закреплены навыки практического использования прикладного программного интерфейса.