УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе № 2

по предмету «Компьютерные системы и сети»

Вариант 2

Выполнил:

Горник А.

гр. 351003

Проверил:

Леванцевич В.А.

Минск, 2025

Вариант №2

Разработать две клиент - серверные программы. Клиент отсылает серверу введенный пользователем номер числа Фибоначчи. Сервер принимает номер, вычисляет число Фибоначчи с этим номером, по формуле *Fi* = *Fi*–1 + *Fi*–2, *F*0 = *F*1 = 1 и выводит полученные значения на экран и отсылает их клиенту. Клиент так же выводит их на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

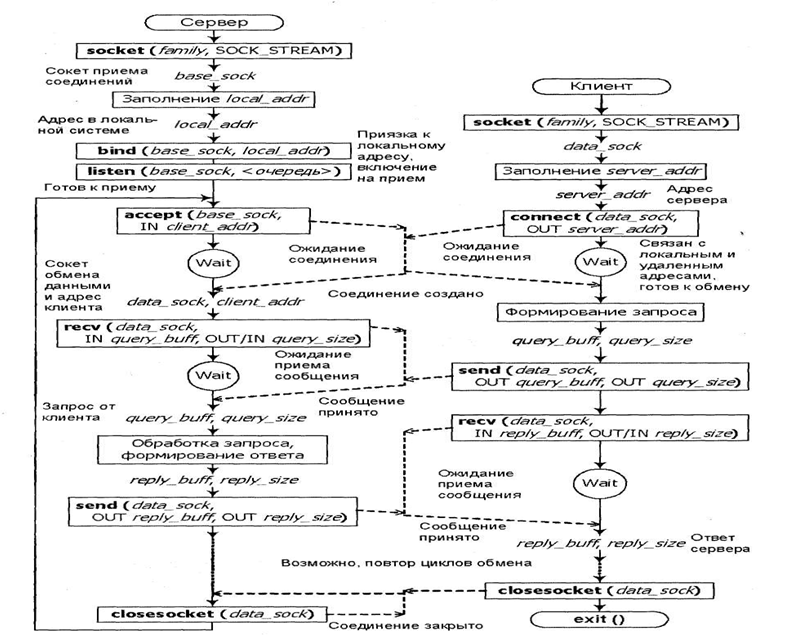


Рис. Взаимодействие с установлением соединения (TCP)

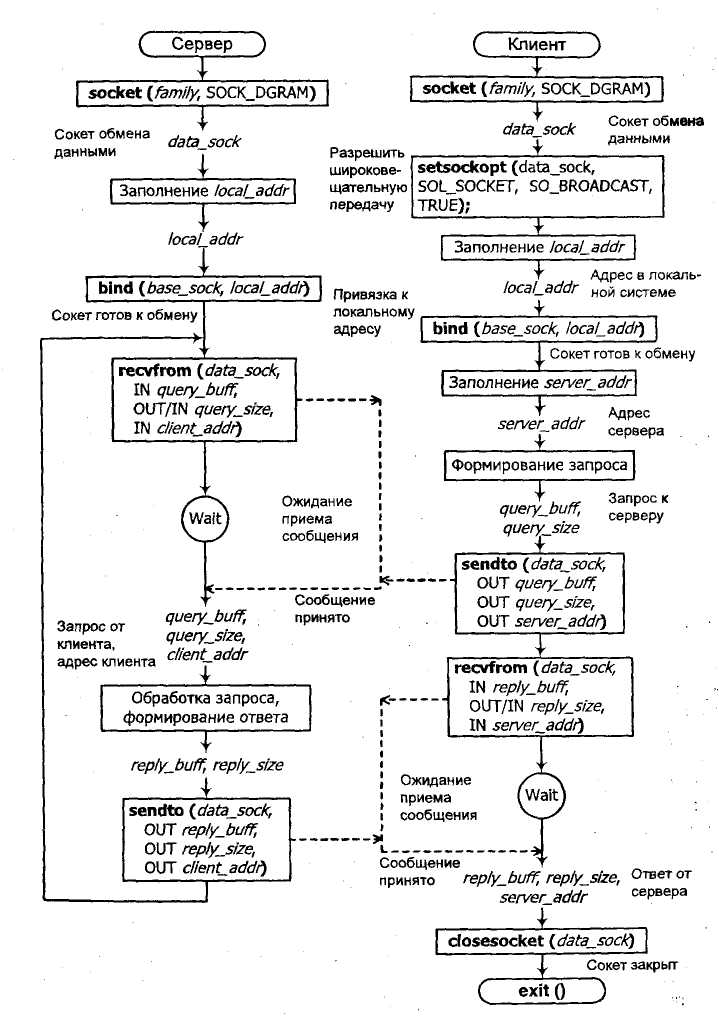


Рис. Взаимодействие без установления соединения (UDP)

**Код UDP client**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#pragma comment(lib, "ws2\_32.lib")

#include <winsock2.h>

#include <ws2tcpip.h>

#include <locale.h>

#include <windows.h>

#include <stdio.h>

#include <winsock.h>

char DataBuffer[1024]; // Буфер для приема и отправки данных (размер 1024 байта)

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

{

// Устанавливаем кодировку консоли для корректного ввода русского текста

SetConsoleCP(1251);

// Устанавливаем кодировку консоли для корректного вывода русского текста

SetConsoleOutputCP(1251);

struct sockaddr\_in SockAddrLocal, SockAddrSend, SockAddrRecv; // Структуры для хранения адресов сокетов

SOCKET SockLocal = INVALID\_SOCKET; // Локальный сокет, пока что неинициализирован

struct hostent\* pHostEnt; // Структура для хранения информации о хосте

int nSockOptBC, nAddrSize, nPortRemote, nMsgLen, i; // Переменные для работы с сокетами и сообщениями

WSADATA WSAData; // Структура для хранения информации о версии WinSock

WORD wWSAVer; // Версия WinSock

// Проверяем количество аргументов командной строки

if (argc != 4) {

puts("Неверные аргументы\n");

puts("Вызов: UDP\_SEND <addr/name> <port> <F>\n");

return -1;

}

// Инициализация библиотеки сокетов WinSock

wWSAVer = MAKEWORD(1, 1); // Указываем версию 1.1

if (WSAStartup(wWSAVer, &WSAData) != 0) { // Запускаем WinSock

puts("Ошибка инициализации WinSockets");

return -1;

}

// Создание UDP-сокета

SockLocal = socket(PF\_INET, SOCK\_DGRAM, 0);

if (SockLocal == INVALID\_SOCKET) {

fputs("Ошибка создания сокета\n", stderr);

return -1;

}

// Разрешаем широковещательную передачу (Broadcast)

nSockOptBC = 1;

setsockopt(SockLocal, SOL\_SOCKET, SO\_BROADCAST, (char\*)(&nSockOptBC), sizeof(nSockOptBC));

// Привязка сокета к локальному адресу

memset(&SockAddrLocal, 0, sizeof(SockAddrLocal)); // Обнуляем структуру

SockAddrLocal.sin\_family = AF\_INET; // Указываем, что используем IPv4

SockAddrLocal.sin\_addr.S\_un.S\_addr = INADDR\_ANY; // Разрешаем использовать все сетевые интерфейсы

SockAddrLocal.sin\_port = 0; // Автоматически назначаем порт

if (bind(SockLocal, (struct sockaddr\*)&SockAddrLocal, sizeof(SockAddrLocal)) != 0) {

fputs("Ошибка привязки к локальному адресу\n", stderr);

return -1;

}

// Подготовка адреса сервера

memset(&SockAddrSend, 0, sizeof(SockAddrSend)); // Обнуляем структуру

SockAddrSend.sin\_family = AF\_INET; // Указываем, что используем IPv4

if (strcmp(argv[1], "255.255.255.255") == 0) { // Проверяем, является ли адрес широковещательным

SockAddrSend.sin\_addr.S\_un.S\_addr = INADDR\_BROADCAST;

}

else {

SockAddrSend.sin\_addr.S\_un.S\_addr = inet\_addr(argv[1]); // Преобразуем строку в IP-адрес

if (SockAddrSend.sin\_addr.S\_un.S\_addr == INADDR\_NONE) { // Если не удалось преобразовать

if ((pHostEnt = gethostbyname(argv[1])) == NULL) { // Пробуем получить IP-адрес по доменному имени

fprintf(stderr, "Хост не опознан: %s\n", argv[1]);

return -1;

}

SockAddrSend.sin\_addr = \*(struct in\_addr\*)(pHostEnt->h\_addr\_list[0]);

}

}

// Преобразуем аргумент с портом из строки в число

if (sscanf(argv[2], "%u", &nPortRemote) < 1) {

fprintf(stderr, "Ошибочный номер порта: %s\n", argv[2]);

return -1;

}

SockAddrSend.sin\_port = htons((unsigned short)nPortRemote); // Конвертируем порт в сетевой порядок байтов

// Основной цикл отправки сообщений

for (i = 3; i < argc; ++i) { // Обрабатываем оставшиеся аргументы командной строки (сообщения)

// Отправка сообщения

fprintf(stdout, "Отсылка на %s:%u: \"%s\" \n",

inet\_ntoa(SockAddrSend.sin\_addr), // Преобразует IP-адрес из двоичного формата в строку (например, "192.168.1.1")

ntohs(SockAddrSend.sin\_port), // Преобразует порт из сетевого порядка байтов в хостовый (чтобы отобразить корректный номер порта)

argv[i] // Выводит сообщение, которое будет отправлено

);

nMsgLen = strlen(argv[i]) + 1; // Определяем длину сообщения (+1 для null-терминатора)

if (sendto(SockLocal, argv[i], nMsgLen, 0, (struct sockaddr\*)&SockAddrSend, sizeof(SockAddrSend)) < nMsgLen) {

fprintf(stderr, "Ошибка отсылки: \"%s\"\n", argv[i]);

continue;

}

// Получение ответа

nAddrSize = sizeof(SockAddrRecv);

nMsgLen = recvfrom(SockLocal, DataBuffer, sizeof(DataBuffer) - 1, 0, (struct sockaddr\*)&SockAddrRecv, &nAddrSize);

if (nMsgLen <= 0) { // Ошибка приема

fputs("Ошибка приема ответа\n", stderr);

continue;

}

DataBuffer[nMsgLen] = '\0'; // Добавляем null-терминатор

fprintf(stdout, "Ответ от %s:%u: \"%s\" \n",

inet\_ntoa(SockAddrRecv.sin\_addr),

ntohs(SockAddrRecv.sin\_port),

DataBuffer

);

}

// Завершение работы

shutdown(SockLocal, 2); // Завершаем соединение

Sleep(100); // Небольшая задержка перед закрытием сокета

closesocket(SockLocal); // Закрываем сокет

SockLocal = INVALID\_SOCKET; // Помечаем, что сокет более не активен

WSACleanup(); // Завершаем работу с WinSock

return 0; // Завершаем программу

}

**Код UDP server**

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения безопасности, связанные с функциями стандартной библиотеки C

#pragma comment(lib, "ws2\_32.lib") // Подключает библиотеку Windows Sockets 2 (ws2\_32.lib) при компиляции

#include <winsock2.h> // Основная библиотека для работы с сокетами в Windows

#include <ws2tcpip.h> // Дополнительные функции для работы с IP-адресами (например, inet\_pton, getaddrinfo)

#include <locale.h> // Для установки локализации (например, для корректного отображения символов)

#include <windows.h> // Заголовочный файл для работы с Windows API

#include <stdio.h> // Стандартный ввод/вывод (printf, scanf и т. д.)

#include <winsock.h> // Устаревшая версия библиотеки для работы с сокетами (лучше использовать winsock2.h)

#define DEFAULT\_ECHO\_PORT 7 // Определение порта по умолчанию (7 — стандартный порт для echo-серверов)

char DataBuffer[1024]; // Буфер для приёма и передачи данных размером 1024 байта

// Функция вычисления n-го числа Фибоначчи

int fibo(int count) {

int a = 1;

int b = 1;

if (count == 0 || count == 1) { // Если n = 0 или n = 1, возвращаем 1

return 1;

}

for (int i = 2; i <= count; i++) { // Итеративный подсчёт числа Фибоначчи

int temp = a + b;

a = b;

b = temp;

}

return b; // Возвращаем n-е число Фибоначчи

}

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

// Устанавливаем кодировку консоли для корректного ввода и вывода русских символов

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

struct sockaddr\_in SockAddrLocal, SockAddrRemote; // Структуры для хранения адресов сервера и клиента

SOCKET SockLocal = INVALID\_SOCKET; // Локальный сокет, инициализируется как недействительный

unsigned short nPort = DEFAULT\_ECHO\_PORT; // Переменная для хранения номера порта, по умолчанию 7

int nAddrSize, nCnt; // Размер структуры адреса и переменная для хранения количества принятых байтов

WSADATA WSAData; // Структура для хранения информации о реализации Windows Sockets

WORD wWSAVer; // Версия Windows Sockets

// Разбор аргументов командной строки: получение номера порта

if (argc > 1)

if (sscanf(argv[1], "%u", &nPort) < 1) // Читаем порт из аргумента командной строки

fprintf(stderr, "Ошибочный порт: %s, use default", nPort); // Ошибка, если номер порта некорректен

// Инициализация библиотеки сокетов

wWSAVer = MAKEWORD(1, 1); // Версия 1.1

if (WSAStartup(wWSAVer, &WSAData) != 0) { // Запуск библиотеки сокетов

puts("Ошибка инициализации подсистемы WinSocket");

return -1;

}

// Создание UDP-сокета

SockLocal = socket(PF\_INET, SOCK\_DGRAM, 0); // Используется протокол UDP (SOCK\_DGRAM)

if (SockLocal == INVALID\_SOCKET) { // Проверяем, удалось ли создать сокет

fputs("Ошибка создания сокета\n", stderr);

return -1;

}

// Привязка сокета к локальному адресу

memset(&SockAddrLocal, 0, sizeof(SockAddrLocal)); // Обнуляем структуру адреса

SockAddrLocal.sin\_family = AF\_INET; // Указываем семейство адресов (IPv4)

SockAddrLocal.sin\_addr.S\_un.S\_addr = INADDR\_ANY; // Привязываем ко всем доступным сетевым интерфейсам

SockAddrLocal.sin\_port = htons(nPort); // Преобразуем номер порта в сетевой порядок байтов

if (bind(SockLocal, (struct sockaddr\*)&SockAddrLocal, sizeof(SockAddrLocal)) != 0) { // Привязываем сокет

fprintf(stdout, "Ошибка привязки сокета, порт %u\n", ntohs(SockAddrLocal.sin\_port));

return -1;

}

fprintf(stderr, "Сервер запущен, порт %u\n", ntohs(SockAddrLocal.sin\_port)); // Выводим сообщение о запуске сервера

// Основной рабочий цикл сервера

while (1) { // Сервер работает бесконечно

nAddrSize = sizeof(SockAddrRemote);

// Получение входящего сообщения от клиента

nCnt = recvfrom(SockLocal, DataBuffer, sizeof(DataBuffer) - 1, 0,

(struct sockaddr\*)&SockAddrRemote, &nAddrSize);

if (nCnt < 0) { // Проверяем, удалось ли получить данные

fputs("Ошибка приема сообщения\n", stderr);

continue;

}

// Вычисление числа Фибоначчи

int res = fibo(atoi(DataBuffer)); // Преобразуем строку в число и вычисляем число Фибоначчи

printf("\nНа сервере произошли вычисления числа Фибоначчи.\nРезультат = %d", res);

memset(DataBuffer, 0, sizeof(DataBuffer)); // Очищаем буфер перед отправкой ответа

snprintf(DataBuffer, sizeof(DataBuffer), "%d", res); // Преобразуем результат обратно в строку

// Отправляем ответ клиенту

sendto(SockLocal, DataBuffer, strlen(DataBuffer), 0,

(struct sockaddr\*)&SockAddrRemote, sizeof(SockAddrRemote));

}

// Завершение работы сервера (этот код никогда не выполнится, так как цикл бесконечный)

shutdown(SockLocal, 2); // Закрываем соединение (2 - запрет отправки и приема)

Sleep(100); // Даем время завершить соединение

closesocket(SockLocal); // Закрываем сокет

SockLocal = INVALID\_SOCKET; // Обнуляем переменную сокета

WSACleanup(); // Освобождаем ресурсы, связанные с Winsock

return 0; // Выход из программы

}

**Код TCP client**

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения безопасности, связанные с функциями стандартной библиотеки C

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS // Отключает предупреждения об устаревших функциях WinSock

#pragma comment(lib, "ws2\_32.lib") // Подключает библиотеку Windows Sockets 2 (ws2\_32.lib) при компиляции

#include <winsock2.h> // Основная библиотека для работы с сокетами в Windows

#include <ws2tcpip.h> // Дополнительные функции для работы с IP-адресами (например, getaddrinfo, inet\_pton)

#include <locale.h> // Для установки локализации (например, для корректного отображения символов)

#include <windows.h> // Заголовочный файл для работы с Windows API

#include <stdio.h> // Стандартный ввод/вывод (printf, scanf и т. д.)

#include <winsock.h> // Устаревшая версия библиотеки для работы с сокетами (лучше использовать winsock2.h)

char DataBuffer[1024]; // Буфер для приёма и передачи данных размером 1024 байта

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

{

// Устанавливаем кодировку консоли для корректного ввода русских символов

SetConsoleCP(1251);

// Устанавливаем кодировку консоли для корректного вывода русских символов

SetConsoleOutputCP(1251);

struct sockaddr\_in SockAddrServer; // Структура для хранения адреса сервера

SOCKET SockData = INVALID\_SOCKET; // Сокет для обмена данными, пока что неинициализированный

struct hostent\* pHostEnt; // Указатель на структуру с информацией о хосте

int nPortServer, nMsgLen, i; // Переменные: номер порта, длина сообщения, счетчик цикла

WSADATA WSAData; // Структура для хранения информации о реализации Windows Sockets

WORD wWSAVer; // Версия Windows Sockets

// Проверяем корректность аргументов командной строки

if (argc != 4) {

puts("Неверные аргументы\n"); // Выводим сообщение об ошибке

puts("Вызов: TCP\_SEND <addr/name> <port> <F>\n"); // Подсказка по использованию

return -1; // Завершаем программу с ошибкой

}

// Инициализация библиотеки сокетов

wWSAVer = MAKEWORD(1, 1); // Версия 1.1

if (WSAStartup(wWSAVer, &WSAData) != 0) { // Запуск библиотеки сокетов

puts("Ошибка инициализации WinSocket");

return -1;

}

// Подготовка структуры с адресом сервера

memset(&SockAddrServer, 0, sizeof(SockAddrServer)); // Обнуляем структуру

SockAddrServer.sin\_family = AF\_INET; // Указываем семейство адресов (IPv4)

// Проверяем, указан ли broadcast-адрес

if (strcmp(argv[1], "255.255.255.255") == 0)

SockAddrServer.sin\_addr.S\_un.S\_addr = INADDR\_BROADCAST; // Устанавливаем широковещательный адрес

else {

SockAddrServer.sin\_addr.S\_un.S\_addr = inet\_addr(argv[1]); // Преобразуем строку IP-адреса в бинарный формат

if (SockAddrServer.sin\_addr.S\_un.S\_addr == INADDR\_NONE) { // Если адрес некорректен

if ((pHostEnt = gethostbyname(argv[1])) == NULL) { // Выполняем DNS-запрос

fprintf(stderr, "Хост не опознан: %s\n", argv[1]); // Ошибка: не удалось получить IP-адрес

return -1;

}

SockAddrServer.sin\_addr = \*(struct in\_addr\*)(pHostEnt->h\_addr\_list[0]); // Берем первый IP-адрес из списка

}

}

// Получаем номер порта из аргументов командной строки

if (sscanf(argv[2], "%u", &nPortServer) < 1) { // Читаем порт и проверяем, корректен ли он

fprintf(stderr, "Ошибочный номер порта: %s\n", argv[2]); // Выводим ошибку

return -1;

}

SockAddrServer.sin\_port = htons((unsigned short)nPortServer); // Преобразуем номер порта в сетевой порядок байтов

// Создаем сокет для передачи данных

SockData = socket(PF\_INET, SOCK\_STREAM, 0); // Создаем TCP-сокет (SOCK\_STREAM)

if (SockData == INVALID\_SOCKET) { // Проверяем, удалось ли создать сокет

fputs("Ошибка создания сокета\n", stderr);

return -1;

}

// Подключаемся к серверу

if (connect(SockData, (const struct sockaddr\*)&SockAddrServer, sizeof(SockAddrServer)) != 0) {

fprintf(stderr, "Ошибка соединения с %s:%u\n",

inet\_ntoa(SockAddrServer.sin\_addr), // Преобразуем IP-адрес в строку

ntohs(SockAddrServer.sin\_port)); // Преобразуем порт в читаемый формат

closesocket(SockData); // Закрываем сокет

return -1;

}

// Выводим сообщение об успешном подключении

fprintf(stdout, "Установлено соединение с сервером: %s:%u\n",

inet\_ntoa(SockAddrServer.sin\_addr),

ntohs(SockAddrServer.sin\_port));

// Основной цикл отправки сообщений

for (i = 3; i < argc; ++i) { // Обрабатываем все аргументы командной строки, начиная с третьего

fprintf(stdout, "Отсылка: \"%s\" \n", argv[i]); // Выводим сообщение о передаче данных

nMsgLen = strlen(argv[i]) + 1; // Вычисляем длину строки (учитываем \0 в конце)

if (send(SockData, argv[i], nMsgLen, 0) < nMsgLen) { // Отправляем данные

fprintf(stderr, "Ошибка отсылки: \"%s\"\n", argv[i]); // Ошибка отправки

continue;

}

// Ожидаем ответ от сервера

fprintf(stdout, "Прием...");

nMsgLen = recv(SockData, DataBuffer, sizeof(DataBuffer) - 1, 0); // Получаем данные от сервера

if (nMsgLen <= 0) { // Если данных нет или произошла ошибка

fputs("Ошибка приема\n", stderr);

continue;

}

DataBuffer[nMsgLen] = '\0'; // Добавляем завершающий нуль-символ

fprintf(stdout, "\b\b\b: \"%s\" \n", DataBuffer); // Выводим полученное сообщение

}

// Завершаем соединение

shutdown(SockData, 2); // Завершаем отправку и прием данных (2 — запрет и приема, и передачи)

Sleep(100); // Даем время закрыть соединение

closesocket(SockData); // Закрываем сокет

SockData = INVALID\_SOCKET; // Обнуляем переменную сокета

WSACleanup(); // Освобождаем ресурсы, связанные с Winsock

return 0; // Выход из программы

}

**Код TCP server**

#define \_CRT\_SECURE\_NO\_WARNINGS // Отключает предупреждения безопасности, связанные с функциями стандартной библиотеки C

#pragma comment(lib, "ws2\_32.lib") // Подключает библиотеку Windows Sockets 2 (ws2\_32.lib) при компиляции

#include <winsock2.h> // Основная библиотека для работы с сокетами в Windows

#include <ws2tcpip.h> // Дополнительные функции для работы с IP-адресами (например, getaddrinfo, inet\_pton)

#include <locale.h> // Библиотека для установки локализации

#include <windows.h> // Заголовочный файл для работы с Windows API

#include <stdio.h> // Стандартный ввод/вывод (printf, scanf и т. д.)

#include <winsock.h> // Устаревшая версия библиотеки для работы с сокетами (используется для совместимости)

#define DEFAULT\_ECHO\_PORT 7 // Определяем стандартный порт для эхо-сервера (7 — стандартный echo-порт)

char DataBuffer[1024]; // Буфер для приема и передачи данных размером 1024 байта

// Функция вычисления числа Фибоначчи

int fibo(int count) {

int a = 1; // Первое число Фибоначчи

int b = 1; // Второе число Фибоначчи

if (count == 0 || count == 1) { // Если входной аргумент 0 или 1, возвращаем 1

return 1;

}

for (int i = 2; i <= count; i++) { // Цикл от 2 до count

int temp = a + b; // Вычисляем следующее число Фибоначчи

a = b; // Сдвигаем числа

b = temp;

}

return b; // Возвращаем результат

}

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

{

// Устанавливаем кодировку консоли для корректного ввода русских символов

SetConsoleCP(1251);

// Устанавливаем кодировку консоли для корректного вывода русских символов

SetConsoleOutputCP(1251);

struct sockaddr\_in SockAddrBase, SockAddrPeer; // Структуры для хранения адресов сервера и клиента

SOCKET SockBase = INVALID\_SOCKET, SockData = INVALID\_SOCKET; // Сокеты: для прослушивания и передачи данных

unsigned short nPort = DEFAULT\_ECHO\_PORT; // Номер порта (по умолчанию 7)

int nAddrSize, nCnt; // Размер структуры адреса и количество полученных байт

WSADATA WSAData; // Структура для информации о версии Windows Sockets

WORD wWSAVer; // Переменная для хранения версии Windows Sockets

// Разбор аргументов командной строки: если передан порт, используем его

if (argc > 1)

if (sscanf(argv[1], "%u", &nPort) < 1) // Читаем порт из аргументов командной строки

fprintf(stderr, "Ошибочный порт: %s, use default", nPort); // Ошибка, если порт некорректен

// Инициализация библиотеки сокетов

wWSAVer = MAKEWORD(1, 1); // Запрашиваем версию 1.1

if (WSAStartup(wWSAVer, &WSAData) != 0) { // Инициализируем WinSock

puts("Ошибка инициализации подсистемы WinSocket");

return -1; // Выход с ошибкой

}

// Создаем сокет для прослушивания входящих соединений

SockBase = socket(PF\_INET, SOCK\_STREAM, 0); // Создаем TCP-сокет

if (SockBase == INVALID\_SOCKET) { // Проверяем успешность создания

fputs("Ошибка создания сокета\n", stderr);

return -1;

}

// Привязываем сокет к локальному адресу и порту

memset(&SockAddrBase, 0, sizeof(SockAddrBase)); // Обнуляем структуру адреса

SockAddrBase.sin\_family = AF\_INET; // Указываем, что используем IPv4

SockAddrBase.sin\_addr.S\_un.S\_addr = INADDR\_ANY; // Слушаем на всех доступных сетевых интерфейсах

SockAddrBase.sin\_port = htons(nPort); // Преобразуем номер порта в сетевой порядок байтов

if (bind(SockBase, (struct sockaddr\*)&SockAddrBase, sizeof(SockAddrBase)) != 0) { // Привязываем сокет

fprintf(stderr, "Ошибка привязки к локальному порту: %u\n", ntohs(SockAddrBase.sin\_port));

return -1;

}

// Включаем режим прослушивания входящих соединений

if (listen(SockBase, 2) != 0) { // Очередь ожидания - максимум 2 клиента

closesocket(SockBase);

fputs("Ошибка включения режима прослушивания\n", stderr);

return -1;

}

// Выводим сообщение о запуске сервера

fprintf(stderr, "Сервер запущен, порт %u\n", ntohs(SockAddrBase.sin\_port));

// Основной рабочий цикл сервера

while (1) { // Бесконечный цикл для обработки клиентов

nAddrSize = sizeof(SockAddrPeer); // Указываем размер структуры адреса клиента

SockData = accept(SockBase, (struct sockaddr\*)&SockAddrPeer, &nAddrSize); // Принимаем входящее соединение

if (SockData == INVALID\_SOCKET) { // Проверяем успешность

fputs("Ошибка приема соединения\n", stderr);

continue; // Пропускаем итерацию

}

// Обрабатываем соединение с клиентом

while (1) {

nCnt = recv(SockData, DataBuffer, sizeof(DataBuffer) - 1, 0); // Получаем данные от клиента

if (nCnt <= 0) // Если произошла ошибка или клиент закрыл соединение

break;

int res = fibo(atoi(DataBuffer)); // Вычисляем число Фибоначчи по переданному числу

printf("\nНа сервере произошли вычисления числа Фибоначчи.\nРезультат = %d", res);

memset(DataBuffer, 0, sizeof(DataBuffer)); // Очищаем буфер

snprintf(DataBuffer, sizeof(DataBuffer), "%d", res); // Конвертируем результат в строку

send(SockData, DataBuffer, strlen(DataBuffer), 0); // Отправляем результат клиенту

}

// Завершаем соединение с клиентом

shutdown(SockData, 2); // Закрываем канал передачи данных

closesocket(SockData); // Закрываем сокет

SockData = INVALID\_SOCKET;

}

// Завершение работы сервера (на самом деле, этот код никогда не выполняется)

shutdown(SockBase, 2); // Завершаем соединения

Sleep(100); // Даем системе время завершить соединения

closesocket(SockBase); // Закрываем сокет сервера

SockBase = INVALID\_SOCKET;

WSACleanup(); // Очищаем ресурсы Winsock

return 0; // Выход из программы

}