Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Теоретической и прикладной математики

Лабораторная работа № 2

по дисциплине «Операционные системы, среды и оболочки»

Технология клиент-сервер: эхо-повтор

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	6
Студент:	Сидоров Даниил,
	Дюков Богдан
Преподаватель:	Кобылянский Валерий Григорьевич,
	Филиппова Елена Владимировна

Новосибирск

2026

1. Цель работы

Изучить основные принципы разработки клиент-серверных приложений на примере простейшей однопользовательской программы.

2. Ход работы

Написали простейшее приложение с одним сервером и одним клиентом, используя API-интерфейс низкого уровня.

Запуск сервера происходит с указанием номера порта (2006) протокола TCP или UDP из диапазона возможных номеров. Сервер начинает свою работу с ожидания запроса от клиента на соединение.

Условие задачи: клиент пересылает серверу имя некоторого файла. Сервер находит файл с указанным именем и пересылает его содержимое клиенту, либо сообщает клиенту, что файл с данным именем не найден.

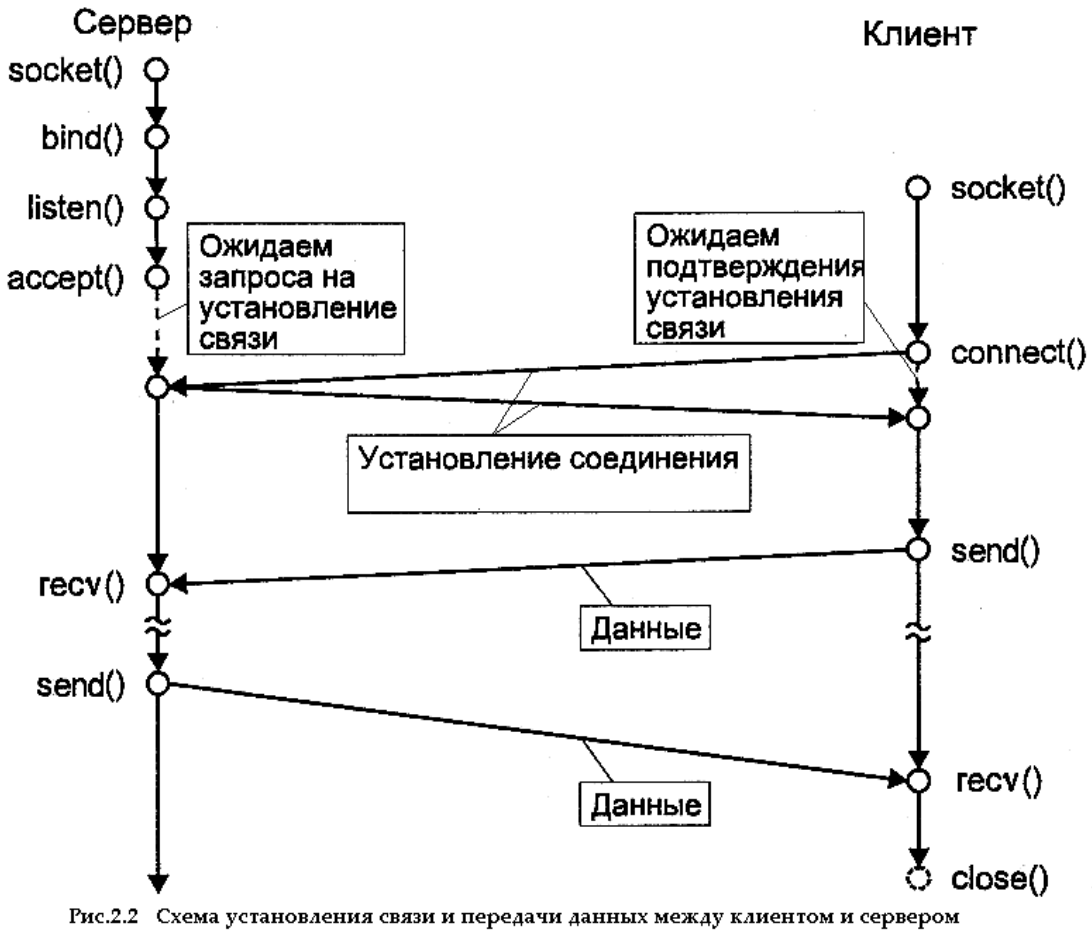

Рис.2.2 Схема установления связи и передачи данных между клиентом и сервером

Описание использованных функций

FindFile – поиск на сервере файла с расширением .txt по имени, которое было передано клиентской программой. Можно выбрать любой путь на компьютере, начиная с которого будет производиться поиск.

socket – создание объекта типа сокет, с помощью которого организуется канал связи с другим компьютером и запускается процесс приема/передачи сообщений;

bind – привязка сокета к конкретному адресу;

listen – перевод сервера в режим прослушивания порта и ожидания запросов на установление соединения от клиентов;

connect – установление соединения с сервером в клиентский программах;

accept – создание нового сокета, через который будет проводиться обмен данными с клиентом, при этом слушающий сокет сервера по прежнему работает в режиме прослушивания порта для получения новых запросов на соединение от клиентов;

send - передача данных, если сокет подключен;

recv – прием данных из подключенного сокета;

closeSocket - сообщение операционной системе, что программа завершила использование сокета;

3. Код Сервера

```
#pragma comment (lib,"Ws2_32.lib")

#include <winsock2.h>
#include <stdio.h>
#include <iostream>
#include <sstream>
#include <string>
#include <Shlwapi.h>

using namespace std;

// Размерность массивов путей к файлу и ip-адресов клиентов
const int N = 256;

const int resultSize = 100000;
const wchar_t* selectedPath = L"C:\\*";

bool FindFile(wchar_t* nameOfCurrentDirectory, wchar_t* foundfile, wchar_t* searchFile)
{
    int slash;
    bool fileFound = false, isFile;
    wchar_t fullFileAddress[N];

    // Создаем структуру, в которой будет содержаться информация о файле, найденном с
помощью FindFirstFile или FindNextFile
    // Потом вызываем функцию FindFirstFile, которая возвращает информацию о первом
найденном файле
    WIN32_FIND_DATA file;
    HANDLE search_handle = FindFirstFile(nameOfCurrentDirectory, &file);

    do
    {

        if (search_handle != INVALID_HANDLE_VALUE)
        {
            isFile = false;

            // Обнуляем строку полного пути к файлу
            fullFileAddress[0] = '\0';

            // Перепишем в строку полного пути название текущей директории
            for (int i = 0; i < wcslen(nameOfCurrentDirectory) - 1; i++, slash = i)
            {
                fullFileAddress[i] = nameOfCurrentDirectory[i];
            }

            // Добавляем к названию текущей директории имя найденного файла
            // Проверяем в процессе, файл ли найден (.txt или .docx например),
потому что можно найти директорию
            for (int i = 0; i < wcslen(file.cFileName); i++, slash++)
            {
                fullFileAddress[slash] = file.cFileName[i];

                if (fullFileAddress[slash] == L'.')
                {
                    isFile = true;
                }

            }

            fullFileAddress[slash] = '\0';

            // Проверим, является ли файл формата .txt
            if (fullFileAddress[slash - 4] == L'.' && fullFileAddress[slash - 3] ==
L't'
```

```
                        && fullFileAddress[slash - 2] == L'x' && fullFileAddress[slash -
1] == L't')
                {

                        // Если название текущего файла совпадает с искомым (функция
wcscmp() возвращает ноль, если переданные строки равны)
                        // То сохраняем результат и завершаем выполнение функции
                        if (wcscmp(file.cFileName, searchFile) == 0)
                        {
                                fileFound = true;
                                wcscpy(foundfile, fullFileAddress);
                        }

                }
                else if (!isFile) // Если текущий адрес - директория
                {
                        fullFileAddress[slash] = L'\\';
                        slash++;
                        fullFileAddress[slash] = L'*';
                        slash++;
                        fullFileAddress[slash] = L'\0';

                        // Рекурсивный запуск поиска в директории
                        fileFound = FindFile(fullFileAddress, foundfile, searchFile);
                }

        }

    } while (!fileFound && FindNextFile(search_handle, &file));

    return fileFound;
}

int main(void)
{
    WSADATA wsaData;

    // Загрузка (инициализация) библиотеки с помощью функции WSAStartup, она возвращает 0,
если инициализация прошла успешно
    // Первый параметр - запрашиваемая версия библиотеки Winsock (версия 2.2)
    // Второй параметр - ссылка на структуру WSAData, в которую возвратятся параметры
инициализации
    // В структуре содержится, например, номер используемой версии Winsock, максимальное
кол-во открытых сокетов одновременно и т. д.
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        printf("Error WinSock version initializaion\n");
        WSACleanup();
        system("pause");
        return SOCKET_ERROR;
    }

    // Создаем объект типа сокет и возвращаем его дескриптор
    // Первый параметр - определяет имя коммуникационного домена
    // Коммуникационный домен определяет систему адресации соединений: для домена Internet
адрес записывается в виде IP-адреса и номера порта
    // Коммуникационный домен определяет также используемые семейства протоколов. Так, для
домена AF_INET – протоколы TCP/IP.
    // Второй параметр - тип связи, использующийся в сокете: обычно задается тип
транспортного протокола TCP (SOCK_STREAM) или UDP (SOCK_DGRAM)
    // Третий параметр необязательный, если тип сокета указан как TCP или UDP – можно
передать значение 0
    SOCKET servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

    if (servSock == INVALID_SOCKET)
    {
        printf("Unable to create socket\n");
        WSACleanup();
        system("pause");
```

```
		return SOCKET_ERROR;
	}

	// Заполнение информации об адресе сокета
	SOCKADDR_IN sin;
	sin.sin_family = AF_INET; // Указываем имя коммуникационного домена
	sin.sin_port = htons(2006); // Порт: номер бригады + 2000, может быть любым, главное, чтобы он не был зарезервирован другой программой
	// IP - адрес
	// Только один IP - сокет может быть связан с каждой заданной локальной парой (адрес, порт)
	// Если при вызове bind указать INADDR_ANY, то сокет будет связан со всеми локальными интерфейсами
	sin.sin_addr.s_addr = INADDR_ANY;

	// Привязка сокета к конкретному адресу (параметры: дескриптор сокета, указатель на структуру sin, размер структуры)
	int retVal = bind(servSock, (LPSOCKADDR)&sin, sizeof(sin));

	if (retVal == SOCKET_ERROR)
	{
		printf("Unable to bind\n");
		WSACleanup();
		system("pause");
		return SOCKET_ERROR;
	}

	// Узнаем IPv4-адрес хоста, чтобы вывести его в консоли
	// Сначала получаем имя хоста, после получаем ip хоста по его имени
	char host[N];
	char HostName[1024]; // Буфер для имени хоста

	if (gethostname(HostName, 1024) == 0)
	{
		strcpy(host, inet_ntoa(*((in_addr*)gethostbyname(HostName)->h_addr_list[0])));
	}

	printf("Server started at %s, port %d\n", host, htons(sin.sin_port));

	while (true)
	{
		// Пытаемся начать слушать сокет
		retVal = listen(servSock, 10);

		if (retVal == SOCKET_ERROR)
		{
			printf("Unable to listen\n");
			WSACleanup();
			system("pause");
			return SOCKET_ERROR;
		}

		// Ждем клиента
		// На этапе, когда выполняется функция accept, программа останавливается и ждет запроса на установление соединения от клиентов
		// Как только появился запрос, программа идет дальше
		SOCKET clientSock;
		SOCKADDR_IN from;
		int fromlen = sizeof(from);

		clientSock = accept(servSock, (struct sockaddr*)&from, &fromlen);

		if (clientSock == INVALID_SOCKET)
		{
			printf("Unable to accept\n");
			WSACleanup();
			system("pause");
			return SOCKET_ERROR;
		}
```

```
            printf("New connection accepted from %s, port %d\n", inet_ntoa(from.sin_addr),
htons(from.sin_port));

            // Пытаемся получить данные от клиента, клиент должен отправить имя файла,
которое запишется в szReq
            // Функция возвращает 0 в случае успешной передачи, иначе -1 (например клиент
завершил программу и оборвал соединение)
            char szReq[N];
            retVal = recv(clientSock, szReq, N, 0);

            if (retVal == SOCKET_ERROR)
            {
                  printf("Unable to recv\n");
                  system("pause");
                  return SOCKET_ERROR;
            }

            printf("Data received\n");

            // Выключаем сервер, если клиент отправил "s"
            if (szReq[0] == 's' && szReq[1] == '\0')
            {
                  const char* szResp = "Server shutdown";
                  retVal = send(clientSock, szResp, 256, 0);
                  closesocket(clientSock);
                  break;
            }
            else
            {
                  char charFoundFile[N];
                  sprintf(charFoundFile, "%s", szReq);

                  wchar_t foundFile[N];
                  wchar_t searchFile[N];

                  int i;

                  // Запишем название искомого файла в тип wchar_t
                  for (i = 0; i < strlen(szReq); i++)
                  {
                        searchFile[i] = szReq[i];
                  }

                  searchFile[i] = '\0';

                  // Запускаем поиск
                  bool isFound = FindFile((wchar_t*)selectedPath, foundFile, searchFile);

                  char c;
                  char result[resultSize];
                  int resultIndex = 0;

                  // Если искомый файл был найден
                  if (isFound)
                  {
                        FILE* file;

                        // Копируем названием искомого файла в тип char
                        for (i = 0; i < wcslen(foundFile); i++)
                        {
                              charFoundFile[i] = foundFile[i];
                        }

                        charFoundFile[i] = '\0';

                        printf("Requested file is found at: %s\n", charFoundFile);

                        // Откроем файл для чтения
```

```
                        fopen_s(&file, charFoundFile, "r");

                        char line[1000];

                        while (!feof(file))
                        {

                                // Если считана не пустая строка
                                if (fscanf(file, "%[^\n]s", line) == 1)
                                {

                                        // Запишем строку в результат
                                        for (int i = 0; i < strlen(line); i++,
resultIndex++)
                                        {
                                                result[resultIndex] = line[i];
                                        }

                                        // Перейдем на чтение новой строки
                                        fscanf(file, "%c", &c);

                                        result[resultIndex] = '\n';
                                        resultIndex++;
                                }
                                else
                                {
                                        result[resultIndex] = '\n';
                                        resultIndex++;

                                        fscanf(file, "%c", &c);
                                }

                        }

                        result[resultIndex] = '\0';

                        // Отправим ответ клиенту
                        retVal = send(clientSock, result, resultSize, 0);
                }
                else
                {
                        printf("Requested file is not found\n");

                        // Отправим сообщение о несуществующем файле
                        retVal = send(clientSock, "No such file", N, 0);
                }

                printf("Sending response from server\n");

                if (retVal == SOCKET_ERROR)
                {
                        printf("Unable to send\n");
                        system("pause");
                        return SOCKET_ERROR;
                }

                // Закрываем клиентский сокет
                closesocket(clientSock);
                printf("Connection closed\n");
        }
    }

    // Закрываем серверный сокет
    closesocket(servSock);
    WSACleanup();

    return 0;
}
```

4. Код Клиента

```
#pragma comment(lib, "Ws2_32.lib")

#include <winsock.h>
#include <stdio.h>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

int main()
{
    // Структура WSAData, в которую возвратятся параметры инициализации
    // В структуре содержится, например, номер используемой версии Winsock, максимальное
кол-во открытых сокетов одновременно и т. д.
    WSADATA wsaData;

    // Функция WSAStartup инициирует использование Windows Sockets DLL процессом.
    WSAStartup(MAKEWORD(2, 2), (LPWSADATA)&wsaData);

    // Функция gethostbyname получает информацию о хосте, соответствующую имени хоста, из
базы данных хоста.
    if (!gethostbyname("localhost"))
    {
        cout << "unable to collect gethostbyname" << endl;

        // Освобождение ресурсов
        WSACleanup();
        return 1;
    }

    string ip;

    cout << "ip:";
    cin >> ip;

    // Удаляем /n
    cin.ignore();

    // Массив под название файла
    char fileName[256];
    fileName[0] = '\0';

    // Переменная для Y/N
    char yesNo = 'Y';

    while (yesNo == 'Y' && fileName[0] != 's' && fileName[1] != '\0')
    {
        // Заполнение информации об адресе сокета
        SOCKADDR_IN serverInfo;
        serverInfo.sin_family = PF_INET;
        serverInfo.sin_addr.S_un.S_addr = inet_addr(ip.c_str());
        // Порт. Функция htons преобразует u_short из хоста в сетевой порядок байтов
TCP/IP
        serverInfo.sin_port = htons(2006);

        // Создаем объект типа сокет и возвращаем его дескриптор
        SOCKET clientSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

        if (clientSock == INVALID_SOCKET)
        {
            cout << "unable to create socket" << endl;

            // Освобождение ресурсов
            WSACleanup();
```

```
		return 1;
	}

	// Установление соединения с сервером
	int retVal = connect(clientSock, (LPSOCKADDR)&serverInfo, sizeof(serverInfo));

	if (retVal == SOCKET_ERROR)
	{
		cout << "unable to connect" << endl;
		WSACleanup();
		return 1;
	}

	cout << "connection made sucessfully" << endl;
	cout << "enter name of file" << endl;

	// Ввод имени файла
	scanf("%s", fileName);

	cout << "sending request from client" << endl;

	// Отправляем имя файла на сервер
	retVal = send(clientSock, fileName, 256, 0);

	if (retVal == SOCKET_ERROR)
	{
		cout << "unable to send" << endl;
		WSACleanup();
		return 1;
	}

	// Получаем содержимое файла от сервера и записываем его в szResponce
	char szResponce[100000];
	retVal = recv(clientSock, szResponce, 100000, 0);

	if (retVal == SOCKET_ERROR)
	{
		cout << "unable to recv" << endl;
		WSACleanup();
		return 1;
	}

	char* Resp = szResponce;

	// Выводим содержимое файла
	if (fileName[0] != 's')
	{
		printf("Content of file:\n %s \n", Resp);
	}
	else
	{
		printf("%s \n", Resp);
	}

	// Предложение продолжить
	cout << endl << "Do you want to continue ? Y / N" << endl;
	cin >> yesNo;

	while (yesNo != 'Y' && yesNo != 'N')
	{
		cout << endl << "Do you want to continue ? Y / N" << endl;
		cin >> yesNo;
	}

	// Закрываем клиентский сокет
	closesocket(clientSock);
}

WSACleanup();
```

```
    return 0;
}
```

5. Тесты

Сервер	Клиент	Примечание
Server started at 26.70.89.129, port 2006 New connection accepted from 26.70.89.129, port 51291 Data received Requested file is not found Sending response from server Connection closed New connection accepted from 26.70.89.129, port 51305 Data received Requested file is found at: D:\testDiskD.txt Sending response from server Connection closed	ip:26.70.89.129 connection made sucessfully enter name of file testDiskD.txt sending request from client Content of file: okay Do you want to continue ? Y / N	**Поиск на диске D**
Server started at 26.70.89.129, port 2006 New connection accepted from 26.70.89.129, port 51240 Data received Requested file is found at: C:\Users\danys\Desktop\test.txt Sending response from server Connection closed New connection accepted from 26.70.89.129, port 51243 Data received Requested file is found at: C:\Users\danys\AppData\Roaming\dekovir\craftheworld\lang.txt Sending response from server Connection closed	ip:26.70.89.129 connection made sucessfully enter name of file test.txt sending request from client Content of file: test word book Do you want to continue ? Y / N Y connection made sucessfully enter name of file lang.txt sending request from client Content of file: russian	**Поиск на диске C**
New connection accepted from 26.70.89.129, port 51319 Data received Requested file is not found Sending response from server Connection closed	connection made sucessfully enter name of file gdfgdfgfdgd.txt sending request from client Content of file: No such file Do you want to continue ? Y / N	**Файл не найден**
	ip:1.1.1.11.1 unable to connect	**Неверный ip**
Server started at 26.70.89.129, port 2006 New connection accepted from 26.70.89.129, port 51350 Data received	ip:26.70.89.129 connection made sucessfully enter name of file s sending request from client Server shutdown	**Остановка сервера**
Server started at 26.70.89.129, port 2006 New connection accepted from 26.70.89.129, port 51437 Unable to recv		**Клиент отключился**
	ip:26.70.89.129 connection made sucessfully enter name of file wow.txt sending request from client unable to send	**Сервер был выключен**

6. Вывод

Контрольные вопросы проработаны.