|  |  |  |
| --- | --- | --- |
| 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 | | |
| 2022 | | |

1. **Цель работы**

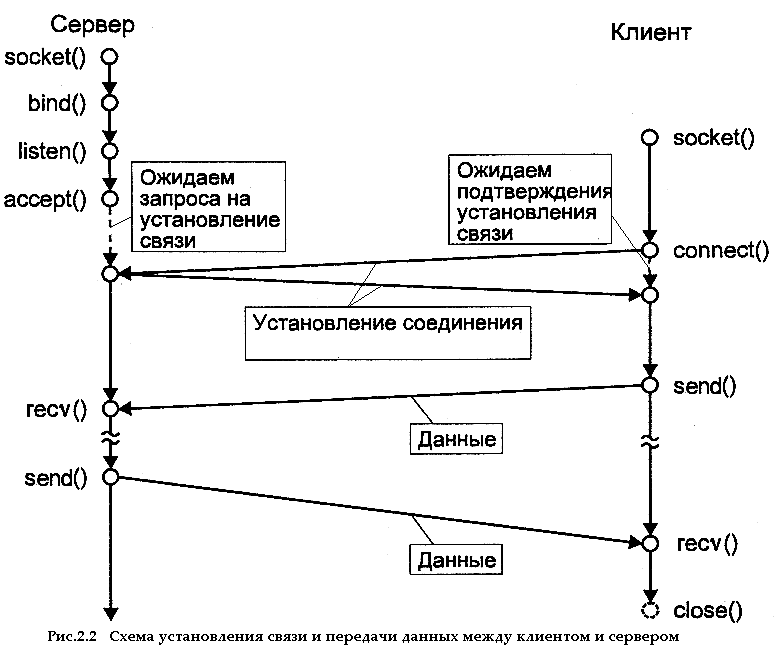
Изучить основные принципы разработки клиент-серверных приложений на примере простейшей однопользовательской программы.

1. **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.

****

**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;

сonnect – 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;

1. **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;

}

// Cоздаем 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;

}

1. **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);

// Cоздаем 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;

}

1. **Тесты**

|  |  |  |
| --- | --- | --- |
| **Сервер** | **Клиент** | **Примечание** |
|  |  | **Поиск на диске D** |
|  |  | **Поиск на диске С** |
|  |  | **Файл не найден** |
|  |  | **Неверный ip** |
|  |  | **Остановка сервера** |
|  |  | **Клиент отключился** |
|  |  | **Сервер был выключен** |

1. **Вывод**

Контрольные вопросы проработаны.