***Сетевое программирование сокетов: понятие, работа с протоколом HTTP.***

***Разработать функцию DeleteSpace(S), удаляющую в строке S лишние пробелы. В основной программе применить функцию двум исходным текстовым файлам, список которых задает пользователь. Вывод исправленного текста осуществить в файлы с префиксом check.***

**Сетевое программирование сокетов** — это работа с сетью на низком уровне с использованием сокетов, которые предоставляют интерфейс доступа к определённому порту определённого хоста. Через сокет один хост может обращаться к приложению на другом хосте.

Работа с протоколом HTTP в контексте сетевого программирования сокетов предполагает реализацию поддержки работы с заголовками запроса и ответа, получение метода запроса, версии протокола и URL-адреса. Например, сервер может принимать запросы клиентов, парсить заголовки и тело запроса и возвращать тестовую HTML-страничку, на которой отображены данные запроса клиента (запрошенный URL, метод запроса, cookie и другие заголовки).

В .NET сокеты представлены классом Socket из пространства имён System.NET.Sockets, который предоставляет низкоуровневый интерфейс для приёма и отправки сообщений по сети.

**Сетевое программирование с использованием сокетов и протокола HTTP** — это важная область разработки, позволяющая создавать веб-клиенты и серверы. HTTP (Hypertext Transfer Protocol) — это протокол прикладного уровня, работающий поверх TCP и предназначенный для передачи гипертекста (HTML), изображений, видео и других ресурсов.

**Понятие HTTP и его работа с сокетами**

HTTP использует сокеты TCP для установления соединения между клиентом (например, веб-браузером) и сервером (например, веб-сервером). Клиент отправляет HTTP-запрос на сервер, а сервер возвращает HTTP-ответ.

**HTTP-запрос состоит из:**

Строки запроса:

Метод (например, GET, POST, PUT, DELETE).

URI (Uniform Resource Identifier) - путь к запрошенному ресурсу.

Версия протокола (например, HTTP/1.1).

Пример: GET /index.html HTTP/1.1

Заголовки:

Дополнительная информация о запросе, например, User-Agent, Host, Content-Type, Accept.

Пример:

Host: [www.example.com](file:///C:\Users\King%20Night\Documents\GitHub%20Repository\UP_sys\UltimateProject\Билет%2012\www.example.com)

User-Agent: MyBrowser/1.0

Тело запроса (необязательно):

Данные, отправляемые на сервер (например, в случае POST-запроса).

**HTTP-ответ состоит из:**

Строки состояния:

Версия протокола.

Код состояния (например, 200 OK, 404 Not Found, 500 Internal Server Error).

Сообщение состояния.

Пример: HTTP/1.1 200 OK

Заголовки ответа:

Дополнительная информация об ответе, например, Server, Content-Type, Content-Length.

Пример:

Server: Apache/2.4.18

Content-Type: text/html

Content-Length: 1234

Тело ответа (обычно содержит запрошенный ресурс):

HTML-страница, изображение, данные JSON и т.д.

**Реализация HTTP в C++**

В C++ можно использовать “чистые” сокеты (низкоуровневые API) или специализированные библиотеки для работы с HTTP. Использование библиотек значительно упрощает процесс, так как они берут на себя большую часть обработки HTTP.

**Использование “чистых” сокетов (пример клиента):**

#include <iostream>

#include <string>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h> // для close()

int main() {

int sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock == -1) {

std::cerr << "Failed to create socket" << std::endl;

return 1;

}

sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_port = htons(80); // HTTP port

inet\_pton(AF\_INET, "127.0.0.1", &(server\_addr.sin\_addr)); // IP address

if (connect(sock, (sockaddr\*)&server\_addr, sizeof(server\_addr)) == -1) {

std::cerr << "Failed to connect" << std::endl;

close(sock);

return 1;

}

std::string request = "GET / HTTP/1.1\r\nHost: 127.0.0.1\r\nConnection: close\r\n\r\n";

send(sock, request.c\_str(), request.length(), 0);

char buffer[4096];

int bytes\_received;

while ((bytes\_received = recv(sock, buffer, sizeof(buffer), 0)) > 0) {

std::cout.write(buffer, bytes\_received);

}

close(sock);

return 0;

}

**Использование библиотеки libcurl (пример клиента):**

#include <iostream>

#include <curl/curl.h>

size\_t WriteCallback(void \*contents, size\_t size, size\_t nmemb, std::string \*output) {

size\_t total\_size = size \* nmemb;

output->append((char\*)contents, total\_size);

return total\_size;

}

int main() {

CURL \*curl;

CURLcode res;

curl\_global\_init(CURL\_GLOBAL\_DEFAULT);

curl = curl\_easy\_init();

if(curl) {

std::string output;

curl\_easy\_setopt(curl, CURLOPT\_URL, "[http://127.0.0.1](http://127.0.0.1/)"); // URL to request

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &output);

res = curl\_easy\_perform(curl);

if(res != CURLE\_OK)

std::cerr << "curl\_easy\_perform() failed: " << curl\_easy\_strerror(res) << std::endl;

else

std::cout << output << std::endl;

curl\_easy\_cleanup(curl);

}

curl\_global\_cleanup();

return 0;

}

**Реализация HTTP в C#**

C# имеет встроенную поддержку HTTP с использованием класса HttpClient.

**Пример клиента в C#:**

using System;

using System.Net.Http;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using (HttpClient client = new HttpClient())

{

try

{

HttpResponseMessage response = await client.GetAsync("[http://127.0.0.1](http://127.0.0.1/)"); // URL to request

response.EnsureSuccessStatusCode();

string responseBody = await response.Content.ReadAsStringAsync();

Console.WriteLine(responseBody);

}

catch (HttpRequestException e)

{

Console.WriteLine($"Request error: {e.Message}");

}

}

}

}

**Пример сервера в C# (с использованием HttpListener):**

using System;

using System.Net;

using System.Text;

using System.Threading.Tasks;

class HttpServer

{

public static async Task RunServer()

{

HttpListener listener = new HttpListener();

listener.Prefixes.Add("http://127.0.0.1:8080/");

listener.Start();

Console.WriteLine("Listening on <http://127.0.0.1:8080/>");

while (true)

{

HttpListenerContext context = await listener.GetContextAsync();

HttpListenerRequest request = context.Request;

HttpListenerResponse response = context.Response;

string responseString = "<html><body><h1>Hello from C# HTTP Server!</h1></body></html>";

byte[] buffer = Encoding.UTF8.GetBytes(responseString);

response.ContentLength64 = buffer.Length;

System.IO.Stream output = response.OutputStream;

output.Write(buffer, 0, buffer.Length);

output.Close();

}

}

static void Main(string[] args)

{

RunServer().GetAwaiter().GetResult();

}

}