МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

Отчёт
о лабораторной работе №6
по дисциплине «Веб-технологии»

Тема: «Простейший HTTP сервер.»

Выполнил студент 2 курса
группы ПО-11 Сымоник И.А.

Проверил: Михняев А.Л.

Брест 2024

Цель работы: изучить работу http сервера и реализовать его.

Ход работы

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <thread>
#include <unordered_map>

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

#define DEFAULT_PORT "8080"
#define DEFAULT_BUFLEN 1024

void logError(const std::string& errorMessage) {
    std::ofstream logFile("server.log", std::ios_base::app);
    if (logFile.is_open()) {
        logFile << errorMessage << std::endl;
        logFile.close();
    }
}

std::string getFileContent(const std::string& filePath) {
    std::ifstream file(filePath);
    if (!file.is_open()) {
```

```
        return "";
    }
    std::stringstream buffer;
    buffer << file.rdbuf();
    return buffer.str();
}

std::string getContentType(const std::string& path)
{
    if (path.ends_with(".html") || path.ends_with(".htm")) {
        return "text/html";
    }
    else if (path.ends_with(".json")) {
        return "application/json";
    }
    else if (path.ends_with(".xml")) {
        return "application/xml";
    }
    return "text/plain";
}

std::string handleRequest(const std::string& request, bool& closeConnection) {
    std::istringstream requestStream(request);
    std::string method, path, version;

    std::string requestLine;
    while (std::getline(requestStream, requestLine)) {
```

```
    if (!requestLine.empty() && requestLine != "\r") {
        break;
    }
}

if (!requestLine.empty() && requestLine.back() == '\r') {
    requestLine.pop_back();
}

std::istringstream requestLineStream(requestLine);
requestLineStream >> method >> path >> version;

closeConnection = (version == "HTTP/1.0");

std::string headers;
std::string body;
bool hostHeaderFound = false;
std::unordered_map<std::string, std::string> headersMap;

std::string line;
while (std::getline(requestStream, line)) {
    if (!line.empty() && line.back() == '\r') {
        line.pop_back();
    }

    if (line.empty()) {
        continue;
```

```
        }

        size_t colonPos = line.find(':');
        if (colonPos != std::string::npos) {
            std::string key = line.substr(0, colonPos);
            std::string value = line.substr(colonPos + 1);
            headersMap[key] = value;
        }

        if (line.find("Host:") != std::string::npos) {
            hostHeaderFound = true;
        }

        if (line == "") {
            break;
        }
    }

    if (version == "HTTP/1.1" && !hostHeaderFound) {
        headers = "HTTP/1.1 400 Bad Request\r\nContent-Type:
text/plain\r\n\r\n";
        body = "400 Bad Request: Host header is missing";
        return headers + body;
    }

    if (method == "GET" || method == "HEAD") {
        std::string filePath = "." + path;
        if (filePath == "./") {
```

```
            filePath = "./index.html";
        }

        body = getFileContent(filePath);
        if (body.empty()) {
            headers = version + " 404 Not Found\r\nContent-Type: text/plain\r\nContent-Length: 13\r\n\r\n";
            body = "404 Not Found";
        }
        else {
            std::string contentType = getContentType(filePath);
            headers = version + " 200 OK\r\nContent-Type: " + contentType + "\r\nContent-Length: " + std::to_string(body.size()) + "\r\n\r\n";
        }

        if (method == "HEAD") {
            return headers;
        }
        else {
            return headers + body;
        }
    }
    else {
        body = "405 Method Not Allowed";
        headers = version + " 405 Method Not Allowed\r\nContent-Type: text/plain\r\nContent-Length: " + std::to_string(body.size()) + "\r\n\r\n";
        return headers + body;
    }
```

```
}

void handleClient(SOCKET ClientSocket) {
    char recvbuf[DEFAULT_BUFLEN];
    int recvbuflen = DEFAULT_BUFLEN;
    int iResult;
    std::string request;
    bool headersReceived = false;

    do {
        iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);
        if (iResult > 0) {
            request.append(recvbuf, iResult);

            size_t headerEndPos = request.find("\r\n\r\n");
            if (headerEndPos != std::string::npos) {
                headersReceived = true;
                bool closeConnection = false;
                std::string response = handleRequest(request.substr(0, headerEndPos
+ 4), closeConnection);
                int sendResult = send(ClientSocket, response.c_str(), response.size(),
0);
                if (sendResult == SOCKET_ERROR) {
                    std::cerr << "send failed: " << WSAGetLastError() << std::endl;
                    logError("send failed: " + std::to_string(WSAGetLastError()));
                }

                if (closeConnection) {
```

```
                iResult = shutdown(ClientSocket, SD_SEND);
                if (iResult == SOCKET_ERROR) {
                    std::cerr << "shutdown failed: " << WSAGetLastError() <<
std::endl;
                    logError("shutdown failed: " +
std::to_string(WSAGetLastError()));
                }
                break;
            }
            else {
                request = request.substr(headerEndPos + 4);
            }
        }
    }
    else if (iResult == 0) {
        std::cerr << "Connection closing...\n";
    }
    else {
        std::cerr << "recv failed: " << WSAGetLastError() << std::endl;
        logError("recv failed: " + std::to_string(WSAGetLastError()));
    }
  } while (iResult > 0);

  closesocket(ClientSocket);
}

int main() {
  WSADATA wsaData;
```

```
    int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        std::cerr << "WSAStartup failed: " << iResult << std::endl;
        logError("WSAStartup failed: " + std::to_string(iResult));
        return 1;
    }

    struct addrinfo* result = nullptr;
    struct addrinfo hints;

    ZeroMemory(&hints, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_protocol = IPPROTO_TCP;
    hints.ai_flags = AI_PASSIVE;

    iResult = getaddrinfo(nullptr, DEFAULT_PORT, &hints, &result);
    if (iResult != 0) {
        std::cerr << "getaddrinfo failed: " << iResult << std::endl;
        logError("getaddrinfo failed: " + std::to_string(iResult));
        WSACleanup();
        return 1;
    }

    SOCKET ListenSocket = socket(result->ai_family, result->ai_socktype,
result->ai_protocol);
    if (ListenSocket == INVALID_SOCKET) {
        std::cerr << "Error at socket(): " << WSAGetLastError() << std::endl;
```

```
    logError("Error at socket(): " + std::to_string(WSAGetLastError()));
    freeaddrinfo(result);
    WSACleanup();
    return 1;
}

iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);
if (iResult == SOCKET_ERROR) {
    std::cerr << "bind failed: " << WSAGetLastError() << std::endl;
    logError("bind failed: " + std::to_string(WSAGetLastError()));
    freeaddrinfo(result);
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}

freeaddrinfo(result);

iResult = listen(ListenSocket, SOMAXCONN);
if (iResult == SOCKET_ERROR) {
    std::cerr << "listen failed: " << WSAGetLastError() << std::endl;
    logError("listen failed: " + std::to_string(WSAGetLastError()));
    closesocket(ListenSocket);
    WSACleanup();
    return 1;
}
```

```
    while (true) {
        SOCKET ClientSocket = accept(ListenSocket, nullptr, nullptr);
        if (ClientSocket == INVALID_SOCKET) {
            std::cerr << "accept failed: " << WSAGetLastError() << std::endl;
            logError("accept failed: " + std::to_string(WSAGetLastError()));
            closesocket(ListenSocket);
            WSACleanup();
            return 1;
        }

        std::thread clientThread(handleClient, ClientSocket);
        clientThread.detach();
    }

    closesocket(ListenSocket);
    WSACleanup();

    return 0;
}
```

Вывод: изучили работу http сервера и реализовали его.