ГУО “БГУИР”

Инженерно-экономический факультет

Кафедра экономической информатики

Отчёт по

Лабораторной работе № 2

**Создание последовательного сервера с установлением логического соединения UDP**

Подготовил:

Студент гр.110101

Пантус Р. В.

Проверил:

Бутов А. А.

Минск 2023

**Цель работы:** изучить методы создания серверов без установления логического соединения *UDP*, используя алгоритм последовательной обработки запросов**.**

**Вариант 3:** Клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши «Ввод». Сервер, получив эту строку, должен выяснить, имеются ли среди символов этой строки все буквы, входящие в слово WINDOWS. Количество вхождений символов в строку передать назад клиенту.

Код клиентской части:

#include <iostream>

#include <cstring>

#include <sstream>

#include <stdexcept>

#include <utility>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <arpa/inet.h>

// appearance

#define CLEAR\_LINE() (std::cout << "\033[2K")

#define GO\_UP\_LINE() (std::cout << "\033[1A")

class UDPClient {

public:

UDPClient(const int& kPort\_ = 9984,

std::string kIpAddress\_ = "127.0.0.1") : kPort(kPort\_), kIpAddress(std::move(kIpAddress\_))

{

client\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (client\_fd < 0) {

throw std::runtime\_error("Cant create socket descriptor.");

}

server\_address.sin\_family = AF\_INET; // IPv4

server\_address.sin\_port = htons(kPort);

address\_size = (socklen\_t\*)sizeof(server\_address);

if (inet\_pton(AF\_INET, kIpAddress.c\_str(), &server\_address.sin\_addr) <= 0) {

throw std::runtime\_error("Cant convert IP address from text to binary form");

}

timeval tv {0, 100000};

if (setsockopt(client\_fd, SOL\_SOCKET, SO\_RCVTIMEO, &tv, sizeof(tv)) < 0) {

throw std::runtime\_error("Cant set timeout for socket");

}

}

std::string getResponse(const std::string& message) const {

// send size => send msg

uint32\_t msg\_size = htonl(message.size());

static std::string kServerUnavailabaleMsg = "Response: Time Limit exceeded";

sendto(client\_fd, &msg\_size, sizeof(msg\_size), 0,

(sockaddr\*)&server\_address, sizeof(server\_address)

);

sendto(client\_fd, message.c\_str(), message.size(), 0,

(sockaddr\*)&server\_address, sizeof(server\_address)

);

msg\_size = 0;

// get msg size from server => accepted response

recvfrom(client\_fd, &msg\_size, sizeof(msg\_size), MSG\_WAITALL,

(sockaddr\*)&server\_address, address\_size);

msg\_size = ntohl(msg\_size);

std::string response(msg\_size, '\0');

recvfrom(client\_fd, response.data(), msg\_size, MSG\_WAITALL,

(sockaddr\*)&server\_address, address\_size);

return response.empty() ? kServerUnavailabaleMsg : response;

}

private:

const int kPort = 0;

const std::string kIpAddress;

int client\_fd = 0; // socket information

sockaddr\_in server\_address{}; // server information

socklen\_t\* address\_size{};

};

int main() {

UDPClient client;

const std::string& kMessage = "Input line // If line == '!q'-> exit";

const std::string& kInputMsg = "Input: ";

const std::string& kQuitMsg = "!q";

std::cout << kMessage << std::endl;

while (true) {

std::string request;

std::cout << std::endl;

do {

std::cout << std::endl << kInputMsg;

std::getline(std::cin, request);

if (request.empty()) {

GO\_UP\_LINE();

GO\_UP\_LINE();

CLEAR\_LINE();

}

} while (request.empty());

if (request == kQuitMsg) {

std::cout << "Bye!" << std::endl;

break;

}

std::cout << client.getResponse(request) << std::endl;

}

return 0;

}

Код серверной части:

#include <iostream>

#include <cstring>

#include <cassert>

#include <unordered\_map>

#include <algorithm>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <arpa/inet.h>

class UDPServer { // UDP server

public:

UDPServer() {

socket\_fd = socket(AF\_INET, SOCK\_DGRAM, 0);

if (socket\_fd < 0) {

throw std::runtime\_error("Cant create socket descriptor.");

}

// setup server information

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

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

server\_address.sin\_port = htons(PORT);

// connect socket with port

if (bind(socket\_fd, reinterpret\_cast<sockaddr\*>(&server\_address), sizeof(server\_address)) < 0) {

throw std::runtime\_error("Cant connect socket with port.");

}

std::cout << "UDP server is running! Port: " << PORT << " IpAddress: localhost" << std::endl;

}

~UDPServer() {

close(socket\_fd);

}

std::string CreateResponse(const char\* buffer, const auto& kKeyWord) {

std::unordered\_map<int, int> symbol\_to\_cnt;

// count number of each sumbol

for (int i = 0; buffer[i]; ++i) {

auto symbol = std::tolower(buffer[i]);

if (kKeyWord.find(symbol) != std::string::npos) {

symbol\_to\_cnt[symbol] += 1;

}

}

// form response

bool is\_all\_contained = std::all\_of(kKeyWord.begin(), kKeyWord.end(), [&](auto& item) {

return symbol\_to\_cnt.contains(item);

});

std::string response{(is\_all\_contained ? "YES" : "NO")};

response += ' ';

for (auto& item : kKeyWord) {

if (symbol\_to\_cnt[item] != -1) {

response += item;

response += '|' + std::to\_string(symbol\_to\_cnt[item]) + ' ';

symbol\_to\_cnt[item] = -1; // indicate, that we already used this item

}

}

response.shrink\_to\_fit();

return response;

}

[[noreturn]] void run() {

/\*

TODO: 1. server accept size of data

2. allocate memory for data

3. recvfrom to accept data

4. from response

5. sendto(client\_socket)

\*/

uint32\_t msg\_size = 0;

const std::string kKeyWord{"windows"};

while (true) {

// Get requested message size

recvfrom(socket\_fd, &msg\_size, sizeof(msg\_size), 0,

(sockaddr\*)&client\_address, &address\_length);

msg\_size = ntohl(msg\_size);

char buffer[msg\_size + 1];

buffer[msg\_size] = '\0';

// read to buffer of size msg\_size data

recvfrom(socket\_fd, buffer, sizeof(buffer), 0,

(sockaddr\*)&client\_address, &address\_length);

std::string response(CreateResponse(buffer, kKeyWord));

// need to write data into udp-socket

// send to client size of the response msg

msg\_size = htonl(response.size());

assert(sendto(socket\_fd, &msg\_size, sizeof(msg\_size), 0,

(sockaddr\*)&client\_address, address\_length

) != -1);

sendto(socket\_fd, response.c\_str(), response.size(), 0,

(sockaddr\*)&client\_address, address\_length

);

ShowMessage(kRequestMsg, buffer, kResponseMsg, response);

}

}

private: // functions

template <class T>

void ShowMessage(const T& message) {

std::cout << message << std::endl;

}

template <class T, class... Args>

void ShowMessage(const T& message, Args... args) {

std::cout << message << ' ';

ShowMessage(args...);

}

private: // variables

const int PORT = 9984;

sockaddr\_in server\_address{}; // server information

sockaddr\_in client\_address{}; // client infromation

socklen\_t address\_length = sizeof(server\_address);

int socket\_fd = 0; // file descriptor

const std::string kRequestMsg{"Request:"};

const std::string kResponseMsg{"Response:"};

};

int main() {

try {

UDPServer server;

server.run();

} catch (std::runtime\_error& e) {

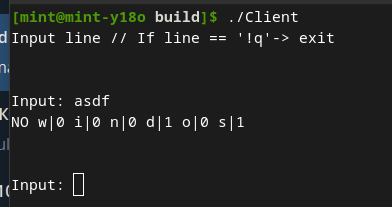
std::cerr << e.what() << std::endl;

return EXIT\_FAILURE;

}

}

Скриншот работы клиентской части:



Скриншот работы серверной части:

