Министерство образования Республики Беларусь

Учреждение образования Белорусский государственный университет информатики и радиоэлектроники

Факультет непрерывного и дистанционного обучения

**КОНТРОЛЬНАЯ РАБОТА №2**

По дисциплине

**КОМПЬЮТЕРНЫЕ СЕТИ**

По теме

**«Программирование протоколов сетевого и транспортного уровней. Создание последовательного сервера без установления логического соединения UDP»**

**Вариант № 7**

Выполнил: Галкин И.В.

Группа № 894351, 3 курс

№ зачетки 6535021

Минск 2020

**Постановка задачи**

Разработать приложение, реализующее архитектуру «клиент-сервер». Необходимо реализовать последовательный сервер без установления логического соединения (UDP). Логику взаимодействия клиента и сервера реализовать следующим образом:

7. Клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши "Ввод". Сервер, получив эту строку, должен определить длину введенной строки, и, если длина больше 15, то из нее удаляются все цифры. Клиент получает преобразованную строку и количество удаленных цифр.

**Краткое описание алгоритма**

При инициализации проектов, подключается библиотека WinSock2 посредством вызова WSAStartup и, в случае неудачного подключения, выводится сообщение об ошибке инициализации и проект завершает работу.

Далее, создается дескриптор сокета с заданным параметром SOCK\_DGRAM указывающий, что это UDP сокет и привязываем к нему объект sockaddr, содержащий параметры подключения с помощью команды bind.

Для объекта sockaddr указываем ip адрес сервера (sin\_addr), занимаемый порт (sin\_port) и то, что сокет работает по internetwork (sin\_family).

* 1. **Серверная часть**

Первым делом, для обработки запросов нужно запустить сервер (проект ServerSide).

После начальной инициализации, сервер, с уже открытым сокетом, начинает ожидать сообщения на него через вызов recvfrom. В случае, если произошла ошибка получения сообщения, в консоль записывается соответствующее сообщение с кодом ошибки. Если же сообщение получено успешно, то сервер принимает строку, полученную от клиента, и начинает обработку.

Если длина строки не превышает 15 символов, то клиент получит точно такую же строку в ответ, с добавочной информацией, что было удалено 0 цифр. Если же длина превышает указанного максимума, то сервер удаляет все цифры из строки и возвращает отформатированную строку с количеством удаленных цифр.

Для отправки ответа клиенту используется вызов sendto с указанием сокета и передачей отформатированной строки. В случае ошибки отправки ответа, в консоль пишется соответствующее сообщение с кодом ошибки.

Весь алгоритм принятия и отправки сообщений обернут в бесконечный цикл для того, чтобы сервер не прекращал работу после обработки первого же запроса.

* 1. **Клиентская часть**

В целом, логика работы клиента (проект ClientSide) схожа с серверной.

Первым делом идет инициализация UDP сокета. После этого, пользователю предлагается ввести строку, которая будет отправлена на сервер.

Далее, введенная строка отправляется на сервер с помощью вызова sendto с указанием сокета и строки.

Ответ от сервера ожидается при помощи вызова recvfrom и, в случае успешного получения, выводится отформатированная сервером строка.

Весь алгоритм также обернут в бесконечный цикл для неоднократного использования.

**Листинги с пояснениями**

**Серверная часть:**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

#include <iostream>

#include <string>

#pragma comment(lib,"Ws2\_32.lib")

using namespace std;

void main()

{

WSADATA wsaData;

const int maxStrLength = 15;

// NOTE: Try to initialize WinSock library

int initializeOutput = WSAStartup(MAKEWORD(2, 2), &wsaData);

// NOTE: Intialize errors handling

if (initializeOutput != 0) {

printf("WSAStartup failed with error: %d\n", initializeOutput);

return;

}

// NOTE: Define socket with UDP protocol

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

if (connectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

return;

}

// Note: Socket params

struct sockaddr\_in ad;

ad.sin\_family = AF\_INET;

ad.sin\_port = htons(1024);

ad.sin\_addr.s\_addr = 0;

int size = sizeof(ad);

// NOTE: Bind socket params to socket descriptor

bind(connectSocket, (struct sockaddr\*)&ad, sizeof(ad));

const int requestBufferLen = 1024;

char requestBuffer[requestBufferLen];

const int responseLen = 1124;

char responseBuffer[responseLen];

// NOTE: Listening for new messages

while (1)

{

printf("Waiting for data...\n");

fflush(stdout);

//NOTE: Clear the buffer by filling null, it might have previously received data

memset(requestBuffer, '\0', requestBufferLen);

memset(responseBuffer, '\0', responseLen);

// NOTE: Try to receive some data

int request = recvfrom(connectSocket, requestBuffer, requestBufferLen, 0, (sockaddr\*)&ad, &size);

// NOTE: Handle error

if (request == SOCKET\_ERROR) {

int iError = WSAGetLastError();

if (iError == WSAEWOULDBLOCK)

printf("recv failed with error : WSAEWOULDBLOCK\n");

else

printf("recv failed with error code: %ld\n", iError);

continue;

}

requestBuffer[request] = '\0';

//NOTE: Print details of the request

printf("Received packet from %s:%d\n", inet\_ntoa(ad.sin\_addr), ntohs(ad.sin\_port));

printf("Data: %s\n", requestBuffer);

// NOTE: Process request

int digitsRemoved = 0;

if (strlen(requestBuffer) > maxStrLength) {

string result;

for (int i = 0; i < strlen(requestBuffer); i++)

{

if (isdigit(requestBuffer[i])) {

digitsRemoved++;

} else {

result += requestBuffer[i];

}

}

strcpy\_s(responseBuffer, result.c\_str());

} else {

strcpy\_s(responseBuffer, requestBuffer);

}

// NOTE: Prepare response

sprintf\_s(responseBuffer, "%s%s%d", responseBuffer, ", digits removed: ", digitsRemoved);

// NOTE: Send response

int responseSendingStatus = sendto(connectSocket, responseBuffer, responseLen, 0, (sockaddr\*)&ad, size);

if (responseSendingStatus == SOCKET\_ERROR) {

printf("Sendto failed with error code : %ld \n", WSAGetLastError());

}

else {

printf("Response: %s\n\n", responseBuffer);

}

}

// NOTE: Close socket

closesocket(connectSocket);

// NOTE: Dispose WinSock lib

WSACleanup();

}

**Клиентская часть:**

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <winsock2.h>

#include <iostream>

#pragma comment(lib,"Ws2\_32.lib")

using namespace std;

void main()

{

WSADATA wsaData;

struct sockaddr\_in ad;

// NOTE: Try to initialize WinSock library

int initializeOutput = WSAStartup(MAKEWORD(2, 2), &wsaData);

// NOTE: Intialize errors handling

if (initializeOutput != 0) {

printf("WSAStartup failed with error: %d\n", initializeOutput);

exit(EXIT\_FAILURE);

}

// NOTE: Define socket with UDP protocol

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

if (connectSocket < 0) {

printf("socket creation failed");

exit(EXIT\_FAILURE);

}

// Note: Socket params

ad.sin\_family = AF\_INET;

ad.sin\_port = htons(1024);

ad.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

int size = sizeof(ad);

// NOTE: UI

cout << "This program will remove digits from your input if the length of typed string exceeds 15 characters" << endl;

const int bufferLen = 1125;

char buffer[bufferLen];

while (1)

{

fflush(stdout);

//NOTE: Clear the buffer by filling null, it might have previously received data

memset(buffer, '\0', bufferLen);

cout << "\nPlease, enter the string:" << endl;

cin.getline(buffer, bufferLen, '\n');

// NOTE: Send request

int sendingStatus = sendto(connectSocket, buffer, strlen(buffer), 0, (sockaddr\*)&ad, size);

if(sendingStatus == SOCKET\_ERROR) {

printf("Sendto failed with error code : %ld \n", WSAGetLastError());

continue;

}

// NOTE: Receive response

int response = recvfrom(connectSocket, buffer, bufferLen, 0, (sockaddr\*)&ad, &size);

if (response == SOCKET\_ERROR) {

int iError = WSAGetLastError();

if (iError == WSAEWOULDBLOCK)

printf("recv failed with error : WSAEWOULDBLOCK\n");

else

printf("recv failed with error code: %ld\n", iError);

continue;

}

buffer[response] = '\0';

cout << endl << "Server response : " << buffer << endl;

}

// NOTE: Close socket

closesocket(connectSocket);

// NOTE: Dispose WinSock lib

WSACleanup();

}

**Результаты тестирования**

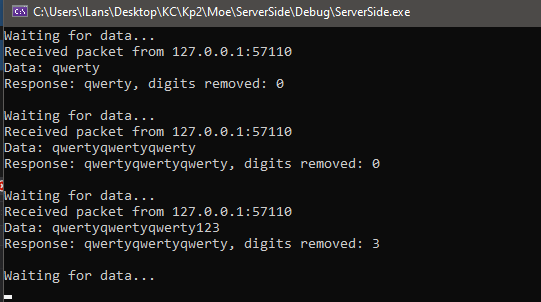


Рисунок 1 – Пример работы сервера

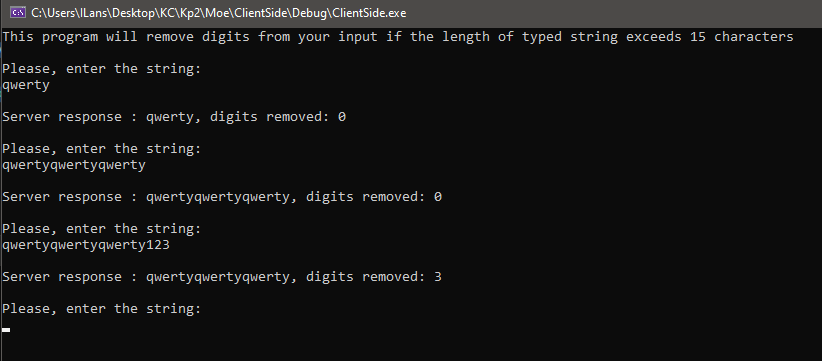


Рисунок 2 – Пример работы клиента

**Контрольные вопросы**

1. **Что содержит UDP-сообщение помимо посылаемых данных?**

Помимо посылаемых данных, UDP сообщение содержит заголовок, который состоит из 4 полей, каждое по 2 байта.

1. Порт отправителя
2. Порт получателя
3. Длина датаграммы
4. Контрольная сумма
5. **Что называется дейтаграммой?**

При работе на хосте-отправителе данные от приложений поступают протоколу UDP через порт в виде сообщений. Протокол UDP добавляет к каждому отдельному сообщению свой 8-байтный заголовок, формируя из этих сообщений собственные протокольные единицы, называемые UDP-дейтаграммами, и передает их нижележащему протоколу IP. В этом и заключаются его функции по мультиплексированию данных.

Каждая дейтаграмма переносит отдельное пользовательское сообщение. Сообщения могут иметь различную длину, не превышающую, однако длину поля данных протокола IP, которое, в свою очередь, ограничено размером кадра технологии нижнего уровня. Поэтому если буфер UDP переполняется, то сообщение приложения отбрасывается.

1. **Какие возможности не предоставляет UDP (в отличие от TCP)?**

* TCP гарантирует доставку пакетов данных в неизменных виде, последовательности и без потерь, UDP ничего не гарантирует.
* TCP нумерует пакеты при передаче, а UDP нет
* TCP работает в дуплексном режиме, в одном пакете можно отправлять информацию и подтверждать получение предыдущего пакета.
* TCP требует заранее установленного соединения, UDP соединения не требует, у него это просто поток данных.
* UDP обеспечивает более высокую скорость передачи данных.
* TCP надежнее и осуществляет контроль над процессом обмена данными.
* UDP предпочтительнее для программ, воспроизводящих потоковое видео, видеофонии и телефонии, сетевых игр.
* UPD не содержит функций восстановления данных

1. **Чем функции sendto() и recvfrom() отличаются от функций send() recv()?**

Эти функции отправляют данные в сокет. send () используется для сокетов, подключенных по протоколу TCP SOCK\_STREAM, а sendto () используется для неподключенных сокетов UDP SOCK\_DGRAM. С неподключенными сокетами необходимо указывать место назначения пакета каждый раз, когда идет отправка, и поэтому последние параметры sendto () определяют, куда идет пакет.

1. **Что происходит, если размер буфера, переданный в функцию recvfrom(), слишком мал для приема всей дейтаграммы целиком?**

Если размер буфера, переданный в функцию recvfrom(), слишком мал для приема всей дейтаграммы целиком, буфер заполняется теми данными, которые в него помещаются, а оставшаяся часть дейтаграммы сбрасывается в подсистему сборки мусора и пропадает безвозвратно. В этом случае функция recvfrom() возвращает значение socket error.

1. **Приведите примеры сетевых приложений, использующих UDP**

Многочисленные ключевые Интернет-приложения используют UDP, в их числе — DNS (где запросы должны быть быстрыми и состоять только из одного запроса, за которым следует один пакет ответа), Простой Протокол Управления Сетями (SNMP), Протокол Маршрутной Информации (RIP), Протокол Динамической Конфигурации Узла (DHCP).

Голосовой и видеотрафик обычно передается с помощью UDP. Протоколы потокового видео в реальном времени и аудио разработаны для обработки случайных потерь пакетов так, что качество лишь незначительно уменьшается вместо больших задержек при повторной передаче потерянных пакетов.

1. **В каких случаях применение UDP протокола может быть предпочтительней, чем TCP?**

UDP доставляет пакеты данных (датаграммы) гораздо быстрее, потому для приложений, которые рассчитаны на широкую пропускную способность и быстрый обмен, UDP можно считать оптимальным протоколом. К таковым относятся сетевые и браузерные игры, а также программы просмотра потокового видео и приложения для видеосвязи (или голосовой): от потери пакета, полной или частичной, ничего не меняется, повторять запрос не обязательно, зато загрузка происходит намного быстрее.