**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Сети ЭВМ и телекоммуникации**

**Тема: Программирование протокола TCP/UDP с использованием**

**библиотеки Winsock**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: ст. пр. ПО и ВТАС

Федотов Евгений Александрович

**Белгород 2020**

**Цель работы:** изучить протоколы TCP/UDP, основные функции библиотеки Winsock и составить программу для приема/передачи пакетов

**Краткие теоретические сведения**

Transmission Control Protocol (TCP) (протокол управления передачей) - один из основных сетевых протоколов Интернета, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Выполняет функции протокола транспортного уровня модели OSI. TCP - это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи. Реализация TCP, как правило, встроена в ядро ОС, хотя есть и реализации TCP в контексте приложения. Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом

компьютере. Программы для электронной почты и обмена файлами используют TCP. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

TCP протокол базируется на IP для доставки пакетов, но добавляются две важные вещи: во первых устанавливается соединение - это позволяет ему, в отличие от IP, гарантировать доставку пакетов, во-вторых - используются порты для обмена пакетами между приложениями, а не просто узлами.

Протокол TCP предназначен для обмена данными — это «надежный» протокол, потому что он во-первых обеспечивает надежную доставку данных, так как предусматривает установления логического соединения; во-вторых, нумерует пакеты и подтверждает их прием квитанцией, а в случае потери организует повторную передачу; в-третьих, делит передаваемый поток байтов на части — сегменты - и передает их нижнему уровню, на приемной стороне снова собирает их в непрерывный поток байтов.

TCP соединение начинается с т.н. “рукопожатия”: узел A посылает узлу B специальный пакет SYN — приглашение к соединению; B отвечает пакетом SYN-ACK — согласием об установлении

соединения; A посылает пакет ACK — подтверждение, что согласие получено. После этого TCP соединение считается установленным, и приложения, работающие в этих узлах, могут посылать друг другу пакеты с данными. «Соединение» означает, что узлы помнят друг о друге, нумеруют

все пакеты, идущие в обе стороны, посылают подтверждения о получении каждого пакета и перепосылают потерявшиеся по дороге пакеты. Для узла A это соединение называется исходящим, а для узла B - входящим. Любое установленное TCP соединение симметрично, и

пакеты с данными по нему всегда идут в обе стороны. В отличие от традиционной альтернативы - UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения,

которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

1. Установка соединения.

2. Передача данных.

3. Завершение соединения

**Протокол UDP**

Протокол UDP (User Datagram Protocol) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP ("бессвязный" протокол). К заголовку IP-пакета UDP добавляет поля порт отправителя и порт получателя, которые обеспечивают мультиплексирование информации между различными прикладными процессами, а также поля длина UDP-дейтограммы и контрольная

сумма, позволяющие поддерживать целостность данных. Таким образом, если на уровне IP для определения места доставки пакета используется адрес, на уровне UDP - номер порта.

Протокол UDP ориентирован на транзакции, получение датаграмм и защита от дублирования не гарантированы. Приложения, требующие гарантированного получения потоков данных, должны использовать протокол управления пересылкой (Transmission Control Protocol - TCP).

UDP - минимальный ориентированный на обработку сообщений протокол транспортного уровня, задокументированный в RFC 768.

UDP не предоставляет никаких гарантий доставки сообщения для протокола верхнего уровня и не сохраняет состояния отправленных сообщений.

UDP обеспечивает многоканальную передачу (с помощью номеров портов) и проверку целостности (с помощью контрольных сумм) заголовка и существенных данных. Надежная передача в случае необходимости должна реализовываться пользовательским приложением.

**Задание к работе**

1. Разработать программу “Сервер” (на языке программирования Pascal или C), которая принимает запросы от клиентов и посылает им в качестве ответа некоторое сообщение.

2. Разработать программу “Клиент” (на языке программирования Pascal или C), которая посылает запрос серверу и “ждет” от него ответного сообщения.

3. Провести анализ функционирования разработанных программ (одновременная работа 2-х, 3-х и т.д. приложений на 2-х, 3-х и т.д. компьютерах ЛВС), сделать выводы.

4. Провести сравнительный анализ протоколов TCP и UDP. Сделать выводы.

**Анализ функционирования разработанной программы.**

|  |  |
| --- | --- |
| **TCP** | **UDP** |
| 1.198 (100% исх. изображения) | 0.068(<5% передано без задержки) |
|  | 0.218(~65-80% передано с задержкой 1 микросекунда) |
|  | 0.917(~99% передано с задержкой 3 микросекунды) |

|  |  |
| --- | --- |
| 2020-03-13_00-14-40 | image |
| image |  |

При передаче изображения через TCP без задержек, всё передаётся отлично без искажений. При передаче изображения через UDP происходят искажения и причём чем меньше задержка, тем очевидно больше потерь при передаче.

*Приложение*

**Server.c**

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <stdio.h>

#include <windows.h>

#include <windowsx.h>

#include <commctrl.h>

#include <winsock.h>

#include <cstdlib>

#include <algorithm>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <iostream>

#include <chrono>

#define MAX 65000

#define BUFLEN 65000

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

using namespace std;

#define PORT 8888

int main()

{

SOCKET s;

struct sockaddr\_in server, si\_other;

int slen, recv\_len;

char buf[BUFLEN];

WSADATA wsa;

bool timerstarted = false;

int batchsize = MAX;

FILE\* stream;

int logging = 0, Err, i, nulls = 0, val;

fopen\_s(&stream, "c:\\a\\image.jpg", "wb");

slen = sizeof(si\_other);

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)

{

printf("Failed. Error Code : %d", WSAGetLastError());

exit(EXIT\_FAILURE);

}

printf("Initialised.\n");

clock\_t begin\_time;

if ((s = socket(AF\_INET, SOCK\_DGRAM, 0)) == INVALID\_SOCKET)

{

printf("Could not create socket : %d", WSAGetLastError());

}

printf("Socket created.\n");

server.sin\_family = AF\_INET;

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

server.sin\_port = htons(PORT);

if (bind(s, (struct sockaddr\*) & server, sizeof(server)) == SOCKET\_ERROR)

{

printf("Bind failed with error code : %d", WSAGetLastError());

exit(0);

}

puts("Bind done");

while (nulls != batchsize)

{

nulls = 0;

memset(buf, 0, BUFLEN);

if ((recv\_len = recvfrom(s,buf,BUFLEN,0,(struct sockaddr\*)&si\_other,&slen)) == SOCKET\_ERROR)

{

printf("recvfrom() failed with error code : %d", WSAGetLastError());

exit(0);

}

if (!timerstarted) {

timerstarted = true;

begin\_time = clock();

}

for (i = 0; i < batchsize; i++) if (buf[i] == 0) nulls++;

if (nulls != batchsize) fwrite(buf, batchsize, 1, stream);

}

closesocket(s);

WSACleanup();

std::cout << float(clock() - begin\_time) / CLOCKS\_PER\_SEC;

return 0;

}

**Client.c**

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <stdio.h>

#include <windows.h>

#include <windowsx.h>

#include <commctrl.h>

#include <winsock.h>

#include <cstdlib>

#include <algorithm>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <iostream>

#include <chrono>

#include <thread>

#define MAX 65000

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

#define BUFLEN 65000

#define PORT 8888

#define SERVER "127.0.0.1"

using namespace std;

int main(void)

{

struct sockaddr\_in si\_other;

int s, slen = sizeof(si\_other);

char buf[BUFLEN];

unsigned char buffer[BUFLEN];

WSADATA wsa;

printf(" \nInitialising Winsock... ");

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)

{

printf(" Failed.Error Code : %d" , WSAGetLastError());

exit(EXIT\_FAILURE);

}

printf("\nInitialised.\n ");

if ((s = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP)) == SOCKET\_ERROR)

{

printf(" socket() failed with error code : %d", WSAGetLastError());

exit(EXIT\_FAILURE);

}

memset((char\*)&si\_other, 0, sizeof(si\_other));

si\_other.sin\_family = AF\_INET;

si\_other.sin\_port = htons(PORT);

si\_other.sin\_addr.S\_un.S\_addr = inet\_addr(SERVER);

int i;

int logging = 0;

FILE\* ptrFile;

char filename[256] = "c:\\a\\a.jpg";

int timedelay = 0;

long last = 0, lSize;

size\_t result = 1;

long sizesended = 0;

char SERVERADDR[256] = "localhost";

fopen\_s(&ptrFile, filename, "rb+");

printf("\n");

fseek(ptrFile, 0L, SEEK\_END);

lSize = ftell(ptrFile);

printf("%i", lSize);

rewind(ptrFile);

while (last < lSize)

{

memset((char\*)&buffer, 0, sizeof(buffer));

result = fread(buffer, sizeof(buffer), 1, ptrFile);

if (sendto(s, (char\*)buffer, sizeof(buffer), 0, (struct sockaddr\*) & si\_other, slen) == SOCKET\_ERROR)

{

printf("sendto() failed with error code : %d", WSAGetLastError());

exit(EXIT\_FAILURE);

}

sizesended += sizeof(buffer);

last = ftell(ptrFile);

}

memset((char\*)&buffer, 0, sizeof(buffer));

std::this\_thread::sleep\_for(std::chrono::milliseconds(10));

sendto(s, (char\*)buffer, sizeof(buffer), 0, (struct sockaddr\*) & si\_other, slen);

closesocket(s);

WSACleanup();

return 0;

}

**Тексты программ (TCP)**

**Server.c**

#include <stdio.h>

#include <string.h>

#include <windows.h>

#include <windowsx.h>

#include <commctrl.h>

#include <winsock.h>

#include <cstdlib>

#include <algorithm>

#include <stdlib.h>

#include <sys/types.h>

#include <iostream>

#include <chrono>

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

#define MAX 65000

int main()

{

char SERVERADDR[256] = "localhost";

clock\_t begin\_time;

bool timerstarted = false;

WSADATA wsaData = { 0 };

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

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

if (s < 0)

{

perror("Error calling socket");

return 0;

}

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(5150);

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(s, (struct sockaddr\*) & addr, sizeof(addr)))

{

perror("Error calling bind");

return 0;

}

int error = 0;

if( error = listen(s, 10) )

{

perror("Error calling listen");

return 0;

}

int s1 = accept(s, NULL, NULL);

if( s1 < 0 )

{

perror("Error calling accept");

return 0;

}

unsigned char buffer[MAX];

int batchsize = MAX;

FILE\* stream;

int logging = 0, Err=0, i, nulls = 0, val;

fopen\_s(&stream, "c:\\a\\image.jpg", "wb");

while (nulls != batchsize)

{

nulls = 0;

int len = 0, server\_addr\_size = sizeof(addr);

int rc = recv(s1, (char\*)&buffer, MAX, 0);

if (!timerstarted) {

timerstarted = true;

begin\_time = clock();

}

if (Err != SOCKET\_ERROR)

{

for (i = 0; i < batchsize; i++)

{

if (buffer[i] == 0) nulls++;

}

if (nulls != batchsize)

{

fwrite(buffer, batchsize, 1, stream);

}

}

else {

printf("Error %i!\n", Err);

}

}

std::cout << float(clock() - begin\_time) / CLOCKS\_PER\_SEC;

printf("\nDownload finished!");

fclose(stream);

return 0;

}

**Client.c**

#include <stdio.h>

#include <string.h>

#include <dos.h>

#include <stdio.h>

#include <windows.h>

#include <windowsx.h>

#include <commctrl.h>

#include <winsock.h>

#include <chrono>

#include <thread>

#include <stdlib.h>

#include <sys/types.h>

#include <errno.h>

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

#define bzero(b,len) (memset((b), '\0', (len)), (void) 0)

#define MAX 65000

void func(int\* sockfd)

{

unsigned char buffer[MAX] = { 0 };

int n, i;

int logging = 0;

FILE\* ptrFile;

char filename[256] = "c:\\a\\a.jpg";

int timedelay = 0;

long last = 0, lSize;

size\_t result = 1;

long sizesended = 0;

char SERVERADDR[256] = "localhost";

fopen\_s(&ptrFile, filename, "rb+");

printf("\n");

fseek(ptrFile, 0L, SEEK\_END);

lSize = ftell(ptrFile);

printf("%i", lSize);

rewind(ptrFile);

struct sockaddr\_in servaddr;

while (last < lSize)

{

bzero(buffer, sizeof(buffer));

result = fread(buffer, sizeof(buffer), 1, ptrFile);

memset(&servaddr, 0, sizeof(servaddr));

int sended = send(\*sockfd, (char\*)buffer, sizeof(buffer),0);

sizesended += sizeof(buffer);

last = ftell(ptrFile);

}

bzero(buffer, sizeof(buffer));

std::this\_thread::sleep\_for(std::chrono::milliseconds(500));

send(\*sockfd, (char\*)buffer, sizeof(buffer),0);

printf("\nMessage has been sended\n");

}

int main()

{

WSADATA wsaData = { 0 };

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

int sockfd, connfd;

struct sockaddr\_in servaddr, cli;

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

if (sockfd == -1) {

printf("socket creation failed...\n");

exit(0);

}

else

printf("Socket successfully created..\n");

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

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

servaddr.sin\_port = htons(5150);

if (connect(sockfd, (struct sockaddr\*)&servaddr, sizeof(servaddr)) != 0) {

printf("connection with the server failed...\n");

exit(0);

}

else

printf("connected to the server..\n");

func(&sockfd);

return 0;

}

**Результат работы программы**

|  |  |
| --- | --- |
| Клиент | Сервер |
|  | |