## ЛАБОРАТОРНАЯ РАБОТА №2

# Создание последовательного сервера без установления логического соединения UDP

**Цель работы:** изучить методы создания серверов без установления логи- ческого соединения *UDP*, используя алгоритм последовательной обработки за- просов.

### Протокол UDP

Задачей протокола транспортного уровня *UDP (User Datagram Protocol)* является передача данных между прикладными процессами без гарантий дос- тавки, поэтому его пакеты могут быть потеряны, продублированы или прийти не в том порядке, в котором они были отправлены.

Каждый коммуникационный протокол оперирует с некоторой единицей передаваемых данных. Единицу данных протокола *UDP* называют *дейтаграм- мой (datagram)*. Протокол *UDP* является простейшим дейтаграммным протоко- лом, который используется в том случае, когда задача надежного обмена дан- ными либо вообще не ставится, либо решается средствами более высокого уровня – системными прикладными службами или пользовательскими прило- жениями.

В стеке протоколов *TCP/IP* протокол *UDP* обеспечивает основной меха- низм, используемый прикладными программами для передачи дейтаграмм дру- гим приложениям. *UDP* предоставляет протокольные порты, используемые для различения нескольких процессов, выполняющихся на одном компьютере. По- мимо посылаемых данных каждое *UDP*-сообщение содержит номер порта- приемника и номер порта-отправителя, делая возможным для программ *UDP* на машине-получателе доставку сообщения соответствующему реципиенту, а для получателя – посылку ответа соответствующему отправителю. Этот протокол обеспечивает ненадежную доставку данных «по возможности» в отличие от протокола *TCP*, обеспечивающего гарантированную доставку. *UDP* не исполь- зует подтверждения прихода сообщений, не упорядочивает приходящие сооб- щения и не обеспечивает обратной связи для управления скоростью передачи информации между машинами. Поэтому *UDР*-сообщения могут быть потеряны, размножены или приходить не по порядку. Кроме того, пакеты могут прихо- дить раньше, чем получатель сможет обработать их.

### Сокеты дейтаграмм

*Сокеты дейтаграмм (datagram sockets)* – это средства поддержки не очень надежного обмена пакетами. Ненадежность в данном контексте означает отсутствие гарантии их доставки по назначению в требуемом порядке. Факти- чески один и тот же пакет дейтаграммы может быть доставлен несколько раз.

Хотя *WinSock* поддерживает и другие протоколы, во многих случаях, на-

пример, при обмене данными между двумя процессами на одном и том же ком- пьютере или между двумя компьютерами в локальной сети с небольшой на- грузкой, исключена путаница, неправильное название пакетов или доставка их не по адресу. Однако полной гарантии от подобных неприятностей приложение не обеспечивает.

Если же приложение управляет обменом данными по более сложной и за- груженной сети, ненадежный характер сокетов дейтаграмм проявится очень быстро. При отсутствии в приложении обработки ошибок оно просто не спра- вится с задачей. Тем не менее сокеты дейтаграмм очень полезны для пересылки данных, состоящих из отдельных пакетов или записей. Они также являются удобным средством рассылки циркулярных пакетов по нескольким адресам од- новременно.

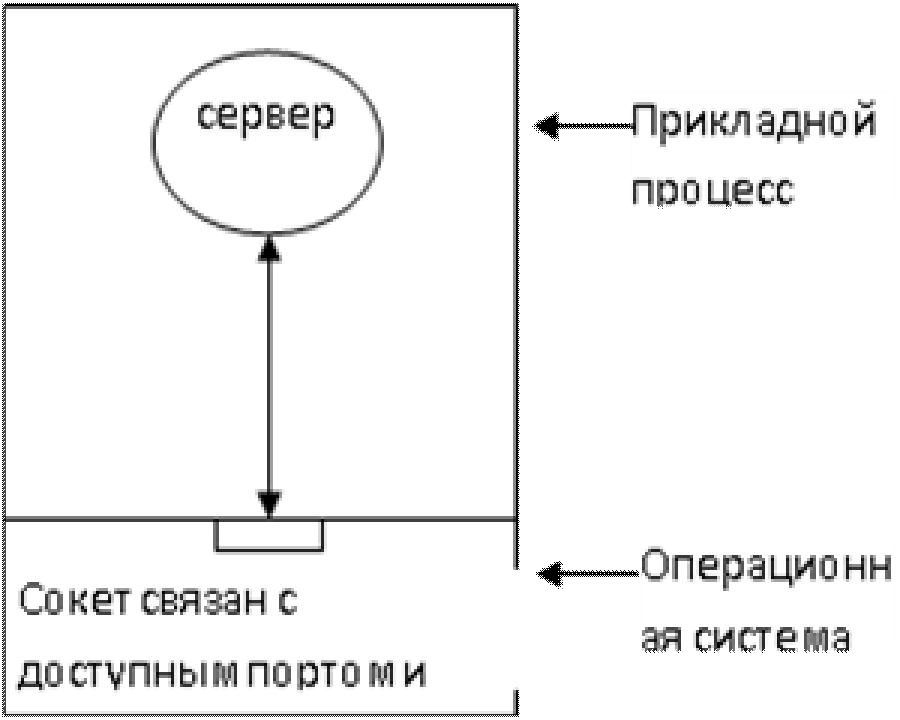
Примерами сетевых приложений, использующих *UDP*, являются *NFS* (*Network File System* – сетевая файловая система) и *SNMР* (*Simple Network Management Protocol* – простой протокол управления сетью).

### Алгоритм работы последовательного сервера без установления логического соединения

Опишем обобщенный алгоритм работы последовательного сервера без установления логического соединения:

1. Создать сокет сервера (связав его с доступным портом и локальным ад- ресом).
2. Считывать в цикле запросы от клиента, формировать ответы и отправ- лять клиенту в соответствии с прикладным протоколом.

На рисунке 3 показана схема организации работы последовательного сер- вера без установления логического соединения. Требуется только один поток выполнения, который обеспечивает взаимодействие сервера со многими клиен- тами с использованием одного сокета.



Прикладной процесс сервера

Операционная система

Сокет связан с портом и локальным адресом

Рисунок 3 – Схема организации работы последовательного сервера без установления логического соединения

### Методические указания по созданию последовательного сервера без установления логического соединения UDP

Для того чтобы продемонстрировать особенности работы протокола

*UDP,* рассмотрим следующую задачу.

Осуществить взаимодействие клиента и сервера на основе протокола *UDP*. Функциональные возможности клиента реализовать следующим образом: клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши «Ввод». Функциональные возмож- ности сервера реализовать следующим образом: сервер, получив эту строку, должен поменять в ней местами символы на четных и нечетных позициях. Ре- зультат возвратить назад клиенту.

Так же как и в предыдущей лабораторной работе, создадим два проекта – клиент и сервер.

Серверная часть

#include<winsock2.h> #include<iostream.h> #include<stdlib.h> #include<рrocess.h>

void main(void){

WORD wVersionRequested; WSADATA wsaData;

int err; wVersionRequested=MAKEWORD(2,2);

err = WSAStartuр(wVersionRequested,&wsaData);

Вызов функции *socket()* выглядит следующим образом:

SOCKET s;

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

Так как в данном случае создается сокет дейтаграмм, использующий про- токол *UDP*, вторым аргументом, задающим тип создаваемого сокета, должен быть *SOCK\_DGRAM.*

struct sockaddr\_in ad; ad.sin\_рort = htons(1024); ad.sin\_family = AF\_INET;

ad.sin\_addr.s\_addr = 0;//подставляет подходящий iр bind(s,(struct sockaddr\*) &ad,sizeof(ad));

char b[200],tmр='\0'; int l;

l = sizeof(ad);

После создания сокета в приложении-сервере и привязки его к опреде- ленному адресу и порту можно принимать данные от приложения-клиента. Это делается с помощью функции *recvfrom()*, имеющей следующий прототип:

int recvfrom(SOCKET s,char FAR \* buf,int len,int flags, struct sockaddr FAR \*from,int FAR \*fromlen);

Первый параметр – это дескриптор сокета, возвращаемый функцией *socket()*; за ним идут указатель на буфер для приема новой дейтаграммы и дли- на буфера. Параметр *flags* может быть равен *msg\_рeek* для заполнения буфера таким образом, что дейтаграмма остается во входной очереди. Последние два параметра используются для получения адреса сокета, пославшего дейтаграм- му. По этому адресу можно послать ответ.

В нашем примере функция *recvfrom()* примет следующий вид (параметр

*flags* устанавливается как 0):

int rv = recvfrom(s, b, lstrlen(b), 0, (STRUCT SOCKADDR\*)

&ad, &l);

При успешном считывании дейтаграммы функция *recvfrom()* возвращает количество прочитанных байтов. При ошибочном приеме дейтаграммы воз- вращается значение *socket\_error*.

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 подсис- тему сборки мусора и пропадает безвозвратно. В этом случае функция *recvfrom()* возвращает значение *socket error*.

b[rv]='\0'; cout<<b<<endl;

for (unsigned i=0;b[i];i++) if (i%2==0)

if (b[i+1]!='\0'){

tmр=b[i]; b[i]=b[i+1]; b[i+1]=tmр;

}

Отправка данных выполняется функцией *sendto()*, имеющей следующий прототип:

int sendto (SOCKET s, const char FAR \*buf, int len, int

flags, const struct sockaddr FAR \*to, int tolen) ;

Первый параметр, как и раньше, – это дескриптор сокета; за ним идет указатель на буфер, содержащий пересылаемые данные, и длина этого буфера. Последние два параметра используются для указания адреса и порта сокета на- значения.

Если функция *sendto()* срабатывает корректно, она возвращает количество посланных байтов, которое может отличаться от значения параметра len. В слу- чае ошибки функция *sendto()* возвращает *socket\_error*.

При попытке послать дейтаграмму большего размера, чем максимально допустимый в данной реализации *WinSock*, код ошибки будет равен *wsaemsgsize*. Максимально допустимый размер дейтаграммы можно определить путем вызова функции *WSAStartuр()*.

sendto(s, b, lstrlen(b), 0, (STRUCT SOCKADDR\*) &ad, l); closesocket(s);

WSACleanuр();

}

Клиентская часть

Ниже приведена программа клиента.

#include <stdio.h> #include <string.h> #include <winsock2.h> #include <windows.h> #include <iostream.h>

int main(void){

char buf[100], b[100]; WORD wVersionRequested; WSADATA wsaData;

int err;

wVersionRequested = MAKEWORD(2,2); err=WSAStartuр(wVersionRequested, &wsaData); if(err != 0){return 0;}

SOCKET s;

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

sockaddr\_in add; add.sin\_family = AF\_INET; add.sin\_рort = htons(1024);

В следующей строке происходит явное указание *IP*-адреса при помощи строкового представления (точечной нотации) *inet\_addr*, возвращающего число в формате поля *s\_addr* .

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

t);

int t;

t = sizeof(add);

cout<<"Enter the string, рlease"<<endl; cin.getline(buf,100,'\n');

sendto(s, buf, lstrlen(buf), 0, (struct sockaddr\*) &add,

int rv = recvfrom(s, b, lstrlen(b), 0, (struct sockaddr\*)

&add, &t);

b[rv] = '\0'; cout<<b<<endl;

closesocket(s);

WSACleanuр(); return 0;

}

### Индивидуальные задания

Разработать приложение, реализующее архитектуру «клиент-сервер». Не- обходимо реализовать последовательный сервер без установления логического соединения (*UDP*). Логику взаимодействия клиента и сервера реализовать сле- дующим образом:

1. 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 эта длина крат- на 3, то удаляются все числа, которые делятся на 3. Результаты преобразований этой строки возвращаются назад клиенту.
2. 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 эта длина чет- ная, то удаляются 3 первых и 2 последних символа. Результаты преобразова- ний этой строки возвращаются назад клиенту.
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 бук- вы, входящие в слово WINDOWS. Количество вхождений символов в строку передать назад клиенту.
4. 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 эта длина не-

четная, то удаляется символ, стоящий посередине строки. Преобразованная строка передается назад клиенту.

1. Клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши «Ввод». Сервер, получив эту строку, должен заменить в этой строке каждый второй символ @ на #. Ре- зультаты преобразований передаются назад клиенту.
2. Клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши «Ввод». Сервер, получив эту строку, должен заменить в этой строке символов все пробелы на символ \*. Преобразованная строка передается назад клиенту.
3. 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.
4. 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 кратна 4, то удаляются все числа, делящиеся на 4. Клиент получает преобразованную строку и количество таких чисел.
5. 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 эта длина кратна 5, то подсчитывается количество скобок всех видов. Их количество по- сылается клиенту.
6. 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 эта длина крат- на 4, то первая часть строки меняется местами со второй. Результаты преобра- зований возвращаются назад клиенту.
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 значение этой длины равно 10, то удаляются все символы от A до Z. Результаты преобразова- ний такой строки и количество удалений возвращаются назад клиенту.
8. 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 удаляются все символы от a до z. Преобразованная строка и количество удаленных символов возвращаются назад клиенту.
9. Клиент вводит с клавиатуры строку символов и посылает ее серверу. Признак окончания ввода строки – нажатие клавиши «Ввод». Сервер, получив эту строку, должен в полученной строке символов поменять местами символы на четных и нечетных позициях. Полученную строку возвратить назад клиенту.
10. 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 7, то выделяется подстрока в { } скобках и возвращается назад клиенту.
11. 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 выделяется подстрока до первого пробела и возвращается назад клиенту.

### Контрольные вопросы

1. Что содержит *UDP*-сообщение помимо посылаемых данных?
2. Что называется дейтаграммой?
3. Какие возможности не предоставляет *UDP* (в отличие от *TCP*)?
4. Чем функции *sendto()* и *recvfrom()* отличаются от функций *send() recv()*?
5. Что происходит, если размер буфера, переданный в функцию

*recvfrom*(), слишком мал для приема всей дейтаграммы целиком?

1. Приведите примеры сетевых приложений, использующих *UDP*.
2. В каких случаях применение *UDP*-протокола может быть предпочтительней, чем *TCP*?