Лабораторная работа №2

Теоретические сведения

**Основы программирования TCP/IP**

Большинство сетевых приложений строится по схеме «клиент-сервер».



Рис.

Основные понятия программного интерфейса ТСР/IР

В IP-сетях адрес любого приложения сети (локальный адрес), состоит из трех элементов :

1. сетевой адрес (IP-адрес или сетевое имя), узла сети на котором работает приложение;
2. идентификатор протокола(для стека TCP/IP- это TCP или UDP);
3. номер порта – идентификатор приложения в рамках этого узла.

Socket-интерфейс представляет собой набор системных вызовов и/или библиотечных функций соответствующего языка программирования.

Для работы с «сокетами» в ОС Windows используется библиотека Winsock,представляющая собой набор библиотечных АPI - функций , разделенных на четыре группы:

* Локального управления
* Установления связи
* Обмена данными (ввода/вывода)
* Закрытия связи

**Функции локального управления** используются, для выполнения подготовительных действий, необходимых для организации взаимодействия двух программ-партнеров.

***Создание socket'а***

Создание socket'а осуществляется следующим системным вызовом

int socket (domain, type, protocol)

int domain;

int type;

int protocol;

Аргументы :

- domain задает используемый для взаимодействия семейство адресов, для стека протоколов TCP/IP он может иметь символьное значение PF/AF\_INET(семейство Internet-адресов).

- type задает тип сокета:

* SOCK\_STREAM - с установлением соединения (потоковый).
* SOCK\_DGRAM - без установления соединения(датаграммный).

- protocol задает конкретный протокол транспортного уровня (из нескольких возможных в стеке протоколов). Если этот аргумент задан равным 0, то будет использован протокол "по умолчанию" (TCP для SOCK\_STREAM и UDP для SOCK\_DGRAM при использовании стека протоколов TCP/IP).

При удачном завершении своей работы данная функция возвращает дескриптор socketа - целое неотрицательное число, однозначно его идентифицирующее.

При ошибке функция возвращает число "-1".

***Привязка сокета к локальному адресу - bind***

Локальный адрес = ( IP- адрес узла + тип протокола+ номер порта), int

*bind (s, addr, addrlen)*

*int s;*

*struct sockaddr \*addr;*

*int addrlen;*

Аргументы:

- s задает дескриптор связываемого socket'а;

- addr указатель на структуру данных, содержащую локальный адрес, приписываемый socketу. Для сетей TCP/IP такой структурой является sockaddr\_in.

- addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

В случае успеха bind возвращает 0, в противном случае - "-1".

Структура sockaddr\_in используется несколькими функциями socket-интерфейса и определена следующим образом

*struct sockaddr\_in { short sin\_family;*

*u\_short sin\_port;*

*struct in\_addr sin\_addr;*

*char sin\_zero[8];* };

Поле sin\_family определяет используемое семейство адресов, (для TCP/IP имеет значение AF\_INET).

Поле sin\_addr содержит адрес (IP) узла сети.

Поле sin\_port содержит номер порта на узле сети.

Поле sin\_zero не используется.

В свою очередь определение структуры **in\_addr** таково:

*struct in\_addr {*

*union {u\_long S\_addr;/*/32 – разрядное целое число для представления IP- адреса

/\* другие (не интересующие нас) члены объединения \*/*}*

*S\_un;*

*#define s\_addr S\_un.S\_addr };*

Структура sockaddr\_in должна быть полностью заполнена перед выдачей истемного вызова bind. При этом, если поле sin\_addr.s\_addr имеет значение INADDR\_ANY, и сервер имеет несколько IP- адресов, то системный вызов будет привязывать socket ко всем сразу.

Для инициализации библиотеки Winsock необходимо выполнить вызов функции WSAStartup ().

**Функции установления связи – listen, accept, connect**

Для установления связи "клиент-сервер" используются системные вызовы listen и accept (на стороне сервера), а также connect (на стороне клиента).

***Ожидание установления связи- listen***

Системный вызов listen открывает порт на прослушивание запросов на установление связи.

*int listen (s, n) ;*

*int s;*

*int n;*

Аргументы:

- s задает дескриптор socket'а, через который программа будет ожидать к ней запросы на соединене от клиентов. Socket должен быть предварительно создан системным вызовом socket и связан с адресом с помощью системного вызова bind.

-n определяет максимальную длину очереди входящих запросов на установление связи. Если какой-либо клиент выдаст запрос на установление связи при полной очереди, то этот запрос будет отвергнут.

Признаком удачного завершения системного вызова listen служит нулевой код возврата.

***Запрос на установление соединения - connect***

*int connect (s, addr, addrlen)*

*int s;*

*struct sockaddr\_in \*addr;*

*int addrlen;*

Аргументы:

-s задает дескриптор socket'а программы клиента,

- addr должен указывать на структуру данных sockaddr\_in, содержащую IP-адрес и порт, приписанный socket'у программы-сервера, к которой делается запрос на соединение.( Для формирования значений полей структуры sockaddr\_in удобно использовать функцию gethostbyname,, транслирующая символическое имя узла сети в его IP адрес).

- addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

Для того, чтобы запрос на соединение был успешным, необходимо, по крайней мере, чтобы программа-сервер выполнила к этому моменту системный вызов listen.

При успешном выполнении запроса системный вызов connect возвращает 0, в противном случае - "-1«.

***Прием запроса на установление соединения и установки виртуального канала - accept,***

*int accept (s, addr, p\_addrlen)*

*int s;*

*struct sockaddr\_in \*addr;*

*int \*p\_addrlen;*

Аргументы:

- s задает дескриптор socket'а, через который программа-сервер получила запрос на соединение (посредством системного запроса listen

-addr должен указывать на область памяти, в которой разместится структуру данных, содержащая адрес socket'а программы-клиента, сделавшей запрос на соединение. Никакой инициализации этой области не требуется.

-p\_addrlen задает размер (в байтах) области памяти, указываемой аргументом addr.

Возвращает новый, «привязанный сокет » s\_new для обслуживания соединения созданного функцией accept

Признаком неудачного завершения accept служит отрицательное возвращенное значение.

Функция блокирующая, если очередь запросов на момент выполнения accept пуста, то программа переходит в состояние ожидания поступления запросов от клиентов на неопределенное время.

Системный вызов accept используется в программах-серверах, функционирующих только в режиме с установлением TCP.

Если к моменту выполнения connect используемый им socket клиента не был привязан к адресу посредством bind ,то такая привязка будет выполнена автоматически.

Примечание. В режиме взаимодействия без установления соединения необходимости в выполнении системного вызова connect нет.

***Функции обмена данными - read, write, send, recv, sendto, recvfrom.***

read и write – аналогичны функциям, используемым для работы с файлами (при этом вместо дескрипторов файлов в них задаются дескрипторы socket'ов).

send, recv – функции специально ориентированные на работу с socket'ами.

sendto и recvfrom - используются, как правило, для обмена данными в режиме без установления логического соединения(UDP).

***Посылка данных - send,***

*int send (s\_new , buf, len, flags)*

*int s;*

*char \*buf;*

*int len;*

*int flags;*

Аргументы:

s\_new - задает дескриптор socket'а( возвращенный функцией accept),через который посылаются данные .

buf- указывает на область памяти, содержащую передаваемые данные.

len -задает длину (в байтах) передаваемых данных.

flags- модифицирует исполнение системного вызова send. При нулевом значении этого аргумента вызов send полностью аналогичен системному вызову write.

При успешном завершении send возвращает количество байт данных переданных из области, указанной аргументом buf.

***Получение данных recv***

Для получения данных от партнера по сетевому взаимодействию используется системный вызов recv, имеющий следующий вид

*int recv (s\_new, buf, len, flags)*

*int s;*

*char \*buf;*

*int len;*

*int flags;*

s\_new - задает дескриптор socket'а, через который принимаются данные.

buf - указывает на область памяти, предназначенную для размещения принимаемых данных.

len - задает длину (в байтах) этой области.

flags - модифицирует исполнение системного вызова recv. При нулевом значении этого аргумента вызов recv полностью аналогичен системному вызову read.

При успешном завершении recv возвращает количество принятых в область, указанную аргументом buf, байт данных. recv – блокирующая функция. Если канал данных, определяемый дескриптором s, оказывается "пустым", то recv переводит программу в остояние ожидания до момента появления в нем данных.

***Функции закрытия связи – shutdown, closesocket***

Сброс буферизованных данных - shutdown

Для закрытия канала связи с партнером (путем "сброса" еще не переданных или не принятых данных) используется системный вызов shutdown, выполняемый перед close и имеющий следующий вид:

*nt shutdown (s\_new, how)*

*int s\_new;*

*int how;*

s\_new - задает дескриптор ранее созданного socket'а.

how - задает действия, выполняемые при очистке системных буферов socket'а:

0 - сбросить и далее не принимать данные для чтения из socket'а;

1 - сбросить и далее не отправлять данные для посылки через socket;

2 - сбросить все данные, передаваемые через socket в любом направлении.

Системный вызов closesocket- используется для закрытия ранее созданного socket'а

*int closesocket (s)*

*int s\_new;*

s\_new - задает дескриптор ранее созданного socket'а.

Однако в режиме с установлением логического соединения внутрисистемные механизмы обмена будут пытаться передать/принять данные, оставшиеся в канале передачи на момент закрытия socket'а. На это может потребоваться значительный интервал времени, неприемлемый для некоторых приложений. Для этого необходимо предварительно использовать системный вызов shutdown.

*Формирование адреса узла сети*

Для получения адреса узла сети TCP/IP по его символическому имени используется библиотечная функция

struct hostent \*gethostbyname (name) char \*name;

Аргумент name задает адрес последовательности литер, образующих символическое имя узла сети. При успешном завершении функция возвращает указатель на структуру hostent, имеющую следующий вид

struct hostent {

char \*h\_name;

char \*\*h\_aliases;

int h\_addrtype;

int h\_lenght;

char \*h\_addr; };

Поле h\_name указывает на официальное (основное) имя узла.

Поле h\_aliases указывает на список дополнительных имен узла (синонимов), если они есть.

Поле h\_addrtype содержит идентификатор используемого набора протоколов, для сетей TCP/IP это поле будет иметь значение AF\_INET.

Поле h\_lenght содержит длину адреса узла.

Поле h\_addr указывает на область памяти, содержащую адрес узла в том виде, в котором его используют системные вызовы и функции socket-интерфейса.

**Взаимодействие без установления соединения**

Этот вид взаимодействия предусматривает обмен датаграммами, характеризуется минимальным уровнем сервиса со стороны системы и минимальными служебными затратами. Целостность передаваемых данных системой не гарантируются, при необходимости о ней должны заботиться сами взаимодействующие программы. Используемый тип сокета- SOCK\_DGRAM, которому в IP-сетях соответствует протокол UDP. Порядок взаимодействия схематично показан на рисунке.

Полезной особенностью большинства протоколов датаграммной передачи, в том числе и UDP, является *широковещание —* доставка соответствующим образом адресованного сообщения всем доступным узлам сети (подсети). На­пример, посредством широковещательного запроса можно обнаружить сервер, адрес которого не известен (очевидно, порт программы-сервера тем не менее знать необходимо), для чего ряд служб, ориентированных на использование TCP-соединений, поддерживают также и интерфейс UDP. (Отметим, что широ­ковещание действует не всегда: оно может не поддерживаться, быть запрещено административно или отключаться при отсутствии реального соединения с се­тью, когда реально функционирует только интерфейс *localhost.)*

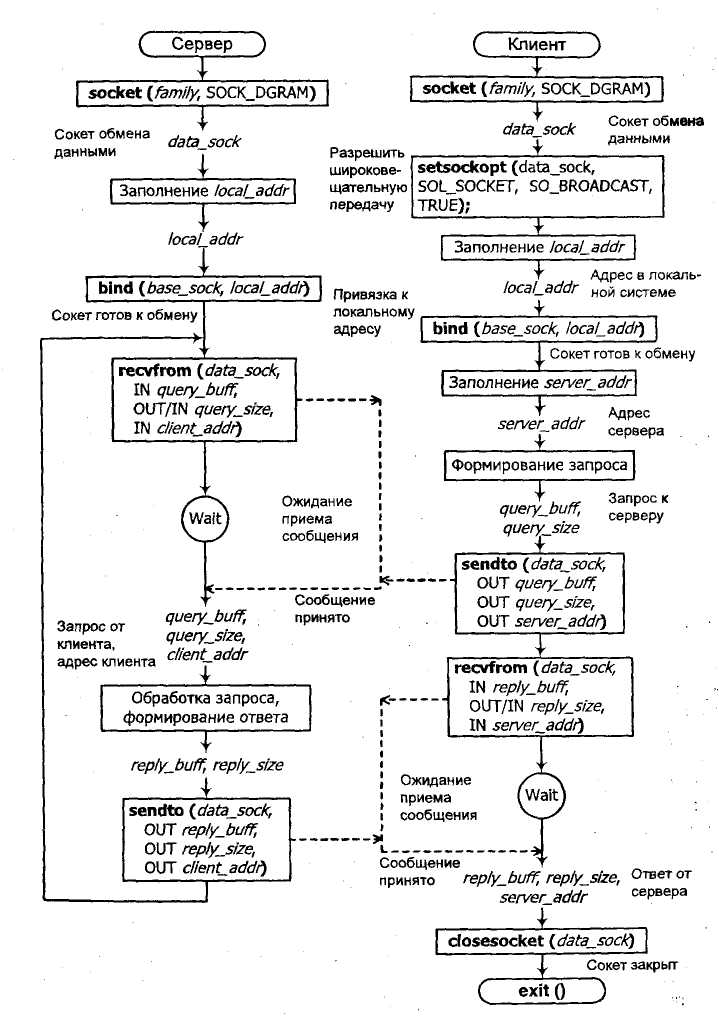


Рис. Взаимодействие без установления соединения

На схеме имена системных функций сохранены, но формат их вызова упрощен для повышения наглядности. Аргументы, описывающие передаваемые и принимаемые сообщения, снабжены пометками, являются ли соответствующие объекты параметрами вызова или создаются (заполняются) в результате его выполнения.

В показанном примере действия клиента и сервера в общем схожи, и с точки зрения программирования сокетов отличия между ними сугубо номи­нальные. Каждый из них использует всего один сокет, служащий и для приема, и для передачи сообщений. Ради упрощения считаем, что клиент отсылает сер­веру единственной запрос *{query)* и ожидает ответ на него *{reply),* а сервер в цикле принимает запросы, обрабатывает их и отсылает ответы, используя «об­ратные адреса» запросов. Цикл сервера показан бесконечным; на практике для управления серверами служат команды, предусмотренные в протоколе обмена, или какие-либо дополнительные средства.

Характерным примером сетевой службы с подобным порядком взаимо­действия служит *echo:* «эхо»-сервер принимает все запросы от всех клиентов и немедленно возвращает их отправителям. Простейшая реализация «эхо»-сервера и обращающегося к нему клиента приведена в прил. 1. Обе программы консольные, необходимые параметры передаются посредством командной строки. Сервер позволяет указывать порт для приема сообщений (по умолча­нию- 7 согласно *RFC 1700).*

Клиент обязательно требует указания адреса или сетевого имени сервера и порта, прочие аргументы рассматриваются как сооб­щения, каждое из которых клиент передает серверу и затем ожидает от него от­вета. Порт локального адреса клиента выбирается автоматически.

Важно отметить, что некоторые действия в программах являются блоки­рующими, то есть приостанавливают выполнение на неопределенный срок. В первую очередь, это ожидание поступления сообщения (функция recvfrom()). Подобная ситуация очень характерна для сетевых приложений, причем она усу­губляется неопределенностью состояния партнера и каналов связи. В результа­те, например, при отсутствии сервера может произойти как распознавание ошибки приема, так и переход клиента в состояние бесконечного ожидания (в ходе экспериментов этот эффект наблюдался только при запросах на широко­вещательный адрес), соответствие пар запрос-ответ может нарушаться, если возвращается более одного ответа, и т. д. Для реальных приложений такие де­фекты обычно недопустимы.

**Взаимодействие с установлением соединения**

Наличие постоянного соединения (виртуального канала) усложняет порядок взаимодействия, но зато обеспечивает более развитый сервис, в первую очередь контроль целостности сеанса и передаваемых данных. Используемый тип сокета- SOCK\_STREAM, которому в IP-сетях соответствует протокол TCP. Порядок взаимодействия схематично показан на рис. ???????

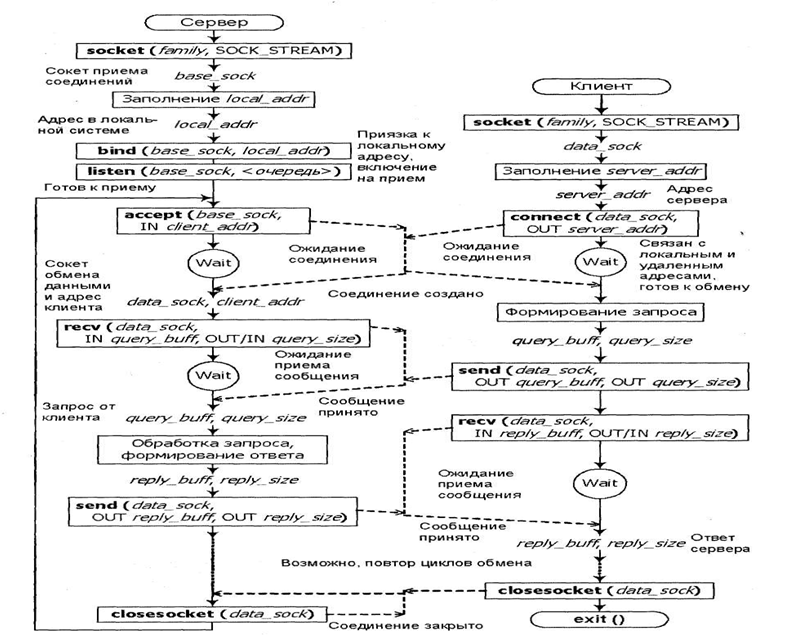


Рис. Взаимодействие с установлением соединения (UDP)

В начале работы сервер создает и связывает с локальным адресом един­ственный сокет (назовем его *базовый сокет),* на котором он будет производить прием запросов на соединение. Именно порт, с которым он связан, указывается в качестве порта, занятого данным сервером. Далее базовый сокет переводится в режим «прослушивания», и, начиная с этого момента, в системе организуется очередь, принимающая запросы на соединение. Основной цикл сервера состоит в приеме этих запросов вызовом accept(). При поступлении такого запроса сис­темой создается новое соединение, и вызов accept() на стороне сервера успеш­но завершается, возвращая новый сокет, связанный с этим соединением - *сокет данных,* уже готовый к использованию. Адрес для сокета данных выбирается системой автоматически, и он может совпадать с адресом базового сокета.

После этого сервер начинает вложенный цикл обмена данными (сообще­ниями) через новое соединение, чаще всего это ожидание входящего запроса, его обработка и отсылка ответа клиенту. В отличие от UDP при передаче дан­ных адрес назначения указывать уже не нужно, поэтому используется более простой вызов send(). Признаком завершения в простейшем случае может слу­жить прием одной или нескольких порций данных нулевой длины. В протоколе прикладного уровня может быть также предусмотрены извещения от клиента о прекращении обмена или возможность разрыва соединения по инициативе сер­вера. После окончания вложенного цикла сервер может вернуться к приему следующего соединения.

Со своей стороны клиент создает сокет и передает ее в вызов connect(), указывая также и адрес сервера, с которым нужно установить соединение -IP-адрес хоста, где находится сервер, и закрепленный за ним порт. Делать пред­варительную привязку этого сокета к локальному адресу необязательно – в этом случае система выполнит ее автоматически. После успешного завершения вызова connect() сокет будет связан с созданным соединением и готов к работе, клиент может начинать цикл обмена данными с сервером.

Блокирующими вызовами здесь помимо recv() являются accept() и соп-nectQ. Вызов accept() фактически принимает запрос на соединение из очереди, и если очередь пуста, то он остается в состоянии ожидания. Вызов connectQ блокируется, если сгенерированный им запрос на соединение не обслужен сер­вером, но поставлен в очередь. Если в очереди сервера свободных мест нет, то запрос отвергается немедленно и соответствующий connect() клиента заверша­ется как неуспешный. Кроме того, во время обслуживания клиентского соеди­нения сервер с описанным алгоритмом приостанавливает прием новых соеди­нений (отметим, что по этой причине здесь не имеет смысла резервировать большую очередь для запросов). Размер очереди задается при «включении» ба­зового сокета вызовом listen().

Определенной проблемой является контроль целостности ТСР-соединения. Алгоритмы TCP таковы, что прекращение связи обнаруживается только при обмене сегментами (система сама также может посылать специаль­ные сегменты для проверки «живости» соединения). Корректный разрыв соединения также должен сопровождаться специальными флагами в заголовке ТСР-сегмента. Поэтому полная потеря связи, аварийное (без со­блюдения процедуры разрыва) завершение или просто неактивность програм­мы на другом конце соединения без специальных мер обычно не отличимы друг от друга. Вариантом решения этой проблемы может быть введение в при­кладной протокол обмена (поверх TCP) обязательных периодических «пустых» посылок или специальных «зондирующих» сообщений в сочетании с использованием тайм-аутов.

В качестве примера приводится TCP-вариант «эхо»-сервера (служба *echo* предусматривает поддержку обоих протоколов: TCP и UDP) и простейший демонстрационный TCP-клиент. Оба приложения близки примеру. Сервер создает базовый сокет и в цикле выполняет на нем прием соединений (функция accept()). Для каждого возвращенного ею сокета данных, соответст­вующего вновь созданному соединению, выполняется вложенный цикл, со­стоящий в приеме (recv()) порции данных из этого соединения и немедленной отсылке (send()) ее обратно. Признаком завершения цикла здесь служит полу­чение сообщения нулевой длины (прием пустого сегмента), после чего сокет данных закрывается, и повторяется итерация внешнего цикла (accept()).

Кли­ент устанавливает соединение с сервером (функция connect()), передает (send()) через это соединение каждую переданную в качестве аргумента стро­ку, всякий раз дожидаясь приема ответа (recv()), после чего разрывает соеди­нение (shutdownQ, closesocketO).

Напомним, что в качестве универсального TCP-клиента может служить программа *telnet,* различные варианты либо аналоги которой присутствуют в большинстве современных систем.

Оба примера работоспособны, но максимально упрощены для сокращения объема и повышения наглядности исходного текста. Реальные приложения должны выполняться в более «правильном» стиле: структуриро­ванными, с выделением подпрограмм, развернутой обработкой ошибок и т. д. Но главное — на практике, как правило, необходимо учитывать вероятные дли­тельные блокировки программы в состоянии ожидания и обеспечивать возмож­ность работы сервера более чем с одним клиентом.

***Пример использования сокетов: датаграммы UDP***

/\* - ---

- **UDP\_ECHO.C - демонстрационный "сервер" UDP** (служба **echo)**: -

- возвращает все получаемые сообщения отправителям -

- Вызов: **udp\_echo** [<***порт***>]; завершение: <Ctrl-C> -

- Порт по умолчанию - **7** -

- В проект должен быть включен файл **wsock32.lib** -

--- - \*/

#include <windows.h>

#include <stdio.h>

#include <winsock.h>

#define DEFAULT\_ECHO\_PORT 7

char DataBuffer [1024];

int main (int argc, char\*\* argv)

{

struct sockaddr\_in SockAddrLocal, SockAddrRemote;

SOCKET SockLocal = INVALID\_SOCKET;

unsigned short nPort = DEFAULT\_ECHO\_PORT;

int nAddrSize, nCnt;

WSADATA WSAData;

WORD wWSAVer;

//разбор командной строки: номер порта

if (argc > 1)

if (sscanf (argv[1], "%u", &nPort) < 1)

fprintf (stderr, "Ошибочный порт: %s, use default", nPort);

//инициализация подсистемы сокетов

wWSAVer = MAKEWORD (1, 1);

if (WSAStartup (wWSAVer, &WSAData) != 0) {

puts ("Ошибка инициализации подсистемы WinSocket");

return -1;

}

//создание локального сокета

SockLocal = socket (PF\_INET, SOCK\_DGRAM, 0);

if (SockLocal==INVALID\_SOCKET) {

fputs ("Ошибка создания сокета\n", stderr);

return -1;

}

//привязка сокета к локальному адресу

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

SockAddrLocal.sin\_family = AF\_INET;

SockAddrLocal.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

SockAddrLocal.sin\_port = htons (nPort); //(<номер\_порта\_сервера>);

if (bind (SockLocal,

(struct sockaddr\*) &SockAddrLocal, sizeof(SockAddrLocal)

) != 0)

{

fprintf (stdout, "Ошибка привязки сокета, порт %u\n",

ntohs(SockAddrLocal.sin\_port)

);

return -1;

}

fprintf (stderr, "Сервер запущен, порт %u\n",

ntohs(SockAddrLocal.sin\_port)

);

//основной рабочий цикл

while (1) { //для сервера цикл обычно бесконечен

nAddrSize = sizeof (SockAddrRemote);

//принять входящее сообщение ("эхо-запрос")

nCnt = recvfrom (SockLocal,

DataBuffer, sizeof(DataBuffer)-1, 0,

(struct sockaddr\*) &SockAddrRemote, &nAddrSize

);

if (nCnt < 0) { //ошибка приема запроса

fputs ("Ошибка приема сообщения\n", stderr);

continue;

}

//вернуть сообщение как эхо-отклик

sendto (SockLocal,

DataBuffer, nCnt, 0,

(struct sockaddr\*) &SockAddrRemote, sizeof (SockAddrRemote)

);

}

//завершение - здесь никогда не достигается!

shutdown (SockLocal, 2);

Sleep (100);

closesocket (SockLocal); SockLocal = INVALID\_SOCKET;

WSACleanup ();

return 0;

/\*UDP\_SEND.C - демонстрационный "клиент" UDP: -

отсылает на "сервер" одно или несколько сообщений,

ждет и отображает полученные ответы -

Вызов: **udp\_send** <***адрес*/*имя***> <***порт***> <***сообщение1***> … -

Завершение: исчерпание списка сообщений или <Ctrl-C> - В проект должен быть включен файл **wsock32.lib** -

--- - \*/

#include <windows.h>

#include <stdio.h>

#include <winsock.h>

char DataBuffer [1024];

int main (int argc, char\*\* argv)

{

struct sockaddr\_in SockAddrLocal, SockAddrSend, SockAddrRecv;

SOCKET SockLocal = INVALID\_SOCKET;

struct hostent\* pHostEnt;

int nSockOptBC, nAddrSize, nPortRemote, nMsgLen, i;

WSADATA WSAData;

WORD wWSAVer;

//командная строка

if (argc < 3) {

puts ("Недостаточно аргументов\n");

puts ("Вызов: UDP\_SEND <addr/name> <port> <msg1> ...\n");

return -1;

}

//инициализация подсистемы сокетов

wWSAVer = MAKEWORD (1, 1);

if (WSAStartup (wWSAVer, &WSAData) != 0) {

puts ("Ошибка инициализации WinSockets");

return -1;

}

//создание локального сокета

SockLocal = socket (PF\_INET, SOCK\_DGRAM, 0);

if (SockLocal==INVALID\_SOCKET) {

fputs ("Ошибка создания сокета\n", stderr);

return -1;

}

//настройка сокета: разрешить отсылку на "широковещательный" адрес

nSockOptBC = 1;

setsockopt (SockLocal,

SOL\_SOCKET, SO\_BROADCAST,

(char\*) (&nSockOptBC), sizeof(nSockOptBC)

);

//привязка сокета к "локальному" адресу

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

SockAddrLocal.sin\_family = AF\_INET;

SockAddrLocal.sin\_addr.S\_un.S\_addr = INADDR\_ANY; //все интерфейсы

SockAddrLocal.sin\_port = 0; //выбирать порт автоматически

if (bind(SockLocal,(struct sockaddr\*)&SockAddrLocal,sizeof(SockAddrLocal)) != 0)

{

fputs ("Ошибка привязки к локальному адресу\n", stderr);

return -1;

}

//подготовка адреса сервера

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

SockAddrSend.sin\_family = AF\_INET;

if (strcmp (argv[1], "255.255.255.255") == 0) //адрес broadcast

SockAddrSend.sin\_addr.S\_un.S\_addr = INADDR\_BROADCAST;

else {

SockAddrSend.sin\_addr.S\_un.S\_addr = inet\_addr (argv[1]);

if (SockAddrSend.sin\_addr.S\_un.S\_addr == INADDR\_NONE) {

if ((pHostEnt = gethostbyname (argv[1])) == NULL) {

fprintf (stderr, "Хост не опознан: %s\n", argv[1]);

return -1;

}

SockAddrSend.sin\_addr = \*(struct in\_addr\*)(pHostEnt->h\_addr\_list[0]);

}

}

if (sscanf (argv[2], "%u", &nPortRemote) < 1) {

fprintf (stderr, "Ошибочный номер порта: %s\n", argv[2]);

return -1;

}

SockAddrSend.sin\_port = htons ((unsigned short)nPortRemote);

//рабочий цикл

for (i=3; i<argc; ++i) { //остальные аргументы командной строки

//отослать сообщение

fprintf (stdout, "Отсылка на %s:%u: \"%s\" \n",

inet\_ntoa(SockAddrSend.sin\_addr),

ntohs(SockAddrSend.sin\_port),

argv[i]

);

nMsgLen = strlen (argv[i]) + 1;

if (sendto (SockLocal, argv[i], nMsgLen, 0,

(struct sockaddr\*) &SockAddrSend, sizeof (SockAddrSend)) < nMsgLen)

{

fprintf (stderr, "Ошибка отсылки: \"%s\"\n", argv[i]);

continue;

}

//принять и отобразить ответ

nAddrSize = sizeof (SockAddrRecv);

nMsgLen = recvfrom (SockLocal, DataBuffer, sizeof(DataBuffer)-1, 0,

(struct sockaddr\*) &SockAddrRecv, &nAddrSize

);

if (nMsgLen <= 0) { //ошибка приема запроса

fputs ("Ошибка приема ответа\n", stderr);

continue;

}

DataBuffer [nMsgLen] = '\0';

fprintf (stdout, "Ответ от %s:%u: \"%s\" \n",

inet\_ntoa (SockAddrRecv.sin\_addr),

ntohs (SockAddrRecv.sin\_port),

DataBuffer

);

}

//завершение

shutdown (SockLocal, 2);

Sleep (100);

closesocket (SockLocal); SockLocal = INVALID\_SOCKET;

WSACleanup ();

return 0;

}

**Пример использования сокетов: соединения TCP**

/\* - ---

**TCP\_ECHO.C** - демонстрационный сервер TCP (служба **echo**): -

принимает соединения от клиентов, -

принимает сообщения и возвращает их клиентам. -

Одновременно поддерживает только одно соединение. -

Вызов: **tcp\_echo** [<***порт***>]; завершение: <Ctrl-C> -

Порт по умолчанию - 7 -

В проект должен быть включен файл **wsock32.lib** -

--- - \*/

#include <windows.h>

#include <stdio.h>

#include <winsock.h>

#define DEFAULT\_ECHO\_PORT 7

char DataBuffer [1024];

int main (int argc, char\*\* argv)

{

struct sockaddr\_in SockAddrBase, SockAddrPeer;

SOCKET SockBase = INVALID\_SOCKET, SockData = INVALID\_SOCKET;

unsigned short nPort = DEFAULT\_ECHO\_PORT;

int nAddrSize, nCnt;

WSADATA WSAData;

WORD wWSAVer;

//разбор командной строки: номер порта

if (argc > 1)

if (sscanf (argv[1], "%u", &nPort) < 1)

fprintf (stderr, "Ошибочный порт: %s, use default", nPort);

//инициализация подсистемы сокетов

wWSAVer = MAKEWORD (1, 1);

if (WSAStartup (wWSAVer, &WSAData) != 0) {

puts ("Ошибка инициализации подсистемы WinSocket");

return -1;

}

//создание локального сокета

SockBase = socket (PF\_INET, SOCK\_STREAM, 0);

if (SockBase == INVALID\_SOCKET) {

fputs ("Ошибка создания сокета\n", stderr);

return -1;

}

//привязка базового сокета к локальному адресу

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

SockAddrBase.sin\_family = AF\_INET;

SockAddrBase.sin\_addr.S\_un.S\_addr = INADDR\_ANY;

SockAddrBase.sin\_port = htons (nPort); //(<номер\_порта\_сервера>);

if (bind (SockBase,

(struct sockaddr\*) &SockAddrBase, sizeof(SockAddrBase)

) != 0)

{

fprintf (stderr, "Ошибка привязки к локальному порту: %u\n",

ntohs(SockAddrBase.sin\_port)

);

return -1;

}

//включение режима "прослушивания"

if (listen (SockBase, 2) != 0) { //очередь на 2 места

closesocket (SockBase);

fputs ("Ошибка включения режима прослушивания\n", stderr);;

return -1;

}

fprintf (stderr,

"Сервер запущен, порт %u\n",

ntohs(SockAddrBase.sin\_port)

);

//основной рабочий цикл - прием и обслуживание соединений

while (1) { //для сервера цикл обычно бесконечен

nAddrSize = sizeof (SockAddrPeer);

SockData = accept (SockBase,

(struct sockaddr\*)&SockAddrPeer, &nAddrSize

);

if (SockData == INVALID\_SOCKET) {

fputs ("Ошибка приема соединения\n", stderr);

continue;

}

//цикл обслуживания одного соединения

while (1) {

nCnt = recv (SockData, DataBuffer, sizeof(DataBuffer)-1, 0);

if (nCnt <= 0)

break;

send (SockData, DataBuffer, nCnt, 0); //возврат "эха"

}

shutdown (SockData, 2);

closesocket (SockData); SockData = INVALID\_SOCKET;

}

//завершение - здесь никогда не достигается!

shutdown (SockBase, 2);

Sleep (100);

closesocket (SockBase); SockBase = INVALID\_SOCKET;

WSACleanup ();

return 0;

}

/\*TCP\_SEND.C - демонстрационный клиент TCP: -

отсылает на сервер одно или несколько сообщений, -

после каждого ждет получения ответа и отображает его -

Вызов: tcp\_send <*адрес/имя*> <*порт*> <*сообщение1*> ... -

Завершение: исчерпание списка сообщений или <Ctrl-C> -

В проект должен быть включен файл wsock32.lib -

--- - \*/

#include <windows.h>

#include <stdio.h>

#include <winsock.h>

char DataBuffer [1024];

int main (int argc, char\*\* argv)

{

struct sockaddr\_in SockAddrServer;

SOCKET SockData = INVALID\_SOCKET;

struct hostent\* pHostEnt;

int nPortServer, nMsgLen, i;

WSADATA WSAData;

WORD wWSAVer;

//командная строка

if (argc < 3) {

puts ("Недостаточно аргументов\n");

puts ("Вызов: TCP\_SEND <addr/name> <port> <msg1> ...\n");

return -1;

}

//инициализация подсистемы сокетов

wWSAVer = MAKEWORD (1, 1);

if (WSAStartup (wWSAVer, &WSAData) != 0) {

puts ("Ошибка инициализации WinSocket");

return -1;

}

//подготовка адреса сервера

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

SockAddrServer.sin\_family = AF\_INET;

if (strcmp (argv[1], "255.255.255.255") == 0) //адрес broadcast

SockAddrServer.sin\_addr.S\_un.S\_addr = INADDR\_BROADCAST;

else {

SockAddrServer.sin\_addr.S\_un.S\_addr = inet\_addr (argv[1]);

if (SockAddrServer.sin\_addr.S\_un.S\_addr == INADDR\_NONE) {

if ((pHostEnt = gethostbyname (argv[1])) == NULL) {

fprintf (stderr, "Хост не опознан: %s\n", argv[1]);

return -1;

}

SockAddrServer.sin\_addr = \*(struct in\_addr\*)(pHostEnt->h\_addr\_list[0]);

}

}

if (sscanf (argv[2], "%u", &nPortServer) < 1) {

fprintf (stderr, "Ошибочный номер порта: %s\n", argv[2]);

return -1;

}

SockAddrServer.sin\_port = htons ((unsigned short)nPortServer);

//создание локального сокета

SockData = socket (PF\_INET, SOCK\_STREAM, 0);

if (SockData == INVALID\_SOCKET) {

fputs ("Ошибка создания сокета\n", stderr);

return -1;

}

//запрос на установление соединения

if (connect (SockData,

(const struct sockaddr\*)&SockAddrServer, sizeof(SockAddrServer)) != 0)

{

fprintf (stderr, "Ошибка соединения с %s:%u\n",

inet\_ntoa (SockAddrServer.sin\_addr),

ntohs (SockAddrServer.sin\_port)

);

closesocket (SockData);

return -1;

}

fprintf (stdout, "Установлено соединение с сервером: %s:%u\n",

inet\_ntoa (SockAddrServer.sin\_addr),

ntohs (SockAddrServer.sin\_port)

);

//рабочий цикл

for (i=3; i<argc; ++i) { //остальные аргументы командной строки

//отослать сообщение

fprintf (stdout, "Отсылка: \"%s\" \n", argv[i]);

nMsgLen = strlen (argv[i]) + 1;

if (send (SockData, argv[i], nMsgLen, 0) < nMsgLen) {

fprintf (stderr, "Ошибка отсылки: \"%s\"\n", argv[i]);

continue;

}

//принять и отобразить ответ

fprintf (stdout, "Прием...");

nMsgLen = recv (SockData, DataBuffer, sizeof(DataBuffer)-1, 0);

if (nMsgLen <= 0) { //ошибка приема ответа

fputs ("Ошибка приема\n", stderr);

continue;

}

DataBuffer [nMsgLen] = '\0';

fprintf (stdout, "\b\b\b: \"%s\" \n", DataBuffer); }

//завершение

shutdown (SockData, 2);

Sleep (100);

closesocket (SockData); SockData = INVALID\_SOCKET;

WSACleanup ();

return 0;

}

Указания к выбору варианта

На основе анализа приведенных примеров написать программу, обмена текстовыми сообщениями между двумя компьютерами ( можно простейший «чат»).

Используя **только Win Api** для программирования сокетов, написать программы клиента и сервера взаимодействующих между собой по протоколу UDP и TCP. (Программы можно запускать две на одном компьютере используя для IP – адреса сервера адрес - 127.0.0.1.

**Варианты заданий**

Вариант №1

Разработать две клиент - серверные программы. Клиент отсылает серверу два числа *L* и *U*, введенные пользователем, где *L* – это нижняя граница диапазона, *U* – верхняя граница диапазона. Сервер принимает значения границ диапазона, вычисляет сумму и произведение чисел от *L* до *U* и выводит полученные значения на экран и отсылает их клиенту. Клиент так же выводит их на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №2

Разработать две клиент - серверные программы. Клиент отсылает серверу введенный пользователем номер числа Фибоначчи. Сервер принимает номер, вычисляет число Фибоначчи с этим номером, по формуле *Fi* = *Fi*–1 + *Fi*–2, *F*0 = *F*1 = 1 и выводит полученные значения на экран и отсылает их клиенту. Клиент так же выводит их на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №3

Разработать две клиент - серверные программы. Клиент отсылает серверу введенную пользователем строку, хранящую знаковое целое число.   
Сервер принимает строку, хранящую знаковое целое число, и выводит на экран строковый эквивалент этого числа прописью (например, ввод «-1211» должен приводить к выводу «минус тысяча двести одиннадцать») после чего отсылает их клиенту. Клиент так же выводит их на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №4

Разработать две клиент - серверные программы. Клиент отсылает серверу введенную пользователем строку, хранящую число со знаком и плавающей точкой. Сервер принимает строку, хранящую число со знаком и плавающей точкой, и выводит на экран строковый эквивалент этого числа прописью (например, ввод «-12.11» должен приводить к выводу «минус двенадцать целых одиннадцать сотых»). сервер отсылает строку клиенту, который так же выводит её на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №5

Разработать две клиент - серверные программы. Клиент отсылает серверу две строки, введенные пользователем. Сервер принимает две строки.   
Далее, если обе строки хранят целые числа со знаком, то на экран выводится сумма чисел, в противном случае – конкатенация двух введенных строк. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №6

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы двух прямоугольных матриц, введенные пользователем. Сервер принимает две прямоугольных матрицы, а затем выводит на экран их сумму и произведение. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №7

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы вектора (одномерного целочисленного массива), введенные пользователем. Сервер принимает вектор , упорядочивает его по возрастанию любым из так называемых «улучшенных алгоритмов» сортировки массивов и выводит на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №8

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы вектора (одномерного массива чисел с плавающей точкой), введенные пользователем. Сервер принимает вектор, упорядочивает его по возрастанию любым из так называемых «улучшенных алгоритмов» сортировки массивов и выводит на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №9

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы вектора (одномерного массива строк), введенные пользователем. Сервер принимает вектор, упорядочивает его по возрастанию любым из так называемых «улучшенных алгоритмов» сортировки массивов и выводит на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №10

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы введенной пользователем квадратной матрицы. Сервер принимает матрицу, затем вычисляет сумму элементов, лежащих на главной и побочной диагоналях, и выводит на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №11

Разработать две клиент - серверные программы. Клиент отсылает серверу элементы введенной пользователем квадратной матрицы. Сервер принимает матрицу, затем вычисляет сумму элементов, не лежащих на главной и побочной диагоналях, и выводит на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №12

Разработать две клиент - серверные программы. Клиент принимает от пользователя две даты – строки вида ЦЦ.ЦЦ.ЦЦЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает серверу. Сервер принимает даты , вычисляет полное количество дней, прошедших между двумя полученными датами, и выводит его на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №13

Разработать две клиент - серверные программы. Клиент принимает от пользователя два значения времени – строки вида ЦЦ.ЦЦ.ЦЦ, где Ц – это любая цифра из диапазона [0-9], и отсылает серверу. Сервер принимает обе строки, вычисляет полное количество секунд, прошедших между двумя значениями времени, и выводит его на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколом *UDP* и TCP.

Вариант №14

Разработать две клиент - серверные программы. Клиент отсылает серверу две строки, введенные пользователем. Сервер принимает, осуществляет поиск вхождения второй строки в первую любым известным методом и выводит на экран значение индекса элемента первой строки, с которого началось совпадение, или -1 в противном случае. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №15

Разработать две клиент - серверные программы. Клиент отсылает серверу две строки, введенные пользователем. Сервер принимает две строки, осуществляет поиск количества вхождений второй строки в первую любым известным методом, и выводит на экран полученное значение. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №16

Разработать две клиент - серверные программы. Клиент принимает от пользователя дату – строку вида ЦЦ.ЦЦ.ЦЦЦЦ, где Ц – это любая цифра из диапазона [0-9,] и отсылает серверу. Сервер принимает дату и выводит на экран число и месяц прописью, а за последними четырьмя – слово «года» (например, ввод «29.02.2016» приводит к выводу «Двадцать девятое февраля 20 Результат отправляется клиенту для вывода на экран. года»). Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №17

Разработать две клиент - серверные программы. Клиент принимает от пользователя значение времени – строку вида ЦЦ.ЦЦ.ЦЦ, где Ц – это любая цифра из диапазона [0-9] и отсылает серверу. Сервер принимает значение времени и выводит на экран значение часов минут и секунд прописью (например, ввод «12.01.20» приводит к выводу «двенадцать часов одна минута двадцать секунд»). Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №18

Разработать две клиент - серверные программы. Клиент принимает от пользователя строку из нулей и единиц («битовую строку») и отсылает серверу. Сервер принимает битовую строку, инвертирует ее, выводит на экран значение инвертированной строки, переводит ее в число в десятичный формат и выводит полученное число на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №19

Разработать две клиент - серверные программы. Клиент принимает от пользователя строку из нулей и единиц («битовую строку») и отсылает серверу. Сервер принимает битовую строку, осуществляет ее реверс (нули заменяются на единицы, а единицы на нули). Полученная строка выводится на экран, затем программа переводит ее в число в десятичном формате и выводит полученное число на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №20

Разработать две клиент - серверные программы. Клиент отсылает число, введенное пользователем, серверу. Сервер принимает число, вычисляет его факториал по формуле *N*! = *N* \* (*N* – 1)!, где 0 != 1, и выводит его на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №21

Разработать две клиент - серверные программы. Клиент принимает от пользователя квадратную матрицу и отсылает на сервер. Сервер принимает матрицу, осуществляет обход только крайних ее элементов, вычисляет их сумму, и выводит на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №22

Разработать две клиент - серверные программы. Клиент принимает от пользователя строку символов и отсылает серверу. Сервер принимает строку, осуществляет замену всех латинских букв на их аналоги из кириллицы и выводит результат на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №23

Разработать две клиент - серверные программы. Клиент принимает от пользователя строку символов и отсылает серверу. Сервер принимает строку, осуществляет смену регистра всех букв и выводит результат на экран. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Вариант №24

Разработать две клиент - серверные программы. Клиент принимает от пользователя беззнаковое целое число и отсылает серверу. Сервер принимает число. Если оно является степенью двойки, то на экран выводится показатель степени, и сообщение «не является степенью двойки» в противном случае. Результат отправляется клиенту для вывода на экран. Для взаимодействия воспользоваться механизмом сокетов и протоколами *UDP и TCP*.

Содержание отчета

Индивидуальная практическая работа № 1 должна содержать:

* титульный лист с фамилией, вариантом и условие индивидуального задания;
* схему алгоритма программы. ;
* текст программы c комментариями.