Лабораторная работа №2

Создание последовательного сервера без установления логического соединения (UDP)

Цель работы – изучить методы создания серверов, используя алгоритм последовательной обработки запросов.

В данной лабораторной работе мы рассмотрим одно из средств доставки пакетов – протокол UDP (User Datagram Protocol).

Каждый коммуникационный протокол оперирует с некоторой единицей передаваемых данных. Единицу данных протокола UDP называют *дейтаграммой (datagram)*. Протокол UDP был разработан Дэвидом П. Ридом в 1980 г. Протокол UDP является простейшим дейтаграммным протоколом, который используется в том случае, когда задача надежного обмена данными либо вообще не ставится, либо решается средствами более высокого уровня – системными прикладными службами или пользовательскими приложениями.

В стеке протоколов TCP/IP протокол UDP обеспечивает основной механизм, используемый прикладными программами для передачи дейтаграмм другим приложениям. UDP предоставляет протокольные порты, используемые для выделения нескольких процессов, выполняющихся на одном компьютере. Помимо посылаемых данных каждое UDP-сообщение содержит номер порта-приемника и номер порта-отправителя, делая возможным для программ UDP на машине-отправителе доставлять сообщение соответствующему реципиенту, а для получателя посылать ответ соответствующему отправителю. Этот протокол обеспечивает ненадежную доставку данных “по возможности”, в отличие от протокола TCP, обеспечивающего гарантированную доставку. UDP не использует подтверждения прихода сообщений, не упорядочивает приходящие сообщения и не обеспечивает обратной связи для управления скоростью передачи информации между машинами. Поэтому, UDP-сообщения могут быть потеряны, размножены или приходить не по порядку. Кроме того, пакеты могут приходить раньше, чем получатель сможет обработать их.

Рисунок 1 – Структура UDP пакета

Сокеты дейтаграмм

Для посылки дейтаграмм отправитель и получатель создают сокеты дейтаграммного типа (*сокеты дейтаграмм* (datagram sockets)), являющимися средством поддержки не очень надежного обмена пакетами. Ненадежность в данном контексте означает отсутствие гарантии их доставки по назначению в требуемом порядке. Фактически один и тот же пакет дейтаграммы может

быть доставлен несколько раз.

Хотя WinSock поддерживает и другие протоколы, во многих случаях, например при обмене данными между двумя процессами на одном и том же компьютере или между двумя компьютерами в локальной сети с небольшой нагрузкой, 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 загруженной сети, ненадежный характер сокетов дейтаграмм проявится очень быстро. При отсутствии в приложении обработки ошибок оно просто не справится 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 одновременно.

Примерами сетевых приложений, использующих UDP, являются NFS (Network File System, сетевая файловая система), SNMP (Simple NetworkManagement Protocol, простой протокол управления сетью). 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.

На рисунках 2-3 показана упрощенная схема организации работы последовательного сервера без установления логического соединения. Требуется только один поток выполнения, который обеспечивает взаимодействие сервера со многими клиентами с использованием одного сокета.

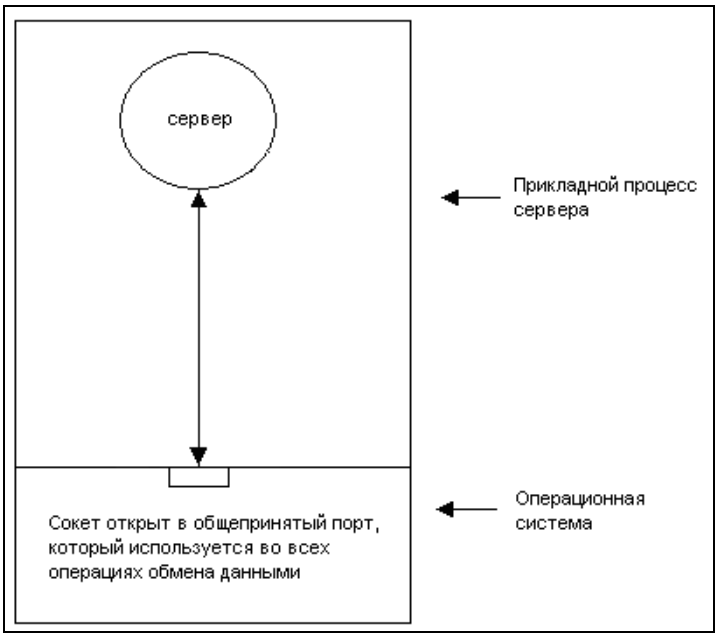

Рисунок 2 – Схема организации работы последовательного сервера без установления логического соединения

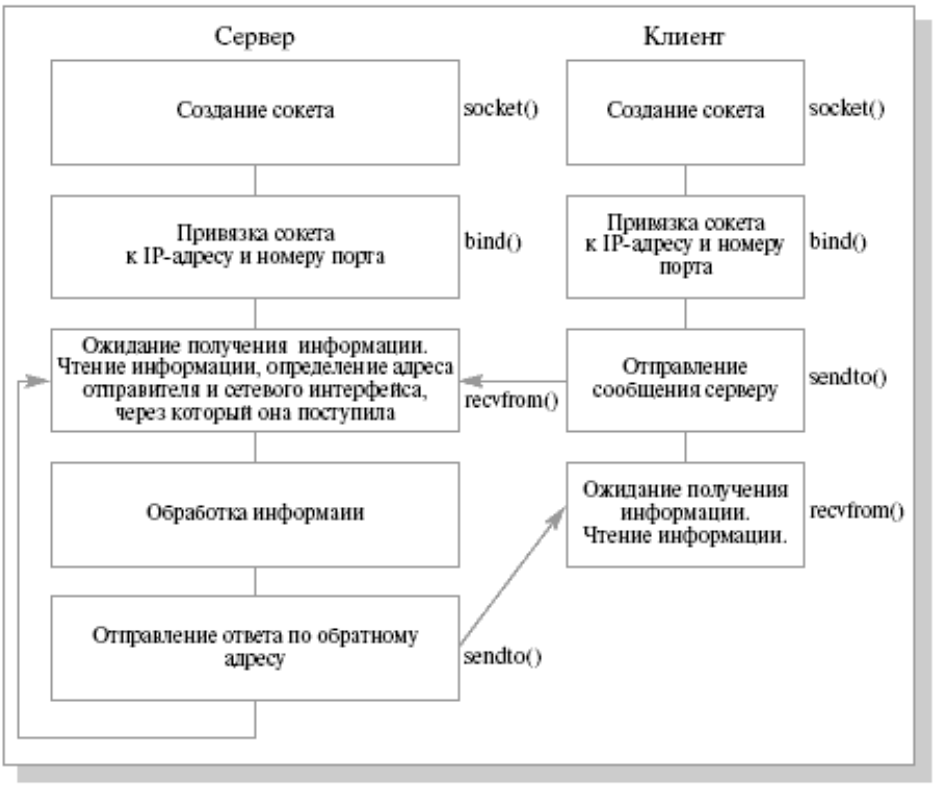

Рисунок 3 – Схема взаимодействия клиента и сервера для протокола UDP

Методические указания

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 возможности сервера реализовать следующим образом: сервер, получив эту строку, должен поменять в ней местами символы на четных и нечетных позициях. Результат возвратить назад клиенту.*

Также как и в предыдущей лабораторной работе создадим два проекта – клиент и сервер.

```
//СЕРВЕРНАЯ ЧАСТЬ
#include<winsock2.h>
#include<iostream>
#include<stdlib.h>
using namespace std;
void main(void){
     WORD wVersionRequested;
     WSADATA wsaData;
     int err;
     wVersionRequested=MAKEWORD(2,2);
     err = WSAStartup(wVersionRequested, &wsaData);
     SOCKET s;
     s = socket(AF_INET,SOCK_DGRAM,0);
```

Так как в данном случае создается сокет дейтаграмм, использующий протокол UDP, вторым аргументом, задающим тип создаваемого сокета, должен быть *SOCK_DGRAM*.

```
struct sockaddr_in ad;
ad.sin_port = htons(1024);
ad.sin_family = AF_INET;
ad.sin_addr.s_addr = 0;//подставляет подходящий ip
bind(s,(struct sockaddr*) &ad,sizeof(ad));
char b[200],tmp='\0';
int l= sizeof(ad);
```

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

```
int recvfrom(
_In_ SOCKET s,
_Out_ char *buf,
_In_ int len,
_In_ int flags,
_Out_ struct sockaddr *from,
_Inout_opt_ int *fromlen);
```

Первый параметр — это дескриптор сокета, возвращаемый функцией *socket()*; за ним идут указатель на буфер для приема новой дейтаграммы и длина буфера. Последние два параметра используются для получения адреса сокета, пославшего дейтаграмму. По этому адресу можно послать ответ.

Некоторые значения параметра *flag*:

– **флаг *MSG_PEEK*** заставляет скопировать данные из начала очереди в буфер, но не удалять их оттуда. Таким образом, последующий вызов функции вернет те же самые данные.

– **флаг *MSG_OOB*** сообщает об отправке привилегированных данных (out of band). То есть, такое сообщение передаётся вне потока. Это значит, что при отправке такого сообщения транспортный протокол не ждёт полного заполнения буфера, а отсылает сообщение немедленно. Данный флаг можно использовать при передаче приоритетных данных. При использовании *MSG_OOB*, WinSock-приложения поддерживающие связь, должны заранее "договориться" о использовании этого флага.

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 байт. При ошибочном приеме дейтаграммы возвращается 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 подсистему 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'){
               tmp=b[i];
               b[i]=b[i+1];
               b[i+1]=tmp;
               }
```

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

```
int sendto(
_In_ SOCKET s,
_In_ const char *buf,
_In_ int len,
_In_ int flags,
_In_ const struct sockaddr *to,
_In_ int tolen);
```

Первый параметр –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 назначения.

Если функция *sendto()* срабатывает корректно, она возвращает количество посланных байт, которое может отличаться от значения параметра *len*. В случае 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 *WSAStartup()*.

```
     sendto(s, b, lstrlen(b), 0, (struct sockaddr*) &ad, l);
     closesocket(s);
     WSACleanup();}

//КЛИЕНТСКАЯ ЧАСТЬ
#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=WSAStartup(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_port = 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");
int t = sizeof(add);
cout<<"Enter the string, please"<<endl;
cin.getline(buf,100,'\n');
sendto(s, buf, lstrlen(buf), 0, (struct sockaddr*) &add, t);
int rv=recvfrom(s,b, lstrlen(b), 0, (struct sockaddr*)&add, &t);
b[rv] = '\0';
cout<<b<<endl;
closesocket(s);
WSACleanup();
return 0;
}
```

Почему существуют два транспортных протокола TCP и UDP, а не один из них?

Дело в том, что они предоставляют разные услуги прикладным процессам. Большинство прикладных программ пользуются только одним из них. Вы, как программист, выбираете тот протокол, который наилучшим образом соответствует вашим потребностям. Если вам нужна надежная доставка, то следует выбрать TCP. Если вам нужна доставка дейтаграмм, то более подходящим будет UDP. Если вам нужна эффективная доставка по длинному и ненадежному каналу передачи данных, то лучшим будет протокол TCP. Если нужна эффективность на быстрых сетях с короткими соединениями – воспользуйтесь протоколом UDP. Если ваши потребности не попадают ни в одну из этих категорий, то выбор транспортного протокола не ясен. Однако прикладные программы могут устранять недостатки выбранного протокола. Например, если вы выбрали UDP, а вам необходима надежность, то прикладная программа должна обеспечить надежность. Если вы выбрали TCP, а вам нужно передавать записи, то прикладная программа должна вставлять маркеры в поток байтов так, чтобы можно было различить записи.

Контрольные вопросы

1. Как расшифровывается аббревиатура UDP?
2. Что содержит UDP-сообщение помимо посылаемых данных?
3. Что называется дейтаграммой (datagram)?
4. Какие возможности не предоставляет UDP (в отличие от TCP)?
5. Чем функции sendto() и recvfrom() отличаются от функций send() и recv()?

6. Что возвращают функции recvfrom() и sendto()?

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

8. Что произойдёт при попытке послать дейтаграмму большего размера, чем максимально допустимый в данной реализации WinSock?

9. Приведите примеры сетевых приложений, использующих UDP.

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

Варианты индивидуального задания

1. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу строку, сервер возвращает эту же строку, но гласные в ней должны быть в верхнем регистре.

2. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу дату в формате ДД.ММ.ГГ, сервер определяет существует ли она и возвращает «истина», если дата корректна, в противном случае – «ложь».

3. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу предложение, сервер возвращает количество слов в нем.

4. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу пятизначное число, сервер возвращает цифры этого числа, расположенные в порядке убывания.

5. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает произвольный набор символов серверу и получает количество сочетаний повторяющихся символов (например, набор a*bb*cmg*hhh*k содержит два сочетания).

6. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу число, сервер определяет является ли это число простым и возвращает «истина», если это так, в противном случае – «ложь» (простым является число, которое делится только само на себя).

7. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу слово и получает это же слово разбитым на слоги.

8. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу три числа и получает назад «истина», если они удовлетворяют теореме Пифагора, в противном случае – «ложь».

9. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим

образом: клиент посылает сумму денег в белорусских рублях и идентификатор валюты (EUR или USD), в зависимости от указанного идентификатора сервер возвращает эту сумму в выбранной валюте.

10. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает серверу три числа и получает назад «истина», если существует треугольник с такими сторонами, в противном случае – «ложь».

11. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает набор слов, сервер возвращает слово с максимальным количеством букв.

12. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает строку, сервер возвращает последовательность, заключенную между открывающей и закрывающей кавычками.

13. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает строку символов, сервер возвращает содержащиеся в ней цифры.

14. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает число в римской системе счисления, сервер возвращает его в арабской системе счисления.

15. Осуществить взаимодействие клиента и сервера на основе протокола UDP. Функционирование клиента и сервера реализовать следующим образом: клиент посылает набор слов, сервер находит и возвращает одинаковые слова.

ПРИЛОЖЕНИЕ А
Списки ошибок

Список кодов ошибок, которые могут возникать при вызове функции *send()*:

Код ошибки	Описание
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию WSAStartup
WSAENETDOWN	Сбой в сети
WSAEACCES	Указанный адрес является широковещательным (broadcast), однако перед вызовом функции не был установлен соответствующий флаг
WSAEINTR	Работа функции была отменена при помощи функцииWSACancelBlockingCall
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса Windows Sockets
WSAEFAULT	Параметр buf указан неправильно (он не указывает на адресное пространство, принадлежащее приложению)
WSAENETRESET	Необходимо сбросить соединение
WSAENOBUFS	Возникла блокировка буфера
WSAENOTCONN	Сокет не подсоединен
WSAENOTSOCK	Указанный в параметре дескриптор не является сокетом
WSAESHUTDOWN	Сокет был закрыт функцией shutdown
WSAEWOULDBLOCK	Сокет отмечен как неблокирующий, но запрошенная операция приведет к блокировке
WSAEMSGSIZE	Был использован сокет типа SOCK_DGRAM (предназначенный для передачи датаграмм). При этом размер пакета данных превышает максимально допустимый для данной реализации интерфейсаWindows Sockets
WSAEINVAL	Сокет не был подключен функцией bind
WSAECONNABORTED	Сбой из-за слишком большой задержки или по другой причине
WSAECONNRESET	Сброс соединения удаленным узлом

Список кодов ошибок, которые могут возникать при вызове функции *recv()*:

Код ошибки	Описание
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию WSAStartup
WSAENETDOWN	Сбой в сети
WSAENOTCONN	Сокет не подсоединен

WSAEINTR	Работа функции была отменена при помощи функцииWSACancelBlockingCall
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса Windows Sockets
WSAENOTSOCK	Указанный в параметре дескриптор не является сокетом
WSAESHUTDOWN	Сокет был закрыт функцией shutdown
WSAEWOULDBLOCK	Сокет отмечен как неблокирующий, но запрошенная операция приведет к блокировке
WSAEMSGSIZE	Размер пакета данных превышает размер буфера, в результате чего принятый пакет был обрезан
WSAEINVAL	Сокет не был подключен функцией bind
WSAECONNABORTED	Сбой из-за слишком большой задержки или по другой причине
WSAECONNRESET	Сброс соединения удаленным узлом

ПРИЛОЖЕНИЕ Б
Процедуры для преобразования сетевого порядка следования байтов

Передача от одного вычислительного комплекса к другому символьной информации, как правило (когда один символ занимает один байт), не вызывает проблем. Однако для числовой информации ситуация усложняется. Разные типы компьютеров отличаются между собой способом хранения в памяти величин целого типа, а в семействе протоколов TCP/IP определен независимый порядок следования байтов. Поэтому в API-интерфейсе сокетов предусмотрены четыре библиотечные функции, которые обеспечивают преобразование порядка следования байтов локальной машины в принятый в сети стандарт, и наоборот. Для обеспечения мобильности программ, они должны быть написаны таким образом, чтобы каждый раз при копировании целого значения из локальной машины в сетевой пакет, или наоборот, вызывались процедуры преобразования порядка следования байтов.

Как известно, порядок байт в целых числах, представление которых занимает более одного байта, может быть для различных компьютеров неодинаковым. Есть вычислительные системы, в которых старший байт числа имеет меньший адрес, чем младший байт (**big-endian byte order**), а есть вычислительные системы, в которых старший байт числа имеет больший адрес, чем младший байт (**little-endian byte order**). При передаче целой числовой информации от машины, имеющей один порядок байт, к машине с другим порядком байт мы можем неправильно истолковать принятую информацию. Для того чтобы этого не произошло, было введено понятие **сетевого порядка байт**, т.е. порядка байт, в котором должна представляться целая числовая информация в процессе передачи ее по сети (на самом деле – это big-endian byte order). Целые числовые данные из представления, принятого на компьютере-отправителе, переводятся пользовательским процессом в сетевой порядок байт, в таком виде путешествуют по сети и переводятся в нужный порядок байт на машине-получателе процессом, которому они предназначены. Для перевода целых чисел из машинного представления в сетевое и обратно используется четыре функции: htons() , htonl() , ntohs() , ntohl(). Все четыре функции возвращают значение переданного им единственного параметра с перегруппированными байтами.

Например, чтобы преобразовать короткое (2-байтовое) целое число из принятого в сети порядка следования байтов в порядок байтов, установленный на локальном узле сети, программист должен вызвать функцию ntohs (network to host short). Эта функция имеет следующий синтаксис:

```
локальное-значение = ntohs(сетевое-значение)
```

Параметр сетевое-значение представляет собой 2-байтовое (16-битовое) целое число, представленное в принятом в сети порядке следования байтов, а возвращаемый параметр локальное-значение представлен в порядке байтов, который установлен для локального узла сети.

В языке программирования С 4-байтовые (32-битовые) целые числа имеют тип long и называются длинными целыми. Функция ntohl (network to host long) преобразует 4-байтовые длинные целые числа из принятого в сети порядка байтов в порядок байтов, установленный для локального узла. При вызове функции ntohl ей в качестве параметра передается длинное целое число, представленное в сетевом порядке следования байтов:

```
локальное-значение = ntohl(сетевое-значение)
```

Две аналогичные функции позволяют программисту преобразовать порядок байтов целого числа, который принят на локальном узле сети, в сетевой порядок. Функция htons преобразует 2-байтовое (короткое) целое число в сетевой порядок следования байтов. При вызове функции htons ей в качестве параметра передается короткое целое число:

```
сетевое-значение = htons(локальное-значение)
```

Функция htonl, преобразует длинные целые числа в принятый в сети порядок следования байтов:

```
сетевое-значение = htonl(локальное-значение)
```

Очевидно, что в подпрограммах преобразования должны сохраняться следующие математические соотношения:

сетевое-значение = htons(ntohs(сетевое-значение))
и
локальное-значение = ntohs(htons(локальное-значение))

Похожие соотношения сохраняются и для подпрограмм преобразования длинных целых чисел.

Для чисел с плавающей точкой на разных машинах могут различаться не только порядок байт, но и форма представления такого числа. Простых функций для их корректной передачи по сети не существует. Если требуется обмениваться действительными данными, то либо это нужно делать на гомогенной сети, состоящей из одинаковых компьютеров, либо использовать символьные и целые данные для передачи действительных значений.

Подпрограммы обработки IP-адресов

Поскольку во многих приложениях требуется выполнить преобразование 32-битового IP-адреса в точечный десятичный формат, и наоборот, в библиотеку сокетов входят соответствующие процедуры, выполняющие такое преобразование. Процедуры inet_addr и inet_network преобразуют IP-адрес из точечного десятичного формата в 32-битовое целое число, представленное в сетевом порядке следования байтов. Процедура inet_addr формирует 32-битовый IP-адрес узла сети, а процедура inet_network формирует 32-битовый IP-адрес сети, в котором биты, относящиеся к узлу сети, заменены нулями. Процедуры имеют следующий синтаксис:

IP-адрес = inet_addr(строка)
и
IP-адрес = inet_network(строка)

Здесь параметр строка задает адрес ASCII-строки, содержащей IP-адрес, представленный в точечной десятичной форме записи. В этой форме IP-адрес может иметь от 1 до 4 сегментов цифр, разделенных точками. Если представление состоит из четырех сегментов, то каждый из них соответствует одному байту полученного в результате 32-битового целого числа. Если сегментов меньше четырех, в оставшиеся байты помещается значение последнего сегмента.

Процедура inet_ntoa осуществляет обратную процедуре inet_addr операцию — преобразует 32-х битовое целое число в ASCII-строку, содержащую IP-адрес, представленный в точечной десятичной форме записи. Эта процедура имеет следующий синтаксис:

строка = inet_ntoa(IP-адрес)

Здесь параметр IP-адрес — 32-битовый IP-адрес, представленный в сетевом порядке следования байтов, а параметр строка — адрес буфера, куда будет помещена сформированная ASCII-строка.

Часто обрабатывающие IP-адреса программы должны объединить адрес сети с локальным адресом узла и сформировать единый IP-адрес. Такое объединение выполняет процедура inet_makeaddr. Она имеет следующий синтаксис:

IP-адрес = inet_makeaddr(адрес-сети, адрес-узла)

Параметр адрес-сети представляет собой 32-битовый IP-адрес сети, заданный в локальном порядке следования байтов, а параметр адрес-узла — целое число, представляющее адрес локального узла данной сети, выраженный в локальном порядке следования байтов.

Процедуры inet_netof и inet_lnaof выполняют операцию обратную процедуре inet_makeaddr, разделяя сетевую и локальную часть IP-адреса. Они имеют следующий синтаксис:

адрес-сети = inet_netof(IP-адрес)
и
адрес-узла = inet_lnaof(IP-адрес),

Здесь параметр IP-адрес представляет собой 32-битовый IP-адрес, заданный в сетевом порядке следования байтов, а полученные значения возвращаются в локальном порядке следования байтов.