

Socket_definition

Сокет

абстрактная структура данных, используемая
для создания точки присоединения канала обмена данными
между процессами.

Socket_definition

Socket - это конечная точка сетевых коммуникаций, является чем-то вроде "портала", через который можно отправлять байты во внешний мир.

Приложение просто пишет данные в сокет; их дальнейшая буферизация, отправка и транспортировка осуществляется используемым стеком протоколов и сетевой аппаратурой.

Чтение данных из сокета происходит аналогичным образом.

Socket_description

Socket API был впервые реализован в операционной системе Berkley UNIX, а сейчас доступен практически в любой модификации Unix/Linux. Изначально сокет использовались в программах на C/C++. В настоящее время доступны для Perl, Java и др.

Наиболее часто сокет используются для работы в IP-сетях. В этом случае их можно использовать для взаимодействия приложений не только по специально разработанным, но и по стандартным протоколам - HTTP, FTP, Telnet и т. д. Например, вы можете написать собственный Web-браузер или Web-сервер, способный обслуживать одновременно множество клиентов.

Socket_attributes

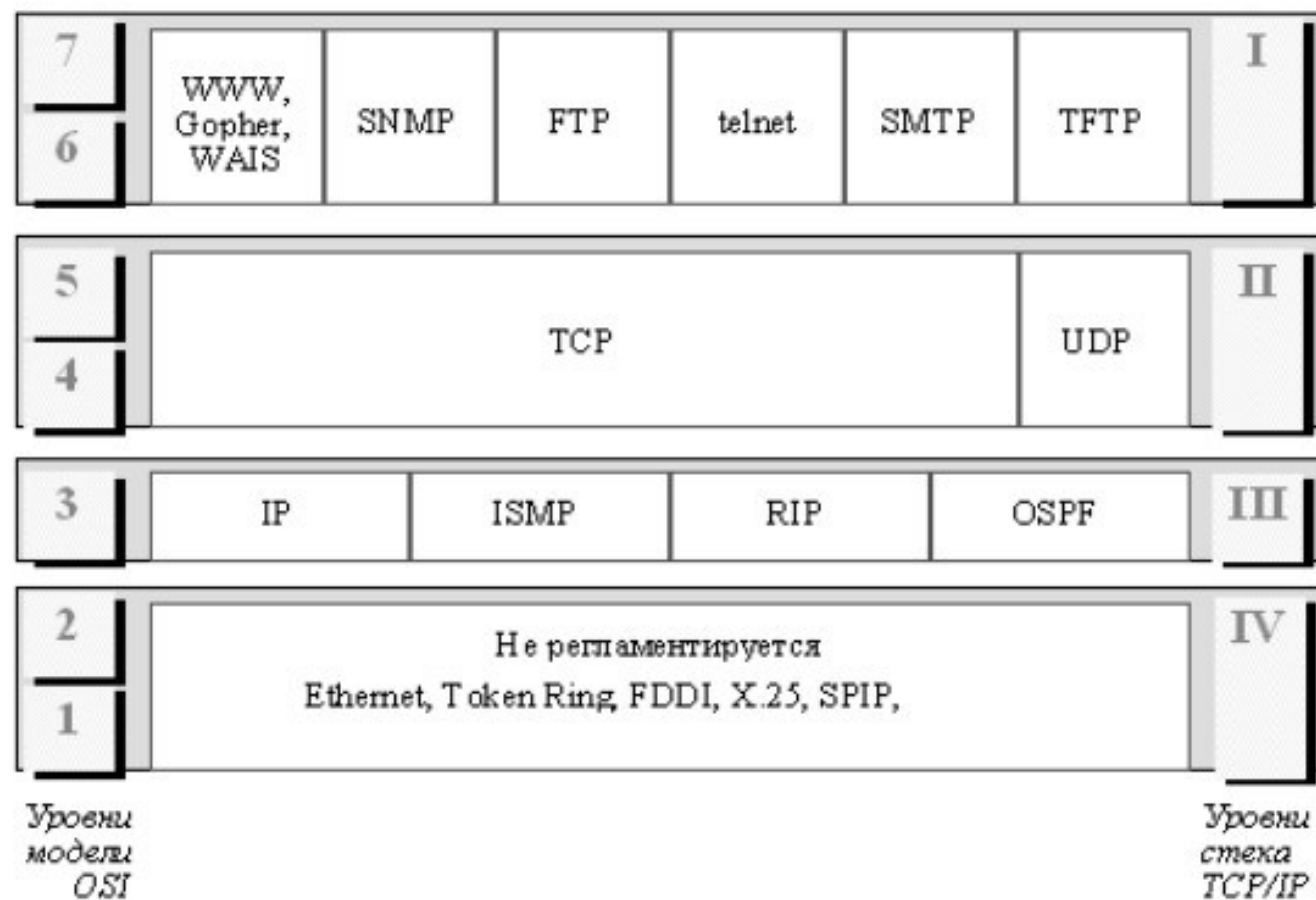
Сокет в программе идентифицируется **дескриптором**, получаемым от операционной системы при его создании.

По типу организации канала различают сокеты, ориентированные на соединение (**stream sockets**),

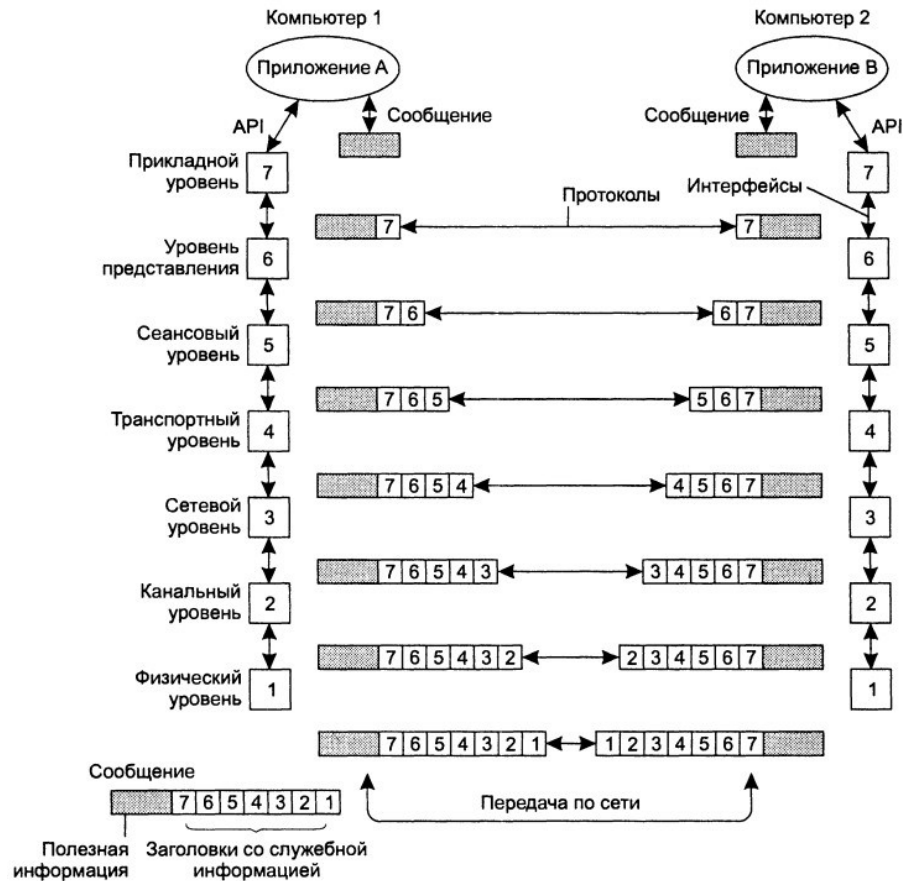
и сокеты не подразумевающие наличия постоянного соединения (**datagram sockets**).

По умолчанию, первая разновидность предполагает интерфейс к транспортному уровню, реализованному на **TCP** (Transmission Control Protocol) протоколе, а вторая - на протоколе **UDP** (User Datagram Protocol).

TCP/IP protocol stack

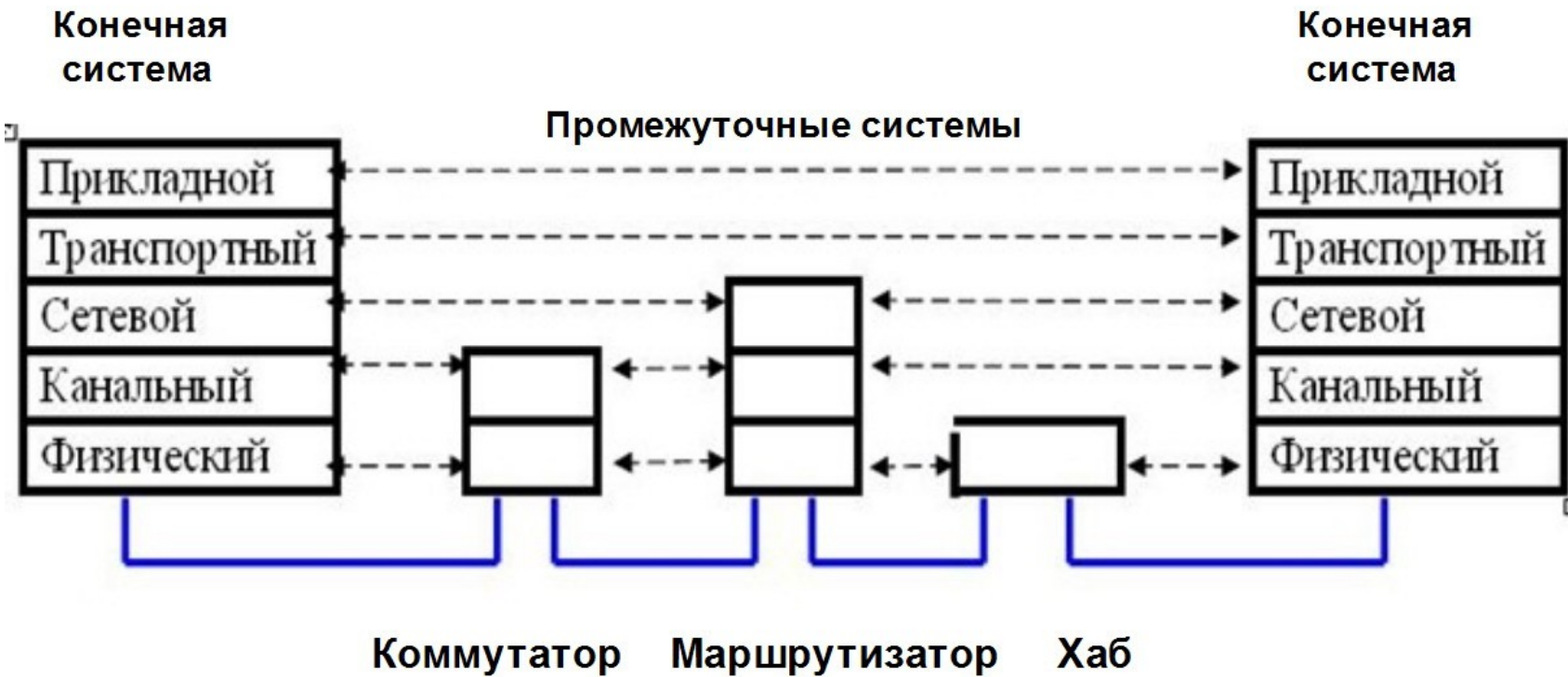


OSI_model packets incapsulation

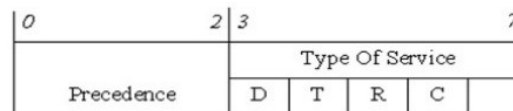
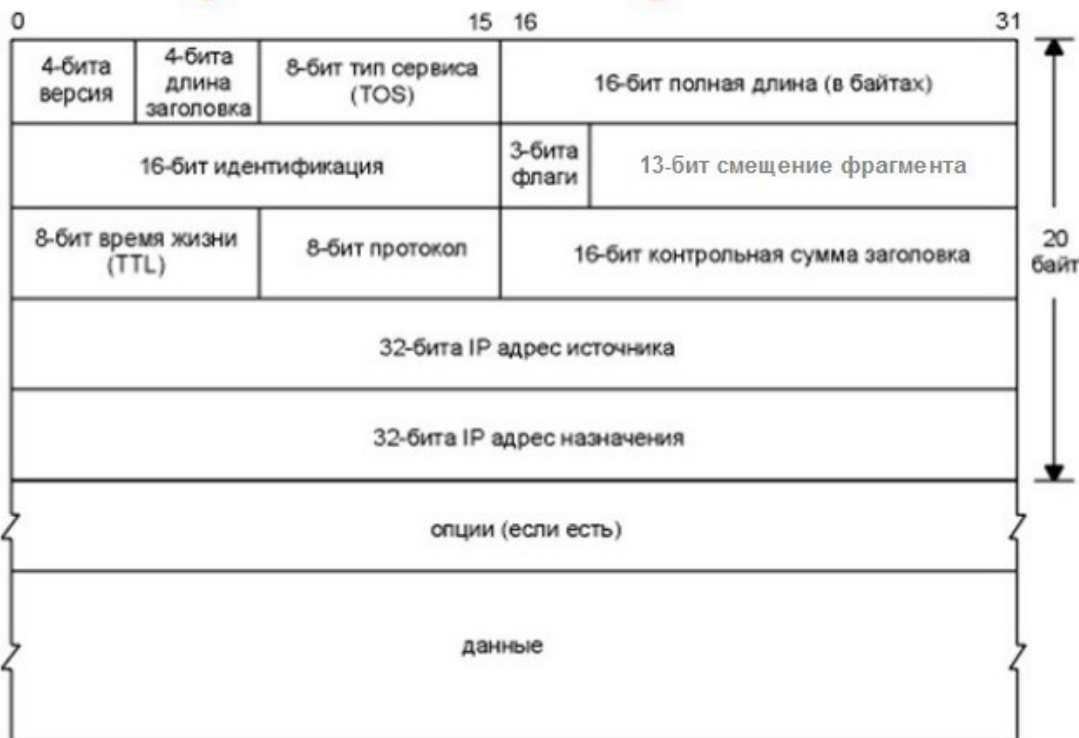


Protocol stack levels distribution


Протокольные сущности протоколов транспортного уровня TCP и UDP, как и протоколы прикладного уровня, устанавливаются только на конечных узлах.



IP protocol packet



Три младших бита ("Precedence") определяют приоритет дейтаграммы:

- 111 - управление сетью (максимальный приоритет)
- 110 - 
- 101 -
- 100 -
- 011 -
- 010 -
- 001 -
- 000 - обычная передача (минимальный приоритет)

Биты D, T, R, C определяют желаемый тип маршрутизации:

- D (Delay) - минимальная задержка,
- T (Throughput) - максимальная пропускная способность,
- R (Reliability) - максимальная надежность,
- C (Cost) - минимальная стоимость.

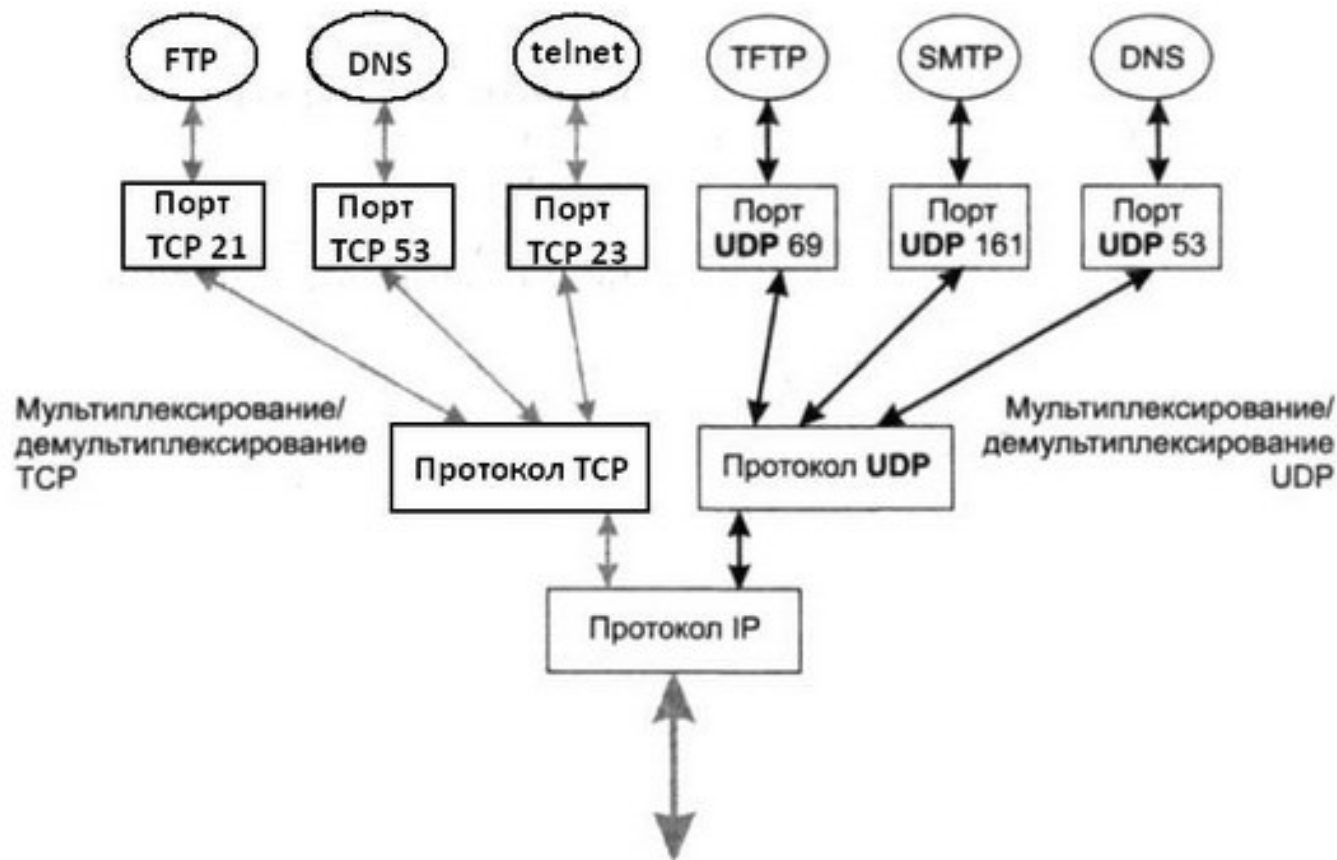
В дейтаграмме может быть установлен только один из битов D, T, R, C. Старший бит байта не используется.

Multiplexing / demultiplexing

После того, как пакет средствами протокола IP доставлен на сетевой интерфейс компьютера-получателя данные необходимо переправить конкретному процессу-получателю. Процедура распределения протоколами TCP и UDP **поступающих от сетевого уровня** пакетов между прикладными процессами называется **демультиплексированием**.

Обратная задача на передающем узле – передача данных от приложений к общему протокольному модулю IP для отправки в сеть - **мультиплексирование**.

Multiplexing / demultiplexing



Ports

Протоколы TCP и UDP ведут для каждого приложения две системные очереди:

- очередь данных, поступающих к приложению из сети, и
- очередь данных, отправляемых этим приложением в сеть.

Множество пакетов, поступающих на транспортный уровень, организуются операционной системой в виде множества очередей к точкам входа различных прикладных процессов.

В терминологии TCP IP такие системные очереди называются **портами**.

Ports numbers

Входная и выходная очереди одного приложения рассматриваются как один порт **port**.

Для однозначной идентификации им присваивают номера. Номера портов используются для адресации приложений.

Номера из диапазона от **0** до **1023** называются назначенными, являются уникальными в пределах Интернета и закрепляются за приложениями **централизованно**.

Номера из диапазона 1024 до 65 535 могут назначаться **локально** разработчиками собственных приложений или операционной системой в ответ на поступление запроса от приложения. На каждом компьютере операционная система ведет список занятых и свободных номеров портов.

Ports UDP / Ports TCP

Никакой связи между назначенными номерами TCP- и UDP-портов нет.

Даже если номера TCP- и UDP-портов совпадают, они идентифицируют разные приложения.

Например, одному приложению может быть назначен TCP - порт 1750, а другому — UDP - порт 1750.

Номера портов могут назначаться как статически, так и выделяться операционной системой динамически. Динамические номера уникальны в пределах одного хоста.

UDP protocol

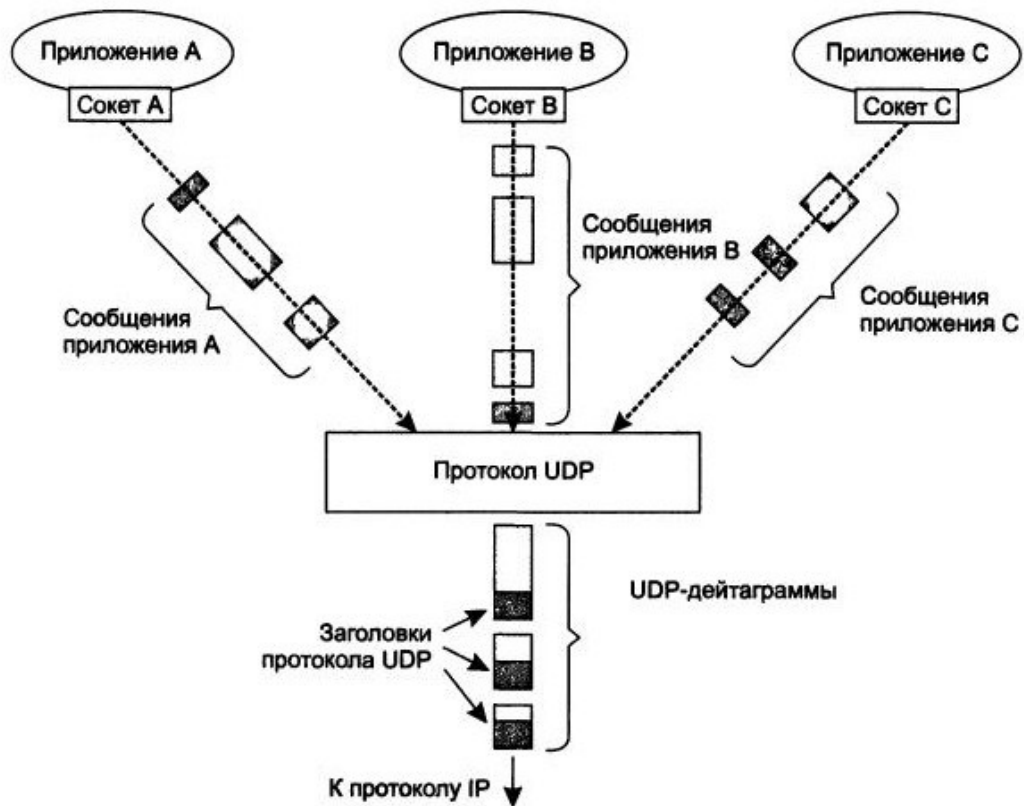
Протокол UDP достаточно прост. Его функции сводятся к простой передаче данных между прикладным и сетевым уровнями, а также примитивному контролю искажений в передаваемых данных.

Каждая дейтаграмма переносит *отдельное* пользовательское сообщение.

При контроле искажений протокол UDP только **диагностирует,**

но не исправляет ошибку. Если контрольная сумма показывает, что в поле данных UDP-дейтаграммы произошла ошибка, протокол UDP просто отбрасывает поврежденную дейтаграмму.

UDP protocol multiplexing



UDP protocol header format

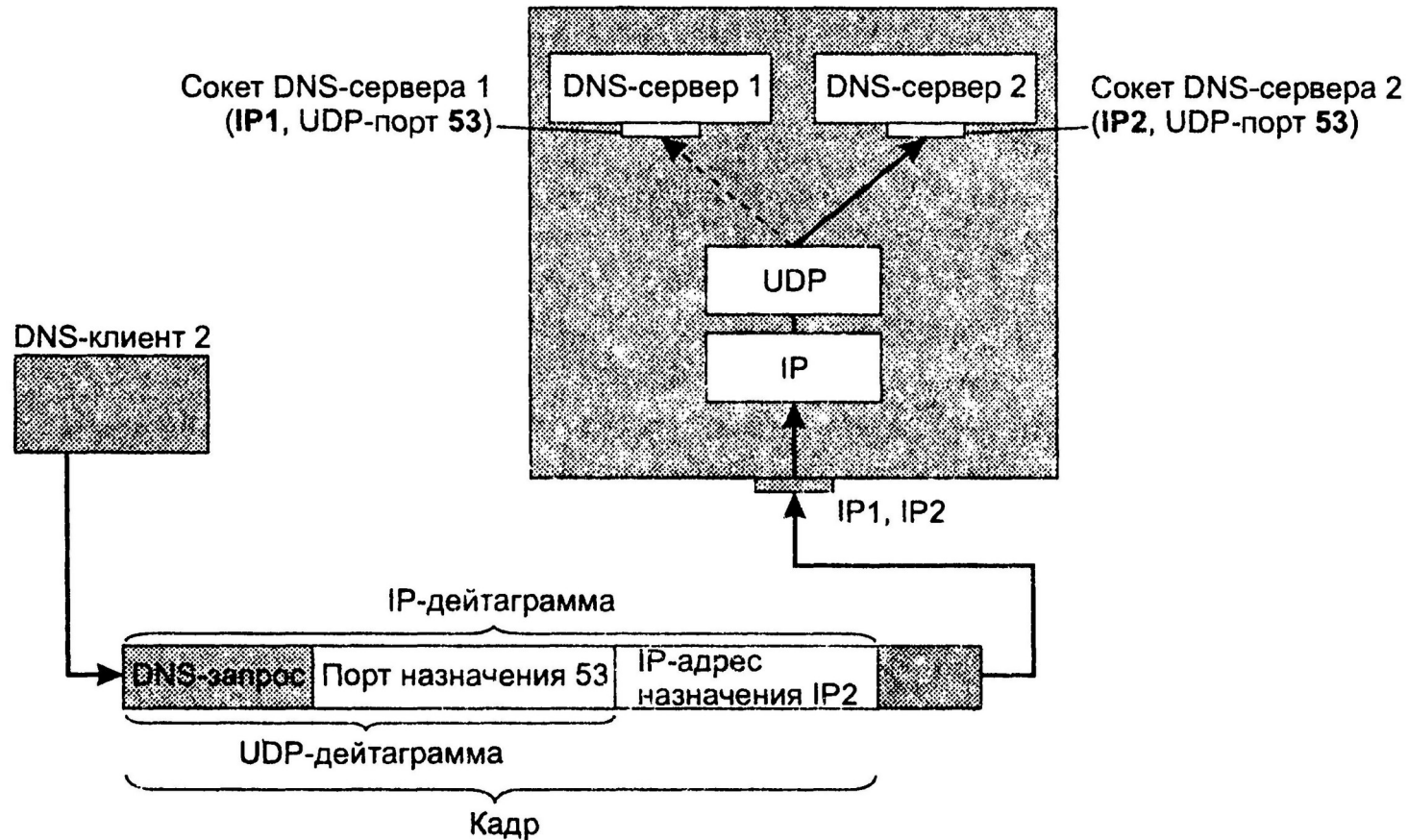
Заголовок UDP состоит из четырех 2-байтных полей:

- номер UDP-порта отправителя;
- номер UDP-порта получателя;
- контрольная сумма;
- и длина дейтаграммы.

Пример заголовка UDP с заполненными полями:

Source Port	= 0x0035
Destination Port	= 0x0411
Total Length	= 132 (0x84) bytes
Checksum	= 0x5333

UDP protocol demultiplexing



UDP protocol socket

На хосте-получателе протокол UDP решает задачу демультиплексирования данных.

Принимает от протокола IP извлеченные из пакетов UDP-дейтаграммы, а из заголовка IP-пакета *IP-адрес назначения* и из UDP-заголовка - *номер порта*.

Они используются для определения **UDP-сокета**, однозначно идентифицирующего приложение, которому направлены поступившие данные.

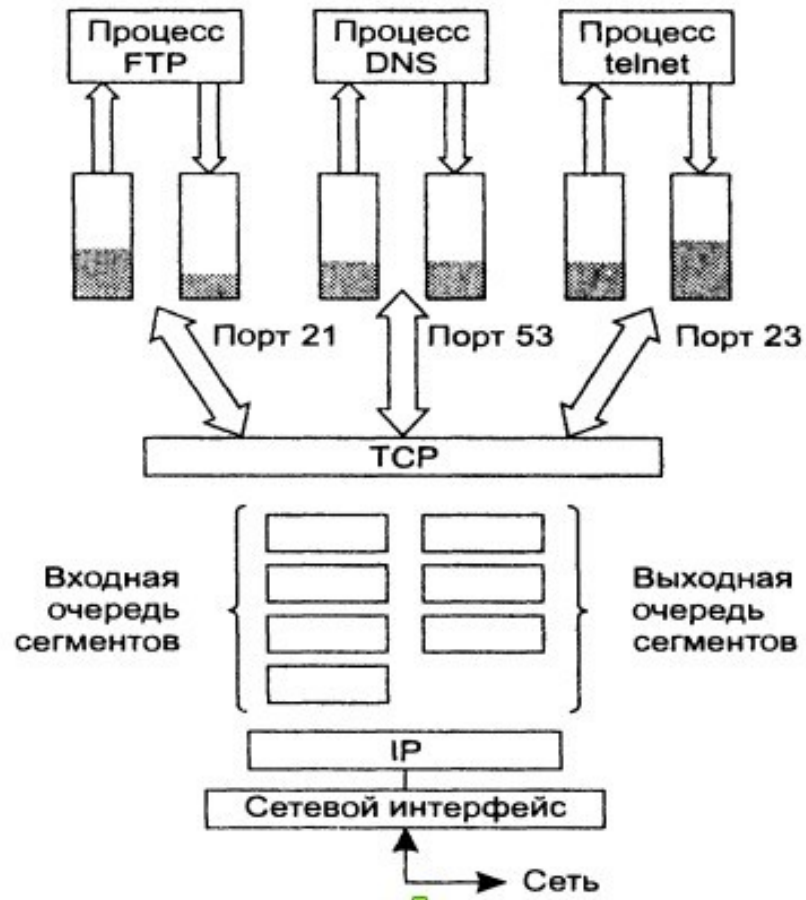
TCP protocol

Поступающая к протоколу TCP от протоколов более высокого уровня информация (от прикладных процессов), рассматривается протоколом TCP как *неструктурированный поток байтов*.

Поступающие данные буферизуются средствами TCP. Для передачи на сетевой уровень из буфера «вырезается» некоторая непрерывная часть данных, которая называется **сегментом** и снабжается *заголовком*.

Заголовок TCP-сегмента содержит значительно больше полей, чем заголовок UDP дейтаграммы, что отражает более развитые возможности протокола TCP (создание логического соединения, гарантированная доставка).

TCP segments from bytes stream



TCP segment header

2 байта		2 байта	
Порт источника (source port)		Порт приемника (destination port)	
Последовательный номер (sequence number) - номер первого байта данных в сегменте, определяет смещение сегмента относительно потока отправляемых данных			
Подтвержденный номер (acknowledgement number) - максимальный номер байта в полученном сегменте, увеличенный на единицу			
Длина заголовка (hlen)	Резерв (reserved)	URG ACK PSH RST SYN FIN	Окно (window) - количество байтов данных, ожидаемых отправителем данного сегмента, начиная с байта, номер которого указан в поле подтвержденного номера
Контрольная сумма (checksum)			Указатель срочности (urgent pointer) - указывает на конец данных, которые необходимо срочно принять, несмотря на переполнение буфера
Параметры (options) - это поле имеет переменную длину и может вообще отсутствовать, используется для решения вспомогательных задач, например, для согласования максимального размера сегмента			
Заполнитель (padding) - это фиктивное поле может иметь переменную длину, используется для доведения размера заголовков до целого числа 32-битовых слов			

TCP segment header description

Поля заголовка TCP сегмента:

- **Порт источника** (source port) занимает 2 байта и идентифицирует процесс-отправитель.
- **Порт приемника** (destination port) занимает 2 байта и идентифицирует процесс-получатель.
- **Последовательный номер** (sequence number) занимает 4 байта и представляет собой номер байта, который определяет смещение сегмента относительно потока отправляемых данных (то есть номер первого байта данных в сегменте).
- **Подтвержденный номер** (acknowledgement number) занимает 4 байта и содержит максимальный номер байта в полученном сегменте, увеличенный на единицу. Это значение используется в качестве квитанции при передаче данных с квитированием. Если установлен контрольный бит АКС, то это поле содержит следующий номер очереди, который отправитель данного сегмента желает получить в обратном направлении.
- **Длина заголовка** (hlen) занимает 4 бита и представляет собой длину заголовка TCP-сегмента, измеренную в 32-х битовых словах. Длина заголовка не фиксирована и может изменяться в зависимости от значений, устанавливаемых в поле параметров.
- **Резерв** (reserve) занимает 6 бит.

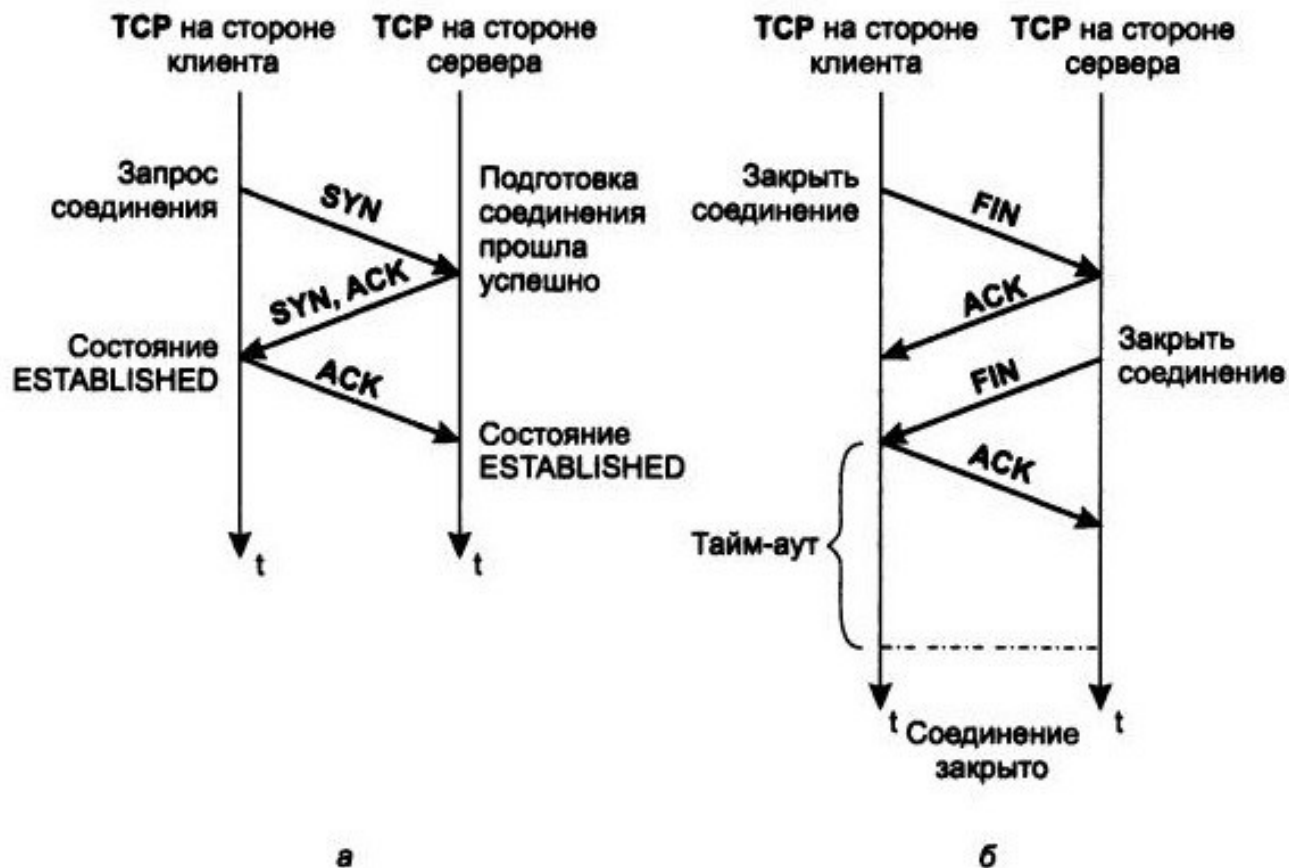
TCP segment header description

- **Кодовые биты** (code bits) 6 флагов содержат служебную информацию о типе данного сегмента. Положительное значение сигнализируется установкой этих битов в единицу:
- **URG** — срочное сообщение;
- **ACK** — квитанция на принятый сегмент;
- **PSH** — запрос на отправку сообщения без ожидания заполнения буфера (протокол TCP может выжидать заполнения буфера перед отправкой сегмента, но если требуется срочная передача, то приложение сообщает об этом протоколу TCP с помощью данного бита);
- **RST** — запрос на восстановление соединения;
- **SYN** — сообщение, используемое для синхронизации счетчиков переданных данных при установлении соединения;
- **FIN** — признак достижения передающей стороной последнего байта в потоке передаваемых данных.

TCP segment header description

- **Окно** (window) занимает 2 байта и задает количество байтов данных, ожидаемых отправителем данного сегмента, начиная с байта, номер которого указан в поле подтвержденного номера.
- **Контрольная сумма** (checksum) занимает 2 байта.
- **Указатель срочности** (urgent pointer) занимает 2 байта и указывает на конец данных, которые необходимо срочно принять, несмотря на переполнение буфера. Указатель срочности используется совместно с кодовым битом URG. То есть, если какие-то данные необходимо переслать приложению-получателю вне очереди, то приложение-отправитель должно сообщить об этом протоколу TCP путем установки в единицу бита URG
- **Параметры** (options) имеют переменную длину и могут вообще отсутствовать. Максимальная величина поля составляет 3 байта. Оно используется для решения вспомогательных задач, например, для выбора максимального размера сегмента. Поле параметров может располагаться в конце заголовка TCP, а его длина кратна 8 бит.
- **Заполнитель** (padding) может иметь переменную длину. Это фиктивное поле, используемое для доведения размера заголовка до целого числа 32-х битовых слов.

TCP logical connection establishment



TCP logical connection establishment

Получив запрос с флагом SYN, **модуль TCP** на стороне сервера пытается создать «инфраструктуру» для обслуживания нового клиента. Он обращается к операционной системе с просьбой о выделении определенных системных ресурсов для организации буферов, таймеров, счетчиков.

Выделяемые ОС ресурсы закрепляются за соединением с момента создания и до момента разрыва.

Если на стороне сервера все необходимые ресурсы были получены и все необходимые действия выполнены, то модуль TCP посылает клиенту сегмент с флагами ACK и SYN.

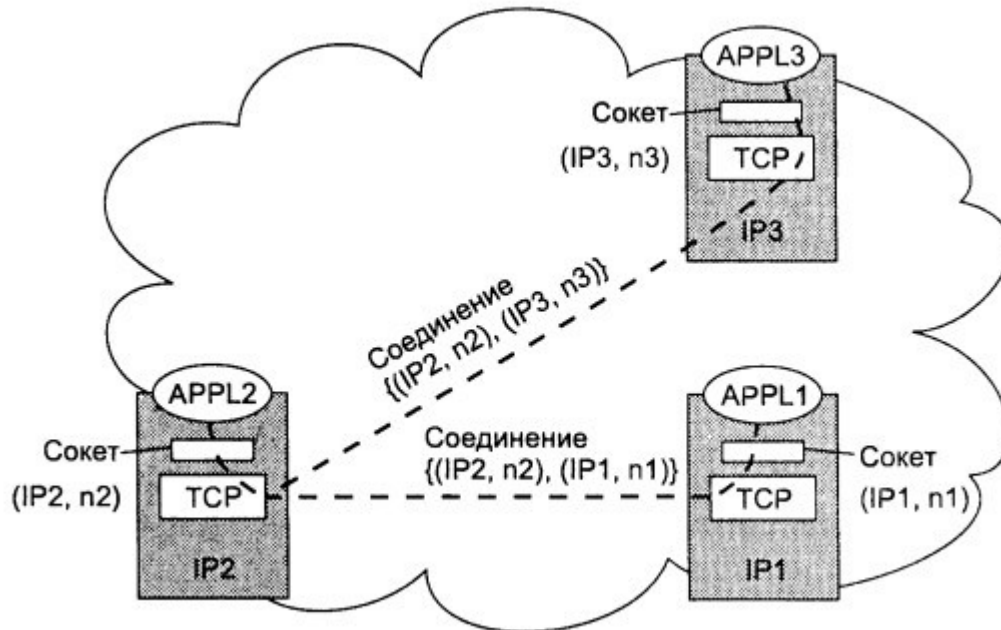
TCP logical connection

При установлении логического соединения модули TCP каждой из сторон договариваются между собой о следующих параметрах процедуры обмена данными:

- максимальный размер сегмента, который сторона готова принимать;
- максимальный объем данных (возможно несколько сегментов), которые она разрешает другой стороне передавать в свою сторону, даже если та еще не получила квитанцию на предыдущую порцию данных (размер окна *slide window*);
- начальный порядковый номер байта, с которого она начинает отсчет потока данных в рамках данного соединения.

TCP logical connection

Логическое TCP соединение однозначно идентифицируется парой **сокетов**.
Причем каждый сокет может одновременно участвовать в нескольких соединениях. Приложение 2 установило два логических соединения (с приложением 1 и с приложением 3).



Sockets address families

Существует две схемы, называемые *address families* (семейства адресации):

- ✓ **Internet domain** addressing. Эта схема базируется на 32-битном *IP* адресе хоста, на котором адресуемый сокет (процесс создавший сокет) находится, и на 16-битном *номере порта*, идентифицирующем процесс на хосте. Адрес сокета в этом случае ассоциируется с парой IP address/Port number.
- ✓ **Unix domain** address family, подразумевает, что сокет ассоциируется с файлом, имя и путь к которому попадает в системный список каталогов и помечается символом **s** (socket). Клиент и сервер обязаны находиться на одной и той же машине (улучшенный вариант *pipe*, но *full-duplex*).

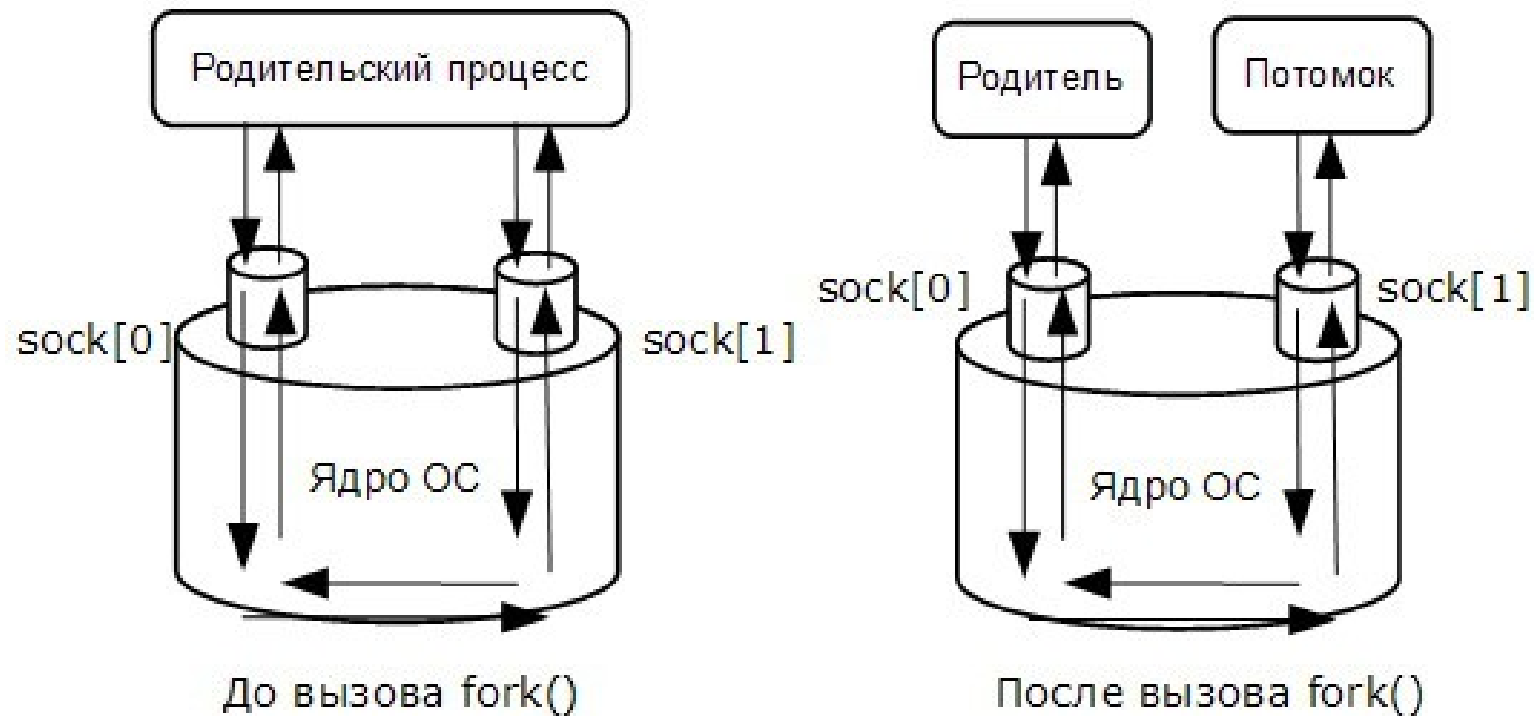
Sockets `socketpair()`

Простейший способ организации обмена на сокетах базируется на системном вызове **`socketpair()`**.

Ограничение **`socketpair()`** - процессы на одном хосте и родственные

- в 1-ом параметре `socketpair()` задается семейство адресации,
- 2-й параметр указывает на тип (`SOCK_STREAM` либо `SOCK_DGRAM`) создаваемого сокета,
- 3-й параметр конкретизирует тип протокола заданного семейства адресации (0 - протокол выбирается самой ОС),
- 4-й параметр указывает на двухэлементный целый массив, который заполняется **дескрипторами** создаваемой пары сокетов (при успешном вызове).

Sockets socketpair chart



Sockets socketpair program

```
/* Программа socketpair.cpp */  
/* Создает пару сокетов, запускает дочерний процесс и  
* использует пару сокетов для обмена информацией  
* между родительским и дочерним процессами.  
* Компилировать с опцией -lsocket  
*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <sys/types>  
#include <sys/socket.h>  
#define BUF_SZ 10
```


Sockets socketpair program

```
main(void){
    int
    sock[2];
    int
    cpid, i;
    static char buf[BUF_SZ];
    if (socketpair(PF_UNIX, SOCK_STREAM, 0, SOCK)<0){
        perror("Generation error");
        exit(1);
    }
    switch (cpid = (int) fork()){
        case -1:
            perror("Bad fork");
            exit(2);
        case 0:
            /* Дочерний процесс */
            close(sock[1]);
            for (i=0; i<10; i+=2){
                sleep(1);
                sprintf(buf, "c:%d\n", i);
                write(sock[0], buf, sizeof(buf));
                read(sock[0], buf, BUF_SZ);
                printf("c->%s", buf); /* Сообщение от родительского процесса */
            }
            close(sock[0]);
            break;
    }
```

Sockets socketpair program

```
default:
/* Родительский процесс */
close(sock[0]);
for(i=1; i<10; i+=2){
sleep(1);
read(sock[1], buf, BUF_SZ);
printf("p->%s", buf);
/* Сообщение от дочернего процесса */
sprintf(buf, "p: %d\n", i);
write(sock[1], buf, sizeof(buf));
}
close(sock[1]);
}
return 0;
}
```

Thanks for your attention

Спасибо за внимание !

vladimir.shmakov.2012@gmail.com