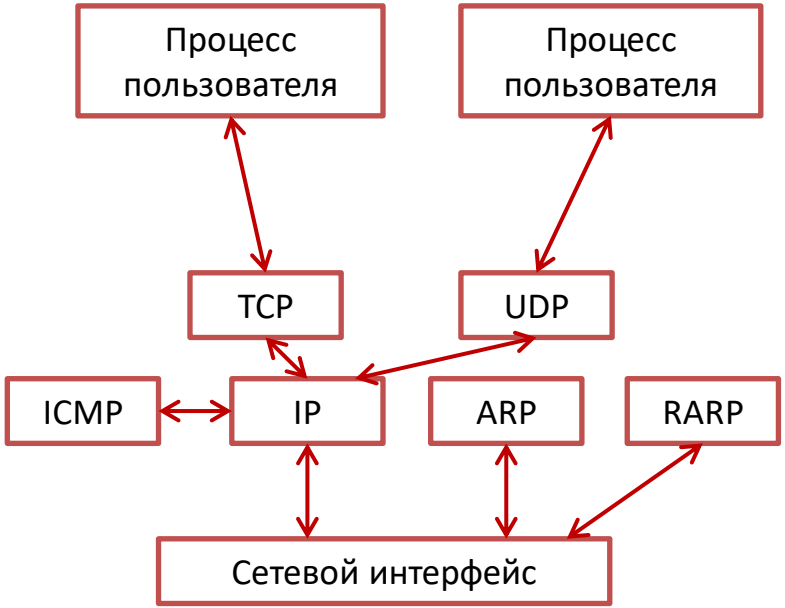
Сетевые протоколы

Модель OSI/ISO

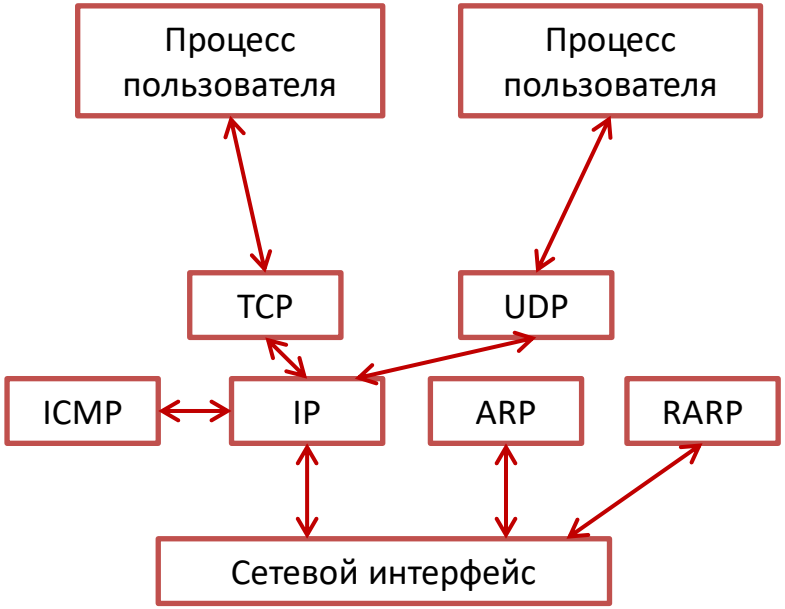
- 7 Уровень приложений
- 6 Уровень представления данных
- 5 Сеансовый уровень
- 4 Транспортный уровень
- 3 Сетевой уровень
- 2 Канальный уровень
- 1 Физический уровень

Модель TCP/IP

- 4 Уровень приложений / процессов
- 3 Транспортный уровень
- 2 Сетевой уровень
- 1 Уровень сетевого интерфейса



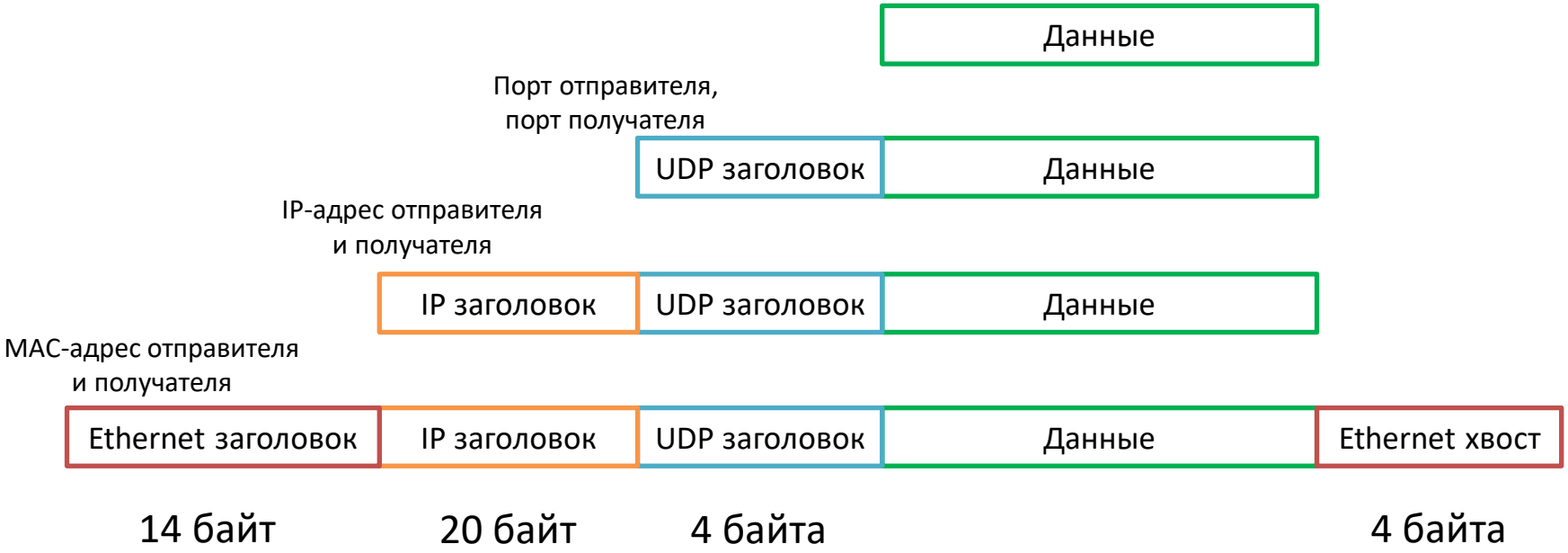
Сетевые протоколы



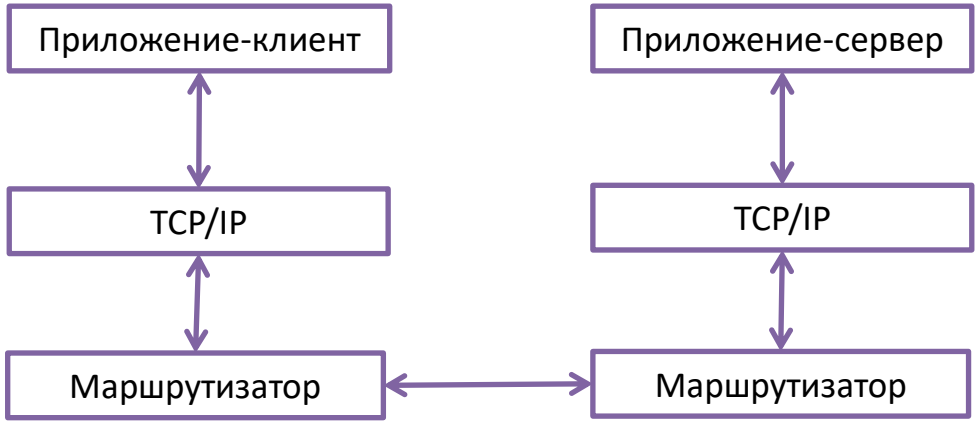
- **ICMP** – Internet Control Message Protocol. Протокол обработки ошибок и обмена управляющей информацией между узлами сети.
- **IP** – Internet Protocol. Это протокол, который обеспечивает доставку пакетов информации для протокола ICMP и протоколов транспортного уровня TCP и UDP.
- **ARP** – Address Resolution Protocol. Это протокол для отображения адресов уровня Internet в адреса уровня сетевого интерфейса.
- **RARP** – Reverse Address Resolution Protocol. Этот протокол служит для решения обратной задачи: отображения адресов уровня сетевого интерфейса в адреса уровня Internet.

Сетевые протоколы

Инкапсуляция для UDP в Ethernet



Взаимодействие клиента и сервера



- Сервер, как правило, работает постоянно, на всем протяжении жизни приложения, а клиенты могут работать эпизодически.
- Сервер ждет запроса от клиентов, инициатором же взаимодействия выступает клиент.
- Как правило, клиент обращается к одному серверу за раз, в то время как к серверу могут одновременно поступить запросы от нескольких клиентов.
- Клиент должен знать полный адрес сервера перед началом организации запроса, в то время как сервер может получить информацию о полном адресе клиента из пришедшего запроса.
- И клиент, и сервер должны использовать один и тот же протокол транспортного уровня.

Сокеты

- Создание сокета
```
#include <sys/socket.h>
int socket(int socket_family, int socket_type, int protocol);
```

- Семейство протоколов socket_family:
  - PF_UNIX (AF_UNIX), **PF_LOCAL (AF_LOCAL)** – Локальное соединение
  - **PF_INET (AF_INET)** – IPv4 протоколы Интернет
  - **PF_INET6** – IPv6 протоколы Интернет
  - PF_IPX – IPX - протоколы Novell
  - PF_NETLINK – Устройство для взаимодействия с ядром
  - PF_X25 – Протокол ITU-T X.25 / ISO-8208
  - PF_AX25 – Протокол AX.25 - любительское радио
  - PF_ATMPVC – асинхронный режим, доступ к низкоуровневым PVC
  - PF_APPLETALK – Appletalk
  - PF_PACKET – Низкоуровневый пакетный интерфейс

Сокеты

- **Создание сокета**

```
#include <sys/socket.h>
int socket(int socket_family, int socket_type, int protocol);
```

- Тип взаимодействия `socket_type`:
  - **SOCK_STREAM** – ориентированное на установку соединения по протоколу TCP (или потоковое)
  - **SOCK_DGRAM** – рассчитанное на использование протокола UDP (дейтаграммное)
  - **SOCK_RAW** – использование сырых, неструктурированных сокетов
  - SOCK_SEQPACKET – последовательный двусторонний канал для передачи дейтаграмм с поддержкой соединений (длина дейтаграммы ограничена, получатель за один раз должен прочитать целый пакет)
  - SOCK_RDM – обеспечивает надежную доставку дейтаграмм без гарантии, что они будут расположены по порядку

- Протокол взаимодействия `protocol: IPPROTO_TCP / IPPROTO_UDP или 0, если семейство PF_INET.`

Сокеты

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
```

- `tcp_socket = socket(AF_INET, SOCK_STREAM, 0);`
- `udp_socket = socket(AF_INET, SOCK_DGRAM, 0);`
- `raw_socket = socket(AF_INET, SOCK_RAW, protocol);`

Сокеты

- **Связывание сокета**
```
int bind(int sockfd, struct sockaddr *my_addr,
socklen_t addrlen);
```

- Привязывает к сокету sockfd локальный адрес my_addr длиной addrlen (sizeof (sockaddr))

- Структура sockaddr
```
struct sockaddr {
  u_short sa_family;
  char sa_data[14] ;
};
```

- sa_data (Internet)
```
struct sockaddr_in {
  short sin_family; // AF_INET
  unsigned short sin_port;
  struct in_addr sin_addr; // IPv4
  char sin_zero[8];
} sa;
```

Сокеты

- sa_data (Internet)

```
struct sockaddr_in {
  short sin_family; // AF_INET
  unsigned short sin_port;
  struct in_addr sin_addr; // IPv4
  char sin_zero[8];
} sa;
```

Номер порта:

- с 1 по 1023 – могут назначать сокетам только процессы, работающие с привилегиями системного администратора.
- с 1024 по 49151 – зарезервированы за системными сетевыми службами независимо от вида используемой операционной системы (ICANN)
- с 49152 по 65535 – предназначены для процессов обычных пользователей.

Сокеты

```
struct in_addr {
  uint32_t      s_addr;     /* address in network byte order */
};
```

s_addr – конкретный адрес или: INADDR_LOOPBACK (127.0.0.1), INADDR_ANY (0.0.0.0), INADDR_BROADCAST (255.255.255.255)

```
sockaddr_in local_addr;
local_addr.sin_family = AF_INET;
local_addr.sin_port = htons(PORT);
local_addr.sin_addr.s_addr = INADDR_ANY; // подключения со всех IP-адресов
// local_addr.sin_addr = inet_addr("192.168.0.1");
bind(socket, (sockaddr *) &local_addr, sizeof(local_addr));
```

Сокеты

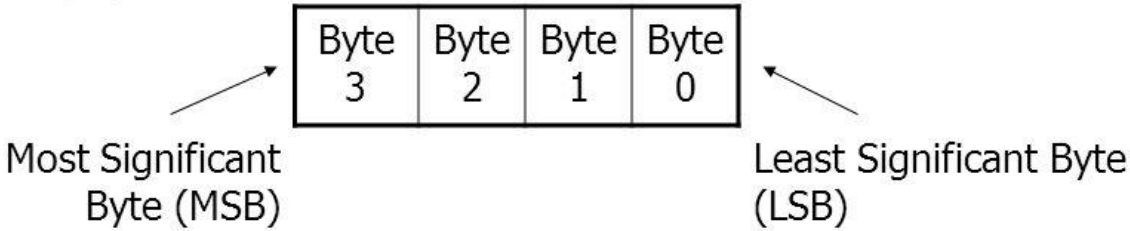
Перевод данных из узлового (host) порядка расположения байтов в сетевой (network) и наоборот:

```
#include <netinet/in.h>
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

В архитектуре i80x86 узловой порядок расположения байтов - это такой порядок, при котором последний значимый байт стоит в начале числа (little-endian), а при сетевом порядке расположения байтов, используемом в интернет, наоборот: первый значимый байт стоит в начале числа (big-endian).

Endian-ness

- Byte Ordering for Little Endian vs. Big Endian



Memory Address	+0	+1	+2	+3	
Big Endian	Byte 3	Byte 2	Byte 1	Byte 0	MSB in the lowest (first) memory address
Little Endian	Byte 0	Byte 1	Byte 2	Byte 3	LSB in the lowest (first) memory address

Сокеты

Преобразование адресов:

```
#include <arpa/inet.h>
```

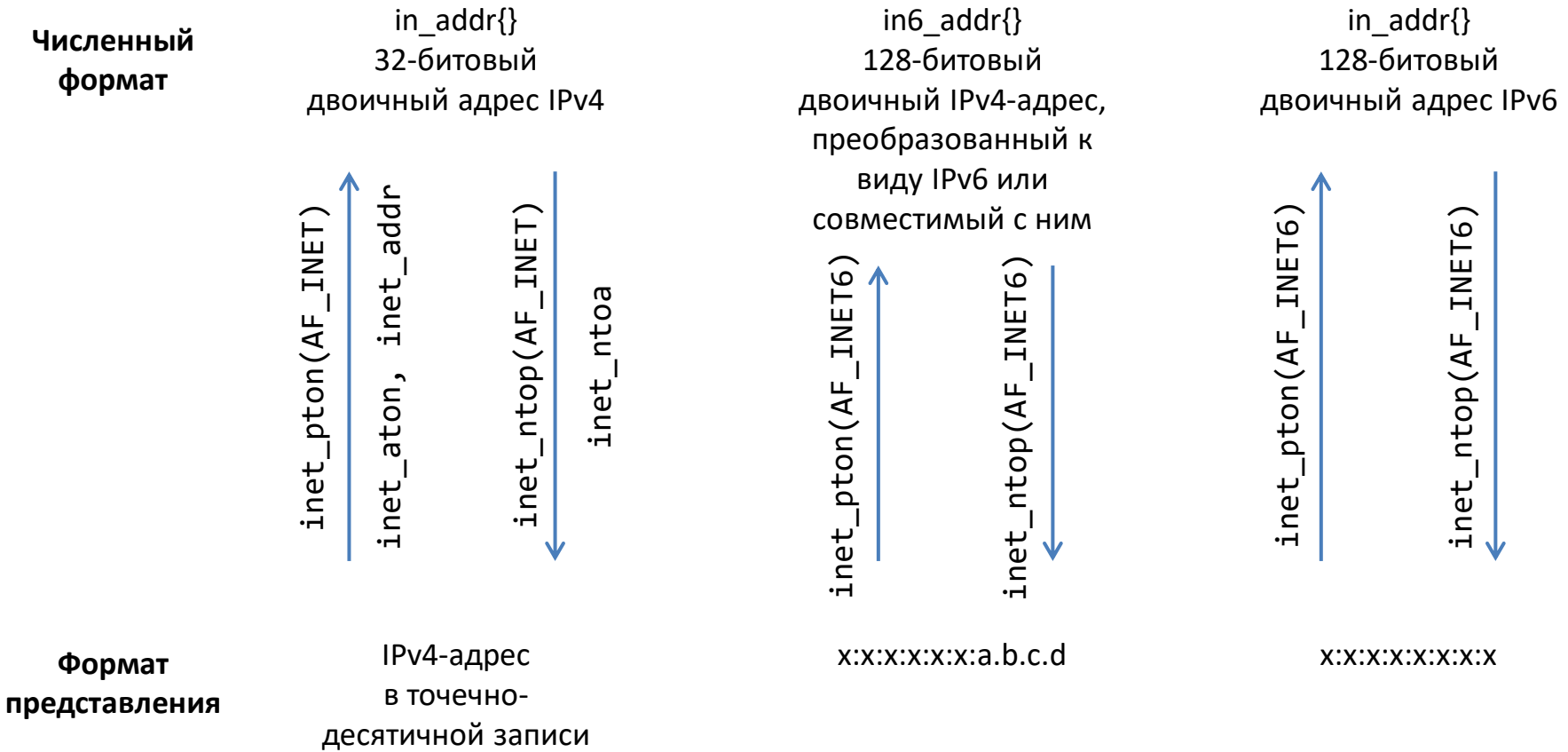
- `int inet_aton(const char *cp, struct in_addr *addrptr);`
- `in_addr_t inet_addr(const char *cp);`
- `char *inet_ntoa(struct in_addr in);`

inet_aton выполняет проверку допустимости адреса, содержащегося во входной строке, даже если addrptr – пустой указатель (null), но не сохраняет результата.

- `int inet_pton(int family, const char *strptr, void *addrptr);`
- `const char *inet_ntop(int family, const void *addrptr, char *strptr, size_t len);`

family: AF_INET, либо AF_INET6

Сокеты


Численный формат
in_addr{}
32-битовый
двоичный адрес IPv4
in6_addr{}
128-битовый
двоичный IPv4-адрес,
преобразованный к
виду IPv6 или
совместимый с ним
in_addr{}
128-битовый
двоичный адрес IPv6
inet_pton(AF_INET)
inet_aton, inet_addr
inet_ntop(AF_INET)
inet_ntoa
inet_pton(AF_INET6)
inet_ntop(AF_INET6)
inet_pton(AF_INET6)
inet_ntop(AF_INET6)
Формат представления
IPv4-адрес
в точечно-
десятичной записи
x:x:x:x:x:x:a.b.c.d
x:x:x:x:x:x:x:x

Работа с DNS

- Получение адреса по имени хоста:

```
#include <netdb.h>
struct hostent *gethostbyname(const char *name);
```

- Описание хоста:

```
struct hostent {
  char *h_name; // Имя хоста
  char **h_aliases; // Псевдонимы хоста (NULL в конце списка)
  int h_addrtype; // Тип адреса – AF_INET
  int h_length; // Длина адреса (байт)
  char **h_addr_list; // Массив адресов (сетевой порядок)
};
#define h_addr h_addr_list[0]
```

Работа с DNS

```
hostent hst;
hst = gethostbyname("ftp.microsoft.com");
if (hst)
  memcpy((char*)&(dest_sin.sin_addr), hst->h_addr, hst->h_length);
```

В случае ошибки её расширенный код записывается в глобальную переменную **h_errno** (а не errno). Для вывода диагностического сообщения следует использовать **herror()** вместо perror().

Работа с DNS

- Получение имени хоста по адресу:
```
#include <netdb.h>
struct hostent *gethostbyaddr(char *addr, int len, int type);
```
- Получение имени локального хоста:
```
#include <unistd.h>
int gethostname(char *hostname, size_t size);
```
- Получение имени сокета:
```
#include <sys/socket.h>
int getsockname(int s, struct sockaddr *name, socklen_t *namelen);
```
- Получение адреса сокета удаленного хоста:
```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

Прием запросов от клиентов

- Преобразование неприсоединенного сокета в пассивный сокет (сервер TCP):
```
#include <sys/socket.h>
int listen(int sockfd, int backlog); // backlog – макс. число соединений
```
- Функция listen() обычно вызывается после функций socket() и bind(), перед вызовом accept().

- Получение установленного соединения из очереди (новый дескриптор – присоединенный сокет):
```
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen);
```
- Если очередь пуста, то вызов будет блокирующим.

- Перед вызовом:
```
addrlen = sizeof(struct sockaddr_in);
```
По завершении функции addrlen содержит действительное число байтов, помещенных ядром в структуру адреса сокета.

Подключение клиента

- Установка соединения с сервером:

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr,
            socklen_t addrlen);
```

- Нет необходимости вызывать bind() до вызова connect(). При необходимости ядро само выберет и динамически назначаемый порт, и IP-адрес отправителя.
- Если сокет имеет тип SOCK_DGRAM, значит адрес serv_addr является адресом по умолчанию, куда посылаются датаграммы, и единственным адресом, откуда они принимаются.
- Если сокет имеет тип SOCK_STREAM или SOCK_SEQPACKET, то данный системный вызов попытается установить соединение с другим сокетом. Другой сокет задан параметром serv_addr, являющийся адресом длиной addrlen в пространстве коммуникации сокета.
- Обычно сокеты с протоколами, основанными на соединении, могут устанавливать соединение только один раз; сокеты с протоколами без соединения могут использовать connect многократно, чтобы изменить адрес назначения.

Обмен данными

- Функции обмена данными (TCP):

```
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buff, size_t nbytes, int flags);
ssize_t send(int sockfd, const void *buff, size_t nbytes, int flags);
```

- Флаги: 0 или комбинация констант MSG_DONTROUTE, MSG_OOB, MSG_PEEK, MSG_WAITALL.

flags	Описание
MSG_DONTROUTE	Не искать в таблице маршрутизации (получатель находится в нашей сети).
MSG_OOB	Отправка или получение срочных (внеполосных, Out Of Band) данных. Сообщение должно быть послано прежде, чем будут посланы какие-либо обычные данные. Для recv этот флаг указывает, что вместо обычных данных должны читаться внеполосные данные.
MSG_PEEK	Просмотр приходящих сообщений (при повторном вызове recv, recvfrom снова возвращаются уже просмотренные данные).
MSG_WAITALL	Ожидание всех данных. Операция чтения должна выполняться до тех пор, пока не будет прочитано запрашиваемое количество байтов.

Обмен данными

- Дейтаграммный сокет (SOCK_DGRAM) также может пользоваться функциями send и recv, если предварительно вызовет connect().
- Функции обмена данными (UDP):

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

- Требуется явное указание адреса узла, принимающего или передающего данные.
- Вызов **recvfrom** не требует предварительного задания адреса передающего узла. Функция принимает все пакеты, приходящие на заданный UDP-порт со всех IP адресов и портов.
- Отвечать отправителю следует на тот же самый порт, откуда пришло сообщение. Поскольку функция **recvfrom** запоминает IP-адрес и номер порта клиента, после получения от него сообщения нужно передать в **sendto** тот же самый указатель на структуру **sockaddr**, который был ранее передан функции **recvfrom**, получившей сообщение от клиента.

Отключение и закрытие сокетов

- Пометить сокет как закрытый и немедленно вернуть управление процессу

```
#include <unistd.h>
int close(int sockfd);
```

- Дескриптор сокета больше не может быть использован функциями read(), write().
- TCP попытается отправить данные, которые уже установлены в очередь, и после их отправки осуществит нормальную последовательность завершения соединения TCP.

- Закрытие дуплексного соединения:

```
int shutdown(int sockfd, int how);
```

- Параметр **how**:
  - SHUT_RD - запрещен прием данных
  - SHUT_WR - запрещена передача данных
  - SHUT_RDWR - запрещены как прием, так и передача данных.

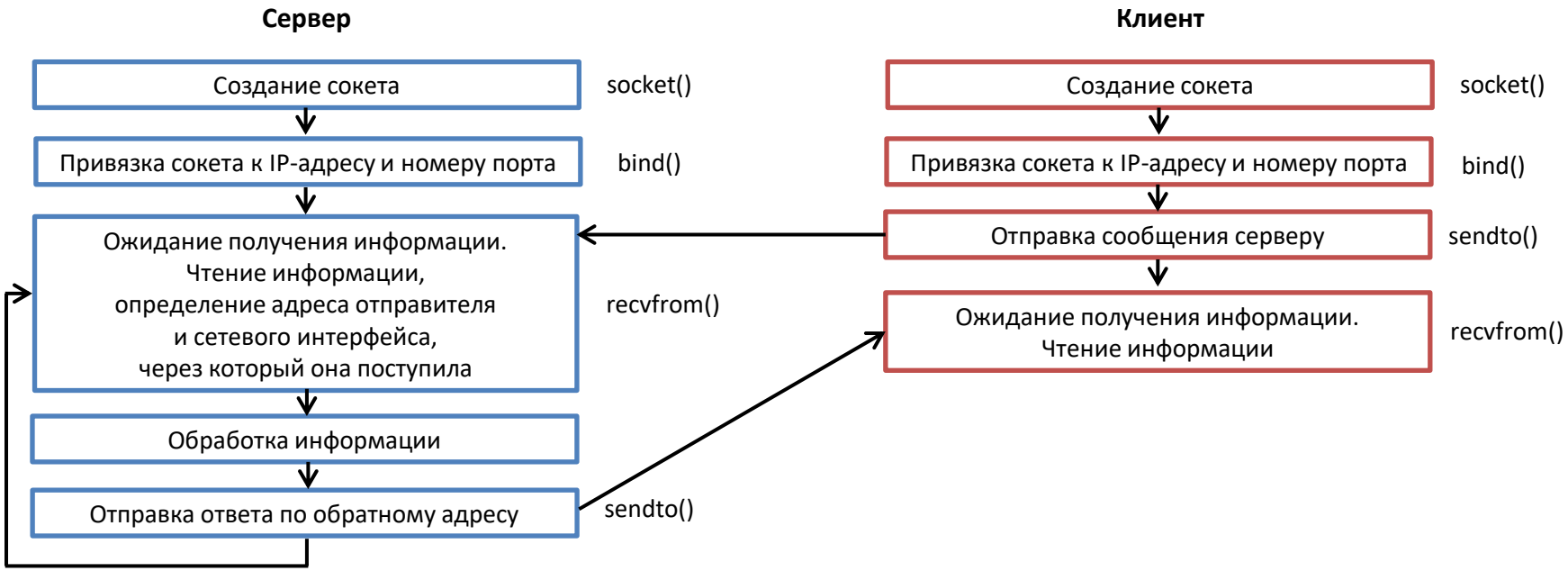
Опции сокетов

- Манипуляция флагами, установленными на сокете:

```
#include <sys/types.h>
#include <sys/socket.h>
int getsockopt(int s, int level, int optname, void *optval,
               socklen_t *optlen);
int setsockopt(int s, int level, int optname, const void *optval,
               socklen_t optlen);
```

- Функция работает на нескольких уровнях (канальный, сетевой, транспортный и т.д.).
- См. man 7 ip, раздел Socket options.

Взаимодействие клиента и сервера



UDP клиент

https://intuit.ru/studies/courses/2249/52/lecture/1567

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>

int main(int argc, char **argv) {
 int sockfd; /* Дескриптор сокета */
 int n, len; /* Переменные для различных длин и количества символов */
 char sendline[1000], recvline[1000]; /* Массивы для отсылаемой и принятой строки */
 struct sockaddr_in servaddr, cliaddr; /* Структуры для адресов сервера и клиента */

 /* Сначала проверяем наличие второго аргумента в
 командной строке. При его отсутствии ругаемся и прекращаем работу */
 if(argc != 2){
  printf("Usage: a.out <IP address>\n");
  exit(1);
 }
```

UDP клиент

```
/* Создаем UDP сокет */
   if((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
       perror(NULL); /* Печатаем сообщение об ошибке */
       exit(1);
   }

   /* Заполняем структуру для адреса клиента */
   bzero(&cliaddr, sizeof(cliaddr));
   cliaddr.sin_family = AF_INET;
   cliaddr.sin_port = htons(0);
   cliaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   /* Настраиваем адрес сокета */
   if(bind(sockfd, (struct sockaddr *) &cliaddr, sizeof(cliaddr)) < 0){
       perror(NULL);
       close(sockfd); /* По окончании работы закрываем дескриптор сокета */
       exit(1);
   }

   /* Заполняем структуру для адреса сервера */
   bzero(&servaddr, sizeof(servaddr));
   servaddr.sin_family = AF_INET;
   servaddr.sin_port = htons(7);
   if(inet_aton(argv[1], &servaddr.sin_addr) == 0) {
       printf("Invalid IP address\n");
       close(sockfd); /* По окончании работы закрываем дескриптор сокета */
       exit(1);
   }
```

UDP клиент

```
/* Вводим строку, которую отошлем серверу */
printf("String => ");
fgets(sendline, 1000, stdin);
/* Отсылаем датаграмму */
if(sendto(sockfd,
          sendline,
          strlen(sendline)+1,
          0,
          (struct sockaddr *) &servaddr,
          sizeof(servaddr)) < 0){
perror(NULL);
close(sockfd);
exit(1);
}
```

```
    /* Ожидаем ответа и читаем его. Максимальная
допустимая длина датаграммы – 1000 символов,
адрес отправителя нам не нужен */
    if((n = recvfrom(sockfd,
                     recvline,
                     1000,
                     0,
                     (struct sockaddr *) NULL,
                     NULL)) < 0){
        perror(NULL);
        close(sockfd);
        exit(1);
    }

    /* Печатаем пришедший ответ и закрываем сокет */
    printf("%s\n", recvline);
    close(sockfd);
    return 0;
}
```

UDP сервер

https://intuit.ru/studies/courses/2249/52/lecture/1567

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
int main()
{
    int sockfd; /* Дескриптор сокета */
    int clilen, n; /* Переменные для различных длин и количества символов */
    char line[1000]; /* Массив для принятой и отсылаемой строки */
    struct sockaddr_in servaddr, cliaddr; /* Структуры для адресов сервера и клиента */

    /* Заполняем структуру для адреса сервера */
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(51000);
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);

    /* Создаем UDP сокет */
    if((sockfd = socket(PF_INET, SOCK_DGRAM, 0)) < 0){
        perror(NULL); /* Печатаем сообщение об ошибке */
        exit(1);
    }
```

UDP сервер

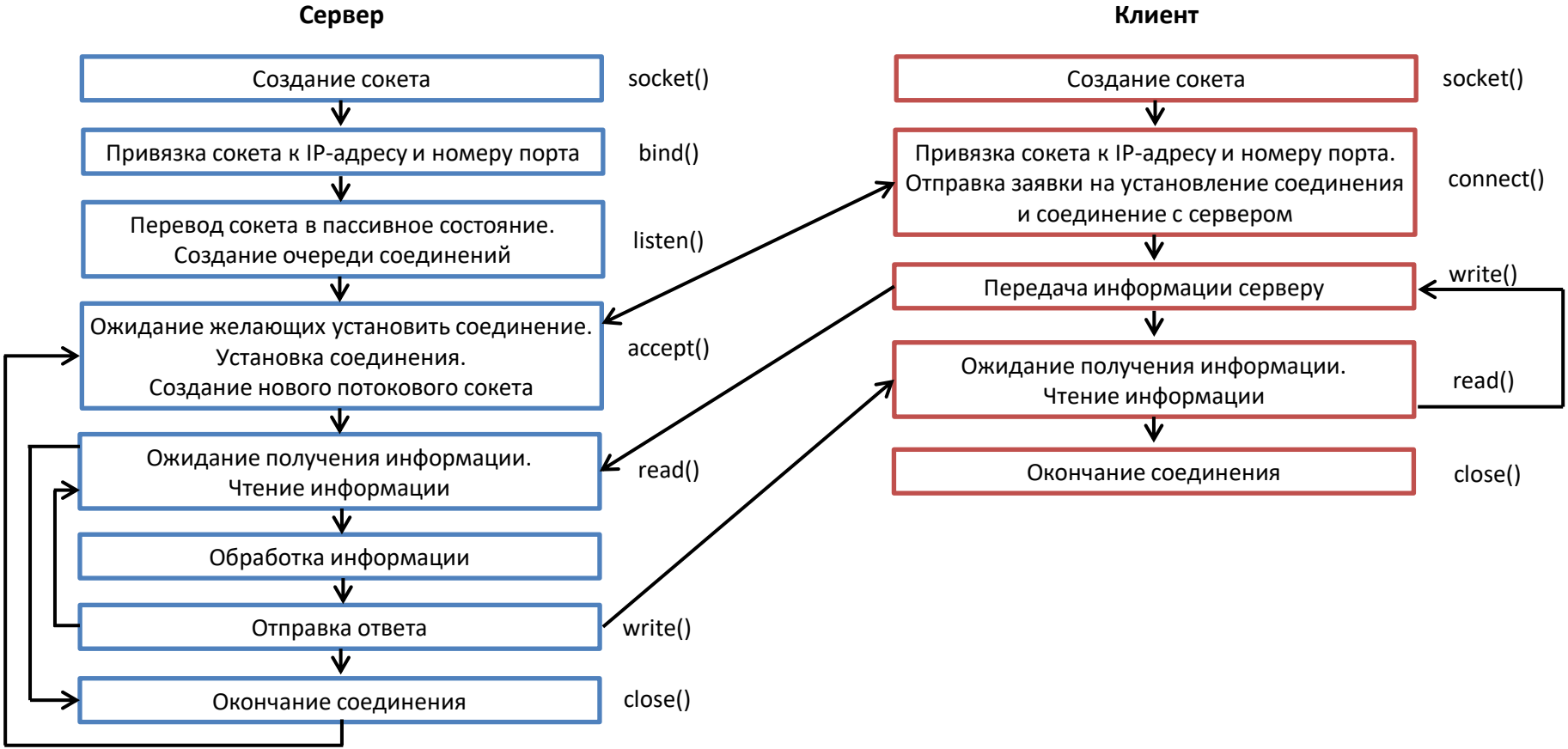
```
 /* Настраиваем адрес сокета */
    if(bind(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0){
        perror(NULL);
        close(sockfd);
        exit(1);
    }

    while(1) {
        /* Основной цикл обслуживания*/
        clilen = sizeof(cliaddr);
        /* Ожидаем прихода запроса от клиента и читаем его */
        if((n = recvfrom(sockfd, line, 999, 0,
        (struct sockaddr *) &cliaddr, &clilen)) < 0){
            perror(NULL);
            close(sockfd);
            exit(1);
        }

        /* Печатаем принятый текст на экране */
        printf("%s\n", line);
        /* Принятый текст отправляем обратно по адресу отправителя */
        if(sendto(sockfd, line, strlen(line), 0,
        (struct sockaddr *) &cliaddr, clilen) < 0){
            perror(NULL);
            close(sockfd);
            exit(1);
        } /* Уходим ожидать новую датаграмму*/
    }
    return 0;
}
```

Задания

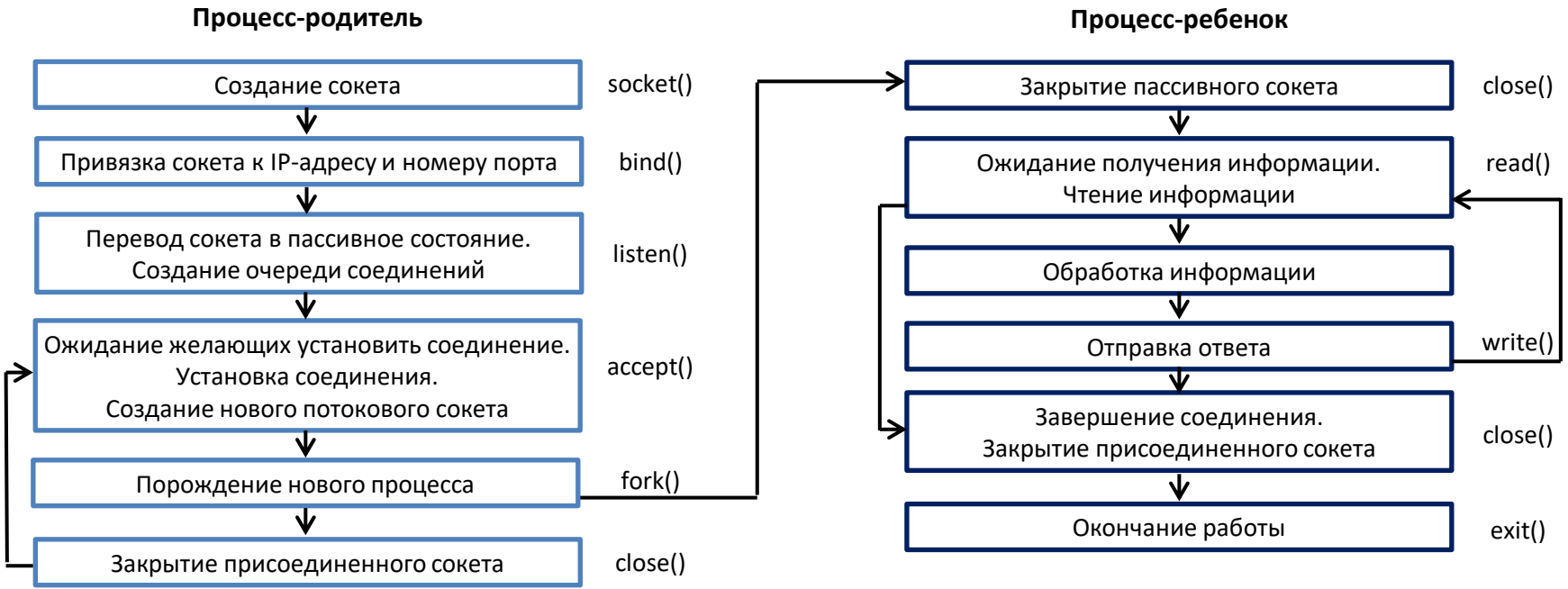
- **5.1** (1 балл). Отладить программы UDP-клиента и сервера. Добавить возможность работы по указанному порту (номер порта в параметрах запуска программы).

- **5.2** (2 балла). Реализовать чат для двух клиентов с использованием UDP-сокетов.

Взаимодействие клиента и сервера



Обработка запросов от клиентов



TCP клиент

http://gun.cs.nstu.ru/ssw/Linsockets/tcp_client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg) {
 perror(msg);
 exit(0);
}
```

```
int main(int argc, char *argv[])
{
 int my_sock, portno, n;
 struct sockaddr_in serv_addr;
 struct hostent *server;
 char buff[1024];

 printf("TCP DEMO CLIENT\n");

 if (argc < 3) {
  fprintf(stderr, "usage %s hostname port\n",
          argv[0]);
  exit(0);
 }

 // извлечение порта
 portno = atoi(argv[2]);
```

TCP клиент

```
// Шаг 1 - создание сокета
my_sock = socket(AF_INET, SOCK_STREAM, 0);

if (my_sock < 0) error("ERROR opening socket");

// извлечение хоста
server = gethostbyname(argv[1]);
if (server == NULL) {
 fprintf(stderr, "ERROR, no such host\n");
 exit(0);
}

// заполнение структуры serv_addr
bzero((char*) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
bcopy((char*)server->h_addr, (char *)&serv_addr.sin_addr.s_addr,
      server->h_length);

// установка порта
serv_addr.sin_port = htons(portno);
```

TCP клиент

```
// Шаг 2 - установка соединения
if (connect(my_sock, (struct sockaddr *) &serv_addr,sizeof(serv_addr)) < 0)
 error("ERROR connecting");

// Шаг 3 - чтение и передача сообщений
while ((n = recv(my_sock, &buff[0], sizeof(buff) - 1, 0)) > 0)
{
 // ставим завершающий ноль в конце строки
 buff[n] = 0;

 // выводим на экран
 printf("S=>C:%s", buff);

 // читаем пользовательский ввод с клавиатуры
 printf("S<=C:");
 fgets(&buff[0], sizeof(buff) - 1, stdin);
```

TCP клиент

```
  // проверка на "quit"
  if (!strcmp(&buff[0], "quit\n")) {
   // Корректный выход
   printf("Exit...");
   close(my_sock);
   return 0;
  }

  // передаем строку клиента серверу
  send(my_sock, &buff[0], strlen(&buff[0]), 0);
 }
 printf("Recv error \n");
 close(my_sock);
 return -1;
}
```

TCP сервер

http://gun.cs.nstu.ru/ssw/Linsockets/tcp_server.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

// функция обслуживания
// подключившихся пользователей
void dostuff(int);

// функция обработки ошибок
void error(const char *msg) {
    perror(msg);
    exit(1);
}

// количество активных пользователей
int nclients = 0;
```

```
// печать количества активных
// пользователей
void printusers() {
 if(nclients) {
  printf("%d user on-line\n",
          nclients);
 }
 else {
  printf("No User on line\n");
 }
}

// функция обработки данных
int myfunc(int a, int b) {
        return a + b;
}
```

TCP сервер

```
int main(int argc, char *argv[])
{
 char buff[1024]; // Буфер для различных нужд
 int sockfd, newsockfd; // дескрипторы сокетов
 int portno; // номер порта
 int pid; // id номер потока
 socklen_t clilen; // размер адреса клиента типа socklen_t
 struct sockaddr_in serv_addr, cli_addr; // структура сокета сервера и клиента

 printf("TCP SERVER DEMO\n");

 // ошибка в случае если мы не указали порт
 if (argc < 2) {
  fprintf(stderr, "ERROR, no port provided\n");
  exit(1);
 }
```

TCP сервер

```
// Шаг 1 - создание сокета
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) error("ERROR opening socket");

// Шаг 2 - связывание сокета с локальным адресом
bzero((char*) &serv_addr, sizeof(serv_addr));
portno = atoi(argv[1]);
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY; // сервер принимает подключения на все
IP-адреса
serv_addr.sin_port = htons(portno);

if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
 error("ERROR on binding");

// Шаг 3 - ожидание подключений, размер очереди - 5
listen(sockfd, 5);
clilen = sizeof(cli_addr);
```

TCP сервер

```
// Шаг 4 - извлекаем сообщение из очереди (цикл извлечения запросов на подключение)
while (1)
{
 newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
 if (newsockfd < 0) error("ERROR on accept");
 nclients++;

 // вывод сведений о клиенте
 struct hostent *hst;
 hst = gethostbyaddr((char *)&cli_addr.sin_addr, 4, AF_INET);
 printf("+%s [%s] new connect!\n",
  (hst) ? hst->h_name : "Unknown host",
  (char*)inet_ntoa(cli_addr.sin_addr));
 printusers();

 pid = fork();
 if (pid < 0) error("ERROR on fork");
 if (pid == 0) {
  close(sockfd);
  dostuff(newsockfd);
  exit(0);
 }
 else close(newsockfd);
}

close(sockfd);
return 0;
}
```

TCP сервер

```
void dostuff (int sock) {
 int bytes_recv; // размер принятого сообщения
 int a,b; // переменные для myfunc
 char buff[20 * 1024];
 #define str1 "Enter 1 parameter\r\n"
 #define str2 "Enter 2 parameter\r\n"

 // отправляем клиенту сообщение
 write(sock, str1, sizeof(str1));

 // обработка первого параметра
 bytes_recv = read(sock,&buff[0],sizeof(buff));
 if (bytes_recv < 0) error("ERROR reading from socket");
 a = atoi(buff); // преобразование первого параметра в int

 // отправляем клиенту сообщение
 write(sock,str2,sizeof(str2));

 bytes_recv = read(sock,&buff[0],sizeof(buff));
 if (bytes_recv < 0) error("ERROR reading from socket");
 b = atoi(buff); // преобразование второго параметра в int
```

TCP сервер

```
 a = myfunc(a,b); // вызов пользовательской функции
 snprintf(buff, strlen(buff), "%d", a); // преобразование результата в строку
 buff[strlen(buff)] = '\n'; // добавление к сообщению символа конца строки

 // отправляем клиенту результат
 write(sock,&buff[0], sizeof(buff));

 nclients--; // уменьшаем счетчик активных клиентов
 printf("-disconnect\n");
 printusers();
 return;
}
```

Задания

- **5.3** (2 балла). Отладить программы TCP-клиента и сервера. Добавить возможность вычисления разности, произведения, частного двух чисел. Необходимое математическое действие указывает клиент.

- **5.4** (4 балла). Написать программу для передачи файлов по сети с использованием TCP-сокетов.