Взаимодействие процессов через сокеты

Понятие сокета

Сокет- это объект ОС, через который можно передавать и принимать данные от процессов независимо от того, выполняются ли они на одном или разных компьютерах сети.

Сокеты, имеющие одинаковые схемы адресации и семейство протоколов группируются в домены, которые определяют тип соединения. В рамках одного домена сокеты могут иметь различный тип и протокол обмена.
AF_UNIX – для взаимодействия внутри одного компьютера.
AF_INET – для взаимодействия через сеть по протоколу TCP/IP.
AF_IPX – для взаимодействия через сеть по протоколу IPX.
AF_INET6 – для взаимодействия через сеть по протоколу IPv6.

Открытие сокета

Эта операция выполняет построение сокета и возвращает его дескриптор. Одновременно в системе могут быть открыты не более 65535 сокетов, по количеству портов на которых они могут быть открыты.

int socket(int domain, int type, int protocol);

domain = AF_INET

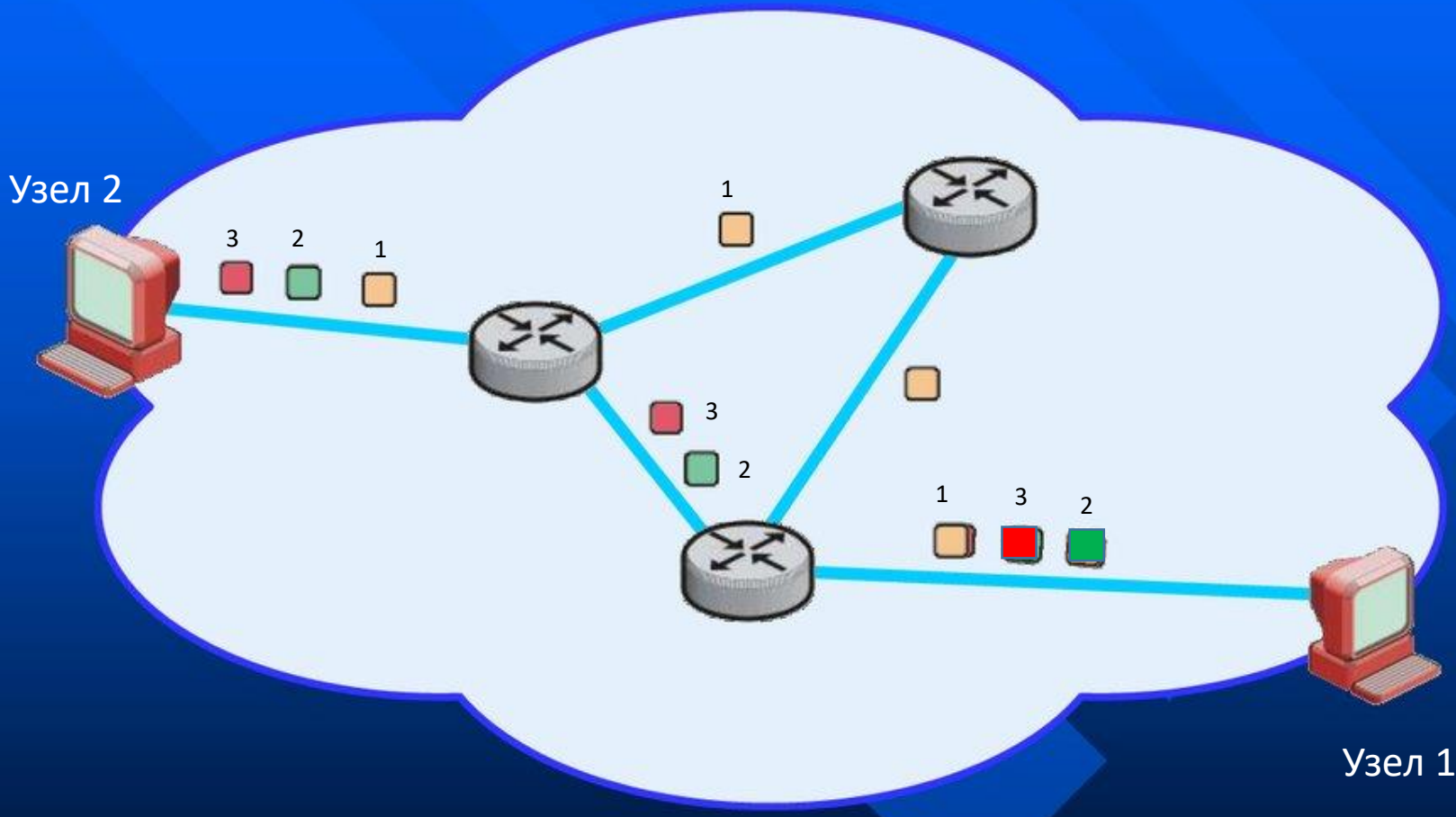
type = {SOCK_STREAM потоковый сокет,
SOCK_DGRAM дейтаграммный сокет
}

Потоковый	Дейтаграммный
Устанавливает соединение	Нет
Гарантирует доставку данных	Нет
Гарантирует порядок доставки данных	Нет
Гарантирует целостность данных	Тоже
Разбивает сообщение на пакеты	Нет
Контролирует поток данных	Нет
Медленнее	Быстрее

protocol – чаще всего 0, в этом случае система самостоятельно назначит протокол в зависимости от типа сокета (для SOCK_STREAM протокол IPPROTO_TCP, для SOCK_DGRAM протокол PPROTO_UDP)

Метод *дейтаграмм*

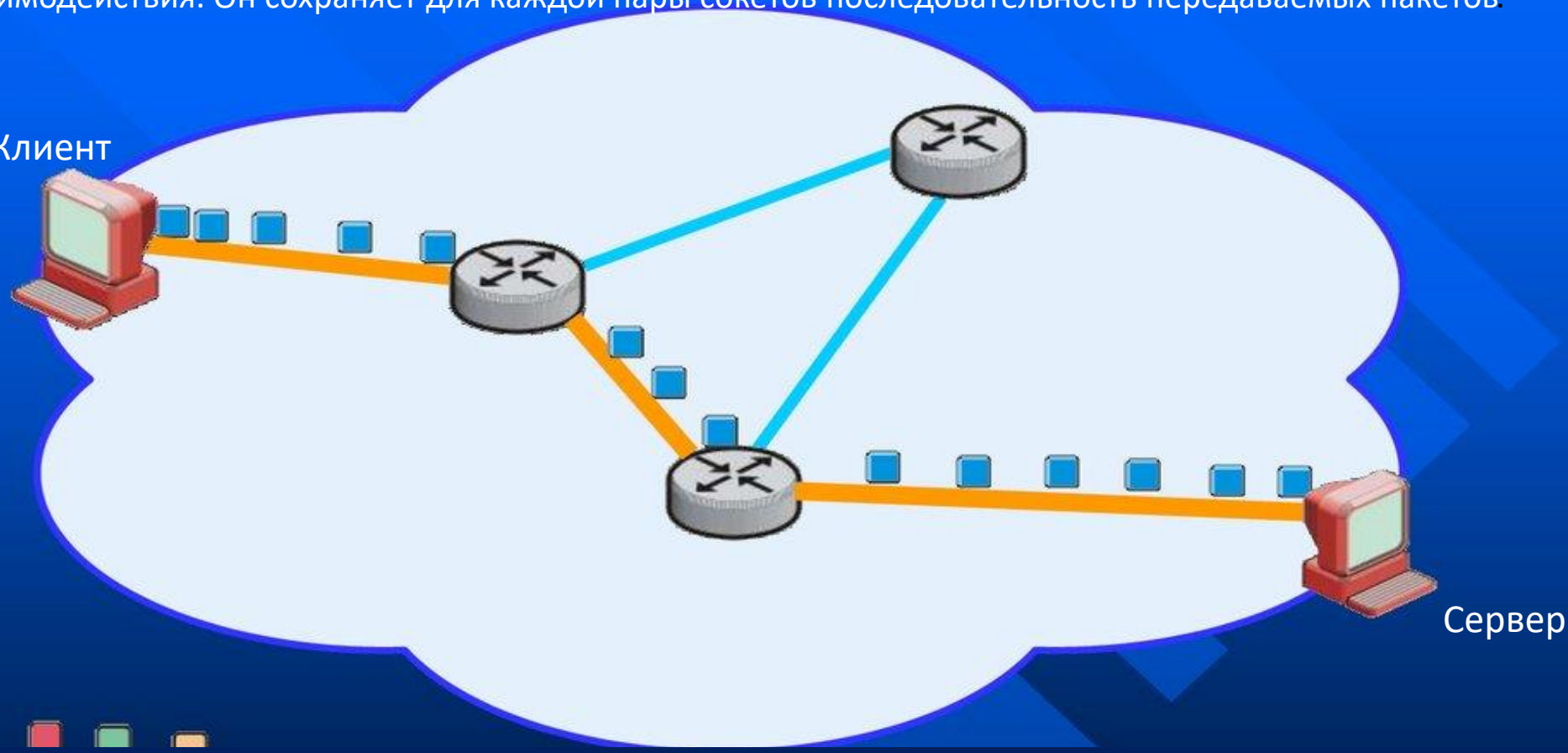
Маршрут сообщения в каждый момент времени зависит от состояния сети.
При движении сообщения по сети возможно нарушение исходной последовательности.



Для восстановления исходной последовательности используется размещение в заголовке сообщения его порядкового номера

Метод виртуального канала

Виртуальный канал – это соединение между двумя потоковыми сокетами, которое устанавливается на время их взаимодействия. Он сохраняет для каждой пары сокетов последовательность передаваемых пакетов.



Создание виртуального канала включает:

- Построение на сервере очереди запросов на соединение с клиентом
- Посылка клиентом запроса на создание канала
- Прием запроса от клиента и установка соединения
-

Именование сокета (привязка к сетевому адресу)

int bind(int sockfd, struct sockaddr *addr, int addrlen);

sockfd – дескриптор сокета

addr – указатель на структуру с адресом

addrlen = sizeof(addr) – длина структуры

Для домена AF_INET адрес задается следующей структурой

```
struct sockaddr_in {
    short int          sin_family;  // Тип домена – значение AF_INET
    unsigned short int sin_port;    // Номер порта в сетевом порядке байт, старший байт передавать первым
// если 0, то система самостоятельно выберет номер порта, первые 1024 порта зарезервированы
    struct in_addr     sin_addr;    // IP-адрес в сетевом порядке байт
    unsigned char      sin_zero[8]; // Дополнение до размера структуры sockaddr
};
```

все значения, возвращенные socket-функциями, уже находятся в сетевом формате

Полю sin_addr, в котором указывается IP-адрес сокета можно присвоить 32х битное значение IP адреса или следующие константы:

INADDR_ANY – сокет будет связан со всеми интерфейсами локального хоста (0.0.0.0);

INADDR_LOOPBACK сокет будет связан с интерфейсом обратной петли (127.0.0.1);

При указании IP-адреса и номера порта необходимо преобразовать число из порядка хоста в сетевой. Для этого используются функции **htons (port)** и **inet_addr (adr)**

addr.sin_port = htons (5000); (короткое целое из узлового 0x8813 в сетевой порядок следования байтов 0x1388)

addr.sin_addr.s_addr = inet_addr ("192.168.0.1"); из точечной нотации в 32-битное число с сетевым порядком следования байтов

Создание очереди для хранения ждущих обработки запросов

Данная функция применима только к сокетам типа SOCK_STREAM. В его задачу входит перевод *сокета* в *пассивное (слушающее)* состояние и создание очередей для порождаемых при установлении соединения *присоединенных сокетов*, находящихся в состоянии *не полностью установленного соединения* и *полностью установленного соединения*.

```
int listen(int sockfd, int backlog);
```

- **sockfd** — дескриптор сокета сервера.
- **backlog** — целое число, означающее длину очереди, максимальное значение 4096, рекомендуемое 5.

Входящие соединения, не превышающие максимальной длины очереди, сохраняются в ожидании сокета; последующим запросам на соединение будет отказано, и клиентская попытка соединения завершится аварийно.

Установка соединения

int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Если соединение или привязка прошла успешно, возвращается нуль. При ошибке возвращается -1.

sockfd - дескриптор сокета, который будет использоваться для обмена данными с сервером

addr - содержит указатель на структуру с адресом сервера или получателя

addrlen - длина этой структуры

Для сокета типа *Stream* вызов *connect()* соединяет сокет клиента с сокетом сервера, если сервер раньше вызовет *accept()*.

Для сокета типа *Datagram* вызов *connect()* запоминает адрес получателя, для отправки сообщений вызовом *send()*. Можно пропустить этот вызов и отправлять сообщения вызовом *sendto()*, явно указывая адрес получателя для каждого сообщения.

Прием запросов на создание канала

int accept(int sockfd, void *addr, int *addrlen);

Возвращает дескриптор сокета для обмена с клиентом, или -1 в случае ошибки.

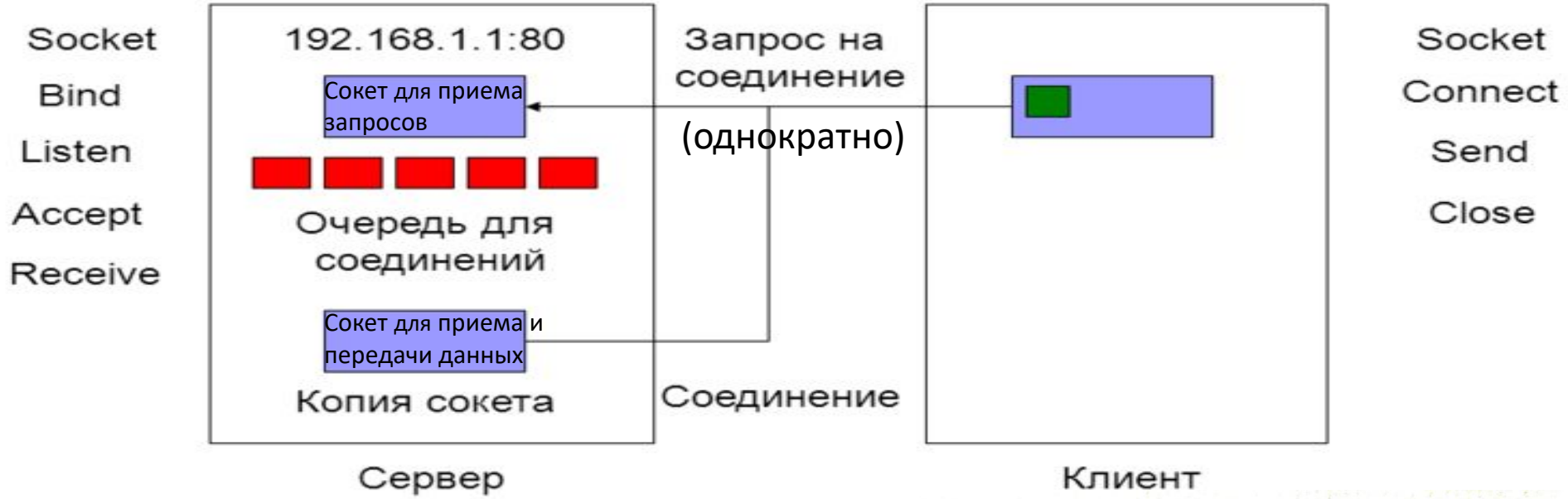
sockfd - дескриптор сокета, принимающий запросы от клиента на соединение

addr содержит указатель на структуру с адресом сервера

addrlen - длина этой структуры.

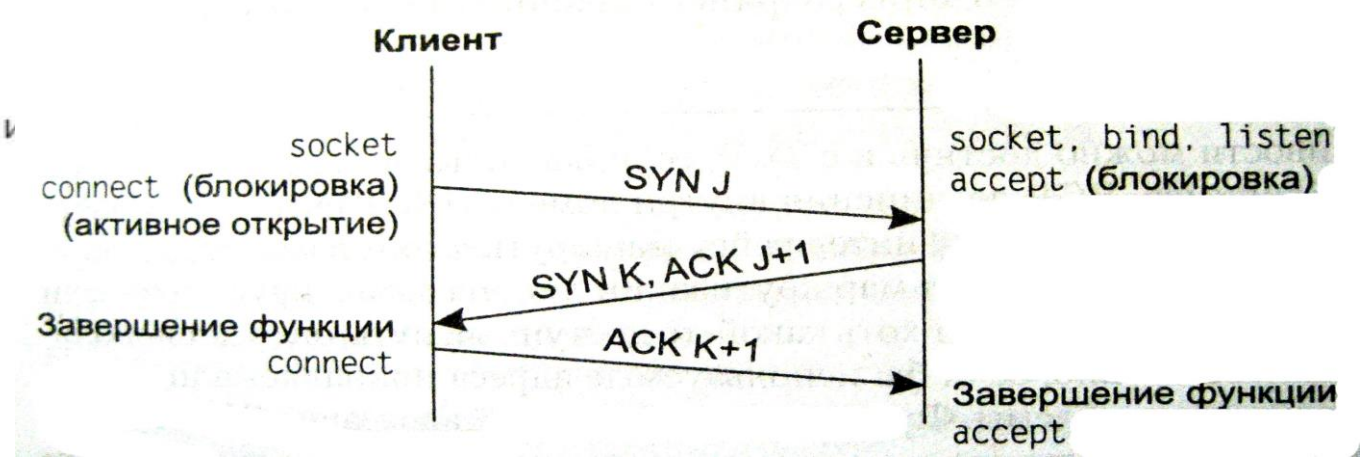
Системный вызов ***accept*** используется сервером, ориентированным на установление связи путем *виртуального соединения*, для приема *полностью установленного соединения* . Если *очередь полностью установленных соединений* не пуста, то он создает **новый сокет** и возвращает его *дескриптор* для общения с первым запросившим *соединение клиентским сокетом* в этой очереди, одновременно удаляя запрос из очереди.
Если *очередь* пуста, то вызов ожидает появления *полностью установленного соединения*. *Системный вызов* также позволяет серверу узнать полный *адрес* клиента, установившего соединение.

Последовательность вызовов методов для создания виртуального канала



Трехэтапное рукопожатие

1. Клиент посылает сегмент SYN (синхронизация), чтобы сообщить серверу, начальный порядковый номер данных, которые он будет посылать по соединению.
2. Сервер подтверждает получение клиентского сегмента SYN, и посылает свой собственный сегмент SYN, содержащий начальный порядковый номер для данных, которые сервер будет посылать по соединению.
3. Клиент подтверждает получение сегмента SYN сервера.



Передача данных через сокет

Для потокового и датаграммного сокетов (предварительно connect**)**

int send(int sockfd, const char *msg, int len, int flags);

Для датаграммного сокета

int sendto(int sockfd, const char *msg, int len, int flags, const struct sockaddr *toaddr, int tolen) ;

- дескриптор *сокета*, через который отсылаются данные;
- адрес области памяти, где лежат данные, которые должны быть отправлены, и их длина;
- флаги, определяющие поведение системного вызова (в нашем случае они всегда будут иметь значение 0);
- указатель на структуру, содержащую *адрес сокета* получателя, и ее фактическая длина (для датаграмм);
- возвращают количество отправленных байт или -1, если произошла ошибка (сообщение слишком длинно для передачи за раз, то сообщение не передаётся).

Прием данных из сокета

Для потокового сокета

int recv(int sockfd, char *buf, int len, int flags);

Для датаграммного сокета

int recvfrom(int sockfd, char *buf, int len, int flags, struct sockaddr *fromaddr, int *fromlen);

- дескриптор *сокета*, через который принимаются данные;
- адрес области памяти, куда читаются данные, и их длина;
- флаги, определяющие поведение системного вызова (0 или MSG_PEEK – читает данные, не удаляя из сокета);
- указатель на структуру, содержащую *адрес сокета* передающего данные, и ее фактическая длина (для датаграмм);
- возвращают количество принятых байт или -1, если произошла ошибка (если сообщение содержит больше байт, чем заказано для чтения, то будет прочитано сколько заказано).

Завершение работы с сокетом

Прерывание связи с сокетом

int shutdown (int sockfd, int cntl);

sockfd – дескриптор сокета

Аргумент cntl может принимать следующие значения:

0: больше нельзя получать данные из сокета;

1: больше нельзя посылать данные в сокет;

2: больше нельзя ни посылать, ни принимать данные через этот сокет.

Закрытие сокета

int close(int sockfd);

После закрытия сокета все операции запрещены, сообщения, находящиеся в сокете теряются, ресурсы освобождаются.

Управление параметрами сокета

Получить параметры сокета

int getsockopt(int sockfd, int level, int optname, void *optval, socklen_t *optlen);

Установить параметры сокета

int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);

Чтобы получить информацию о сокете параметр level=SQL_SOCKED.

Для чтения и установки размера буферов приема и передачи сообщений, параметр optname должен соответственно иметь значение SO_RCVBUF и SO_SNDBUF. Максимальные размеры буферов сокетов 212992 байт. Значения SO_RCVTIMEO и SO_SNDTIMEO задают значение тайм-аута, определяющее максимальное время ожидания функции приема или передачи данных до ее завершения. Время указывается в секундах и микросекундах структуры struct timeval tv.

setsockopt(sockfd, SOL_SOCKET, SO_RCVTIMEO, (const char*)&tv, sizeof tv);

Параметры сетевого интерфейса

Имя компьютера определяется во время установки системы. Его имя и IP-адрес можно получить с помощью функций

int uname(struct utsname *_buf_);

```
struct utsname {
   char sysname[];    /* название операционной системы   */
   char nodename[];   /* имя машины */
   char release[];    /* идентификатор выпуска ОС (например, «2.6.28») */
   char version[];    /* версия ОС */
   char machine[];    /* идентификатор аппаратного обеспечения */
};
```

struct hostent *gethostbyname(const char *name);

```
struct hostent
{
        char FAR * h_name;                      // имя хоста
        char FAR * FAR * h_aliases;             // дополнительные названия
        short h_addrtype;                       // тип домена
        short h_length;                         // длинна каждого адреса в байтах
        char FAR * FAR * h_addr_list;    // список ip-адресов
};
```

Адрес протокола, связанный с локальным или удаленным сокетом можно узнать с помощью функции

int getsockname(int *sockfd*, struct sockaddr **localaddr*, socklen_t **addrlen*);

Параметры имеют тот же смысл, как и в методе bind. Для удобства чтения IP-адреса его числовое значение можно преобразовать в строку символов с помощью функции

char *inet_ntoa(struct in_addr in);

Определение состояния сокетов

int select(int *n*, fd_set **readfds*, fd_set **writefds*, fd_set **exceptfds*, struct timeval **timeout*);

n – кол-во опрашиваемых дескрипторов сокетов(на единицу больше самого большого номера описателей из всех наборов),если все FD_SETSIZE

readfds, writefds, exceptfds наборы дескрипторов, которые следует проверять, соответственно, на готовность к чтению, записи и на наличие исключительных ситуаций.

timeout- время ожидания (верхняя граница времени, которое пройдет перед возвратом из **select**), если 0, то процесс будет приостановлен до тех пор, пока один из сокетов не изменит свое состояние

void FD_CLR(int *fd*, fd_set **set*); удаление дескриптора из набора
int FD_ISSET(int *fd*, fd_set **set*); проверка наличия дескриптора в наборе
void FD_SET(int *fd*, fd_set **set*); добавление дескриптора в набор
void FD_ZERO(fd_set **set*); очистка набора

Значением функции является число сокетов, от которых поступила информация, а в наборах *readfds, writefds, exceptfds* будут содержаться номера дескрипторов этих сокетов, если 0, то истекло время ожидания

Пример использования select

```
int s1, s2, n, rv;
fd_set readfds;
struct timeval tv;
char buf1[256], buf2[256];
// предположим, что мы связались с обоими серверами в данной точке
//s1 = socket(...);
//s2 = socket(...);
//connect(s1, ...)...
//connect(s2, ...)...
// очищаем набор дескрипторов
FD_ZERO(&readfds);
// добавляем наши дескрипторы в набор
FD_SET(s1, &readfds);
FD_SET(s2, &readfds);
n = s2 + 1;
// задаём время тайм аута 10.5 сек
tv.tv_sec = 10;
tv.tv_usec = 500000;
rv = select(n, &readfds, NULL, NULL, &tv);
if (rv == -1) {
printf("select"); // ошибка при вызове select()
} else if (rv == 0) {
printf("Timeout occurred! No data after 5.5 seconds.\n");
} else {
// один или оба дескриптора имеют данные для использования функции recv()
if (FD_ISSET(s1, &readfds)) {
recv(s1, buf1, sizeof buf1, 0);
}
if (FD_ISSET(s2, &readfds)) {
recv(s2, buf2, sizeof buf2, 0);
}
}
```

Для повторного использования select надо снова задать tv.tv_sec = 10; tv.tv_usec = 500000;

Опрос сокетов с помощью poll()

```
#include <sys/poll.h>
int poll(struct pollfd *ufds, unsigned int nfds, int timeout);
```

ufds – массив структур
nfds – количество структур
timeout – время ожидания в миллисекундах

```
struct pollfd {
    int fd;         /* дескриптор сокета*/
    short events;    /* запрошенные события */
    hort revents;   /* возвращенные события */
    };
```

При успешном завершении вызова возвращается положительное значение, равное количеству структур с ненулевыми полями revents (другими словами, сокеты с обнаруженными событиями или ошибками). Нулевое значение указывает на то, что время ожидания истекло, и ни один из сокетов не был выбран. При ошибке возвращается -1, а переменной errno присваивается номер ошибки.
Поле events это входной параметр, указывающий на битовую маску событий:

```
#define POLLIN     0x0001    /* Есть данные для чтения */
#define POLLOUT    0x0004    /* Можно записывать данные*/
#define POLLERR    0x0008    /* Произошла ошибка  для revents */
```

- Нет никакого лимита количества наблюдаемых дескрипторов, можно мониторить более 1024 штук
- Не модифицируется структура pollfd, что даёт возможность её переиспользования между вызовами poll() — нужно лишь обнулить поле revents.
- Как и при использовании select, невозможно определить какие именно дескрипторы сгенерировали события без полного прохода по всем наблюдаемым структурам и проверки в них поля revents.

Пример использования pool

```
// два события
struct pollfd fds[2];

// от sock1 мы будем ожидать входящих данных
fds[0].fd = sock1;
fds[0].events = POLLIN;

//  от sock2 - исходящих
fds[1].fd = sock2;
fds[1].events = POLLOUT;

// ждём до 10 секунд
int ret = poll( &fds, 2, 10000 );
// проверяем успешность вызова
if ( ret == -1 )
   // ошибка
else if ( ret == 0 )
   // таймаут, событий не произошло
else
{
   // обнаружили событие, обнулим revents чтобы можно было переиспользовать структуру
   if ( pfd[0].revents & POLLIN )
      pfd[0].revents = 0;
      // обработка входных данных от sock1

   if ( pfd[1].revents & POLLOUT )
      pfd[1].revents = 0;
      // обработка исходящих данных от sock2
}
```

Схема взаимодействия процессов через датаграммные сокеты

Схема работы TCP-сервера с последовательной обработкой запросов

Блокирующие сокеты

сервер может обрабатывать только одно соединение за раз, выполнение программы остановится и не возобновится до тех пор, пока не поступят данные из сети

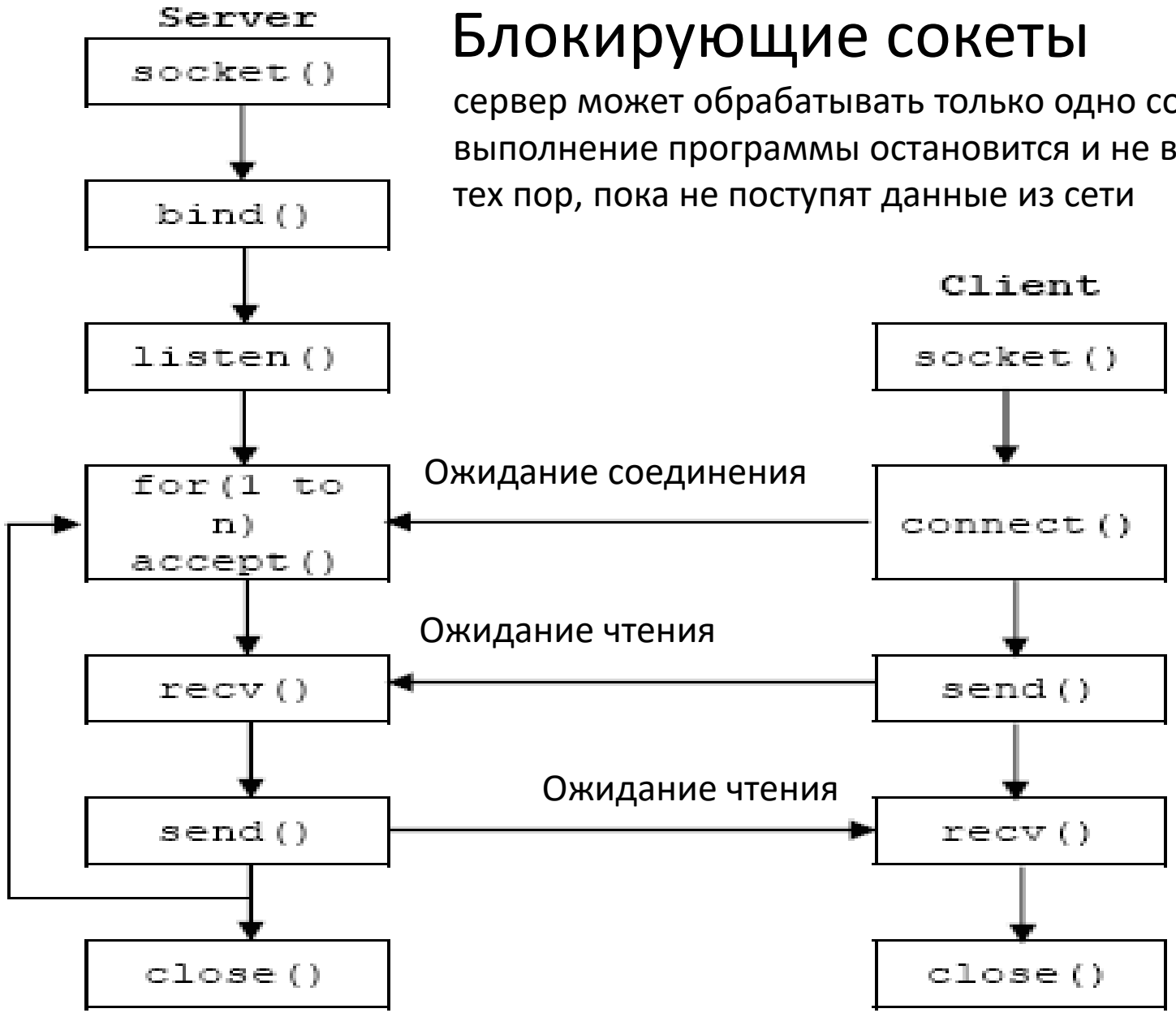


Схема работы TCP-сервера с параллельной обработкой запросов

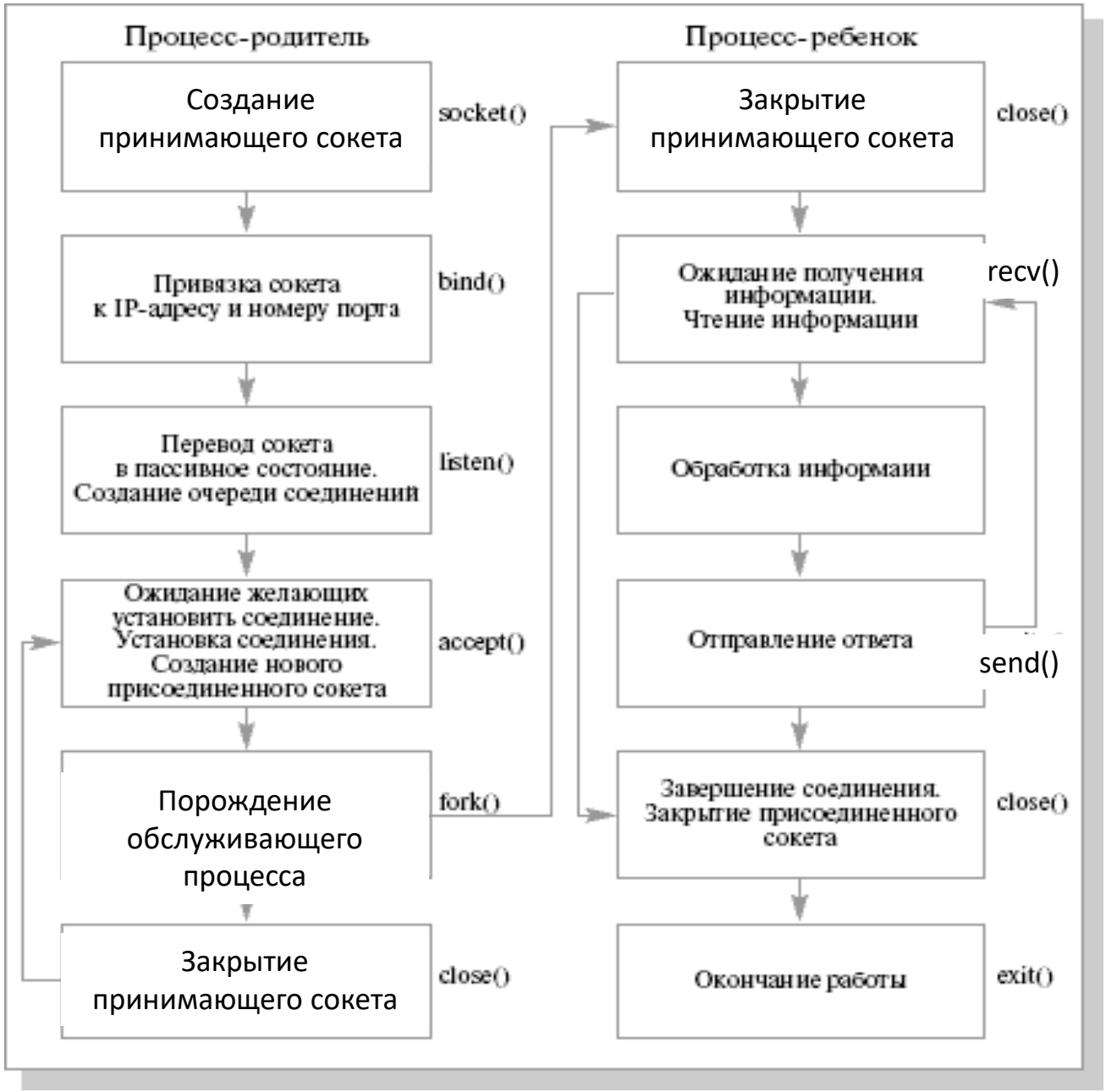


Схема работы TCP-сервера с неблокирующими сокетами

Задание 1

Написать две программы, которые обмениваются сообщениями через дейтаграммные сокеты и выводят принятые сообщения на экран. Первая программа посылает второй сообщение (данные, полученные от функции **uname**), а затем ждет от нее ответа. Вторая программа принимает сообщение, а затем посылает ответ первой программе (свой IP-адрес). Если обе программы не получают сообщение в течение определенного времени, которое задается при запуске обеих программ, то они заканчивают работу с уведомлением о времени ожидания и непринятии сообщения.

Задание 2

Написать две программы (сервер и клиент) , которые обмениваются сообщениями через потоковые сокеты. Сервер может принимать последовательно сообщения от нескольких клиентов и, если в течение определенного времени после обращения последнего клиента новые запросы на соединения и прием сообщения от клиентов не поступают, то сервер завершает свою работу с выводом на экран времени ожидания, которое задается при запуске программы. Клиенты запрашивают у сервера текущие дату и время и сообщает ему свое имя, которое задается при запуске клиента. Сервер выводит на экран имя клиента и передает ему дату и время. Полученный ответ клиент выводит на экран и завершает свою работу.

Задание 3

Написать две программы (сервер и клиент) , которые обмениваются сообщениями через потоковые сокеты. Клиенты проверяют возможность соединения с сервером и в случае отсутствия соединения или истечения времени ожидания отправки сообщения завершают работу. После соединения с сервером они генерируют случайную последовательность чисел и выводят ее на экран, а затем отсылают серверу. Сервер в течение определенного времени ждет запросы от клиентов и в случае их отсутствия завершает работу. При поступлении запроса от клиента сервер порождает обслуживающий процесс, который принимает последовательность чисел, упорядочивает ее и выводит на экран, а затем отсылает обратно клиенту и завершают работу. Клиент полученную последовательность выводит на экран и заканчивает свою работу.

Сервер на основе SOCK_DGRAM

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
int main()
{
   int sock;
   struct sockaddr_in addr;
   char buf[1024];
   int bytes_read;
   sock = socket(AF_INET, SOCK_DGRAM, 0);
   if(sock < 0)
   {  perror("socket");
      exit(1);
   }
      addr.sin_family = AF_INET;
   addr.sin_port = htons(4444);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   if(bind(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
   {   perror("bind");
      exit(2);
   }
```

```
printf("start reciever\n");

   while(1)
   {
      bytes_read = recvfrom(sock, buf, 1024, 0, NULL, NULL);
      buf[bytes_read] = '\0';
      printf(buf);
   }

   return 0;
}
```

Клиент на основе SOCK_DGRAM

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdlib.h>
char msg1[] = "Hello !\n";
char msg2[] = "Bye bye!\n";
int main()
{
   int sock;
   struct sockaddr_in addr;
   sock = socket(AF_INET, SOCK_DGRAM,
0);
   if(sock < 0)
   {  perror("socket");
      exit(1);
   }
   addr.sin_family = AF_INET;
   addr.sin_port = htons(4444);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   sendto(sock, msg1, sizeof(msg1), 0, (struct sockaddr *)&addr,
          sizeof(addr));
   connect(sock, (struct sockaddr *)&addr, sizeof(addr));
   printf("connect sender\n");
   send(sock, msg2, sizeof(msg2), 0);

   close(sock);

   return 0;
}
```

Сервер TCP

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdlib.h>
int main()
{   int sock, listener;
    struct sockaddr_in addr;
    char buf[1024];
    int bytes_read;
    listener = socket(AF_INET, SOCK_STREAM, 0);
    if(listener < 0)
    {  perror("socket");
        exit(1);
    }
        addr.sin_family = AF_INET;
    addr.sin_port = htons(3434);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(listener, (struct sockaddr *)&addr, sizeof(addr)) < 0)
    {   perror("bind");
        exit(2);
    }
listen(listener, 1);
 while(1)
  {
      sock = accept(listener, NULL, NULL);
      if(sock < 0)
      {  perror("accept");
         exit(3);
      }
      while(1)
      {
         bytes_read = recv(sock, buf, 1024, 0);
         if(bytes_read <= 0) break;
         printf(buf);
         send(sock, buf, bytes_read, 0);
       }
   close(sock);
  }
      return 0;
}
```

Клиент сервера TCP

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
char message[] = "Hello!\n";
char buf[sizeof(message)];
int main()
{
   int sock;
   struct sockaddr_in addr;
   sock = socket(AF_INET, SOCK_STREAM, 0);
   if(sock < 0)
   {  perror("socket");
      exit(1);
   }
   addr.sin_family = AF_INET;
   addr.sin_port = htons(3434);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   if(connect(sock, (struct sockaddr *)&addr,
   sizeof(addr)) < 0)
      {
         perror("connect");
         exit(2);
      }
      send(sock, message, sizeof(message), 0);
      recv(sock, buf, sizeof(message), 0);
      printf(buf);
      close(sock);

      return 0;
   }
```