Взаимодействие процессов: сокеты

Создание сокета

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int socket ( int domain, int type, int protocol ) ;
```

Параметры

domain — коммуникационный домен:

- **AF_UNIX**
- **AF_INET**

type — тип сокета:

- **SOCK_STREAM** — виртуальный канал
- **SOCK_DGRAM** — датаграммы

protocol — протокол:

- **0** — автоматический выбор протокола
- **IPPROTO_TCP** — протокол TCP (AF_INET)
- **IPPROTO_UDP** — протокол UDP (AF_INET)

Связывание

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int bind ( int sockfd, struct sockaddr * myaddr, int
addrlen ) ;
```

Параметры

sockfd — дескриптор сокета

myaddr — указатель на структуру, содержащую адрес сокета

Структура адреса для домена **AF_UNIX**

```
#include <sys/un.h>
struct sockaddr_un {
   short sun_family ;   /* == AF_UNIX */
   char sun_path [ 108 ] ;
} ;
```

Связывание

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int bind ( int sockfd, struct sockaddr * myaddr, int
addrlen ) ;
```

Параметры

sockfd — дескриптор сокета

myaddr — указатель на структуру, содержащую адрес сокета

Структура адреса для домена **AF_INET**

```
#include <netinet/in.h>
struct sockaddr_in {
  short sin_family ;    /* == AF_INET */
  u_short sin_port ;    /* port number */
  struct in_addr sin_addr ;    /* host IP address */
  char sin_zero [ 8 ] ; /* not used */
} ;
```

Связывание

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int bind ( int sockfd, struct sockaddr * myaddr, int
addrlen ) ;
```

Параметры

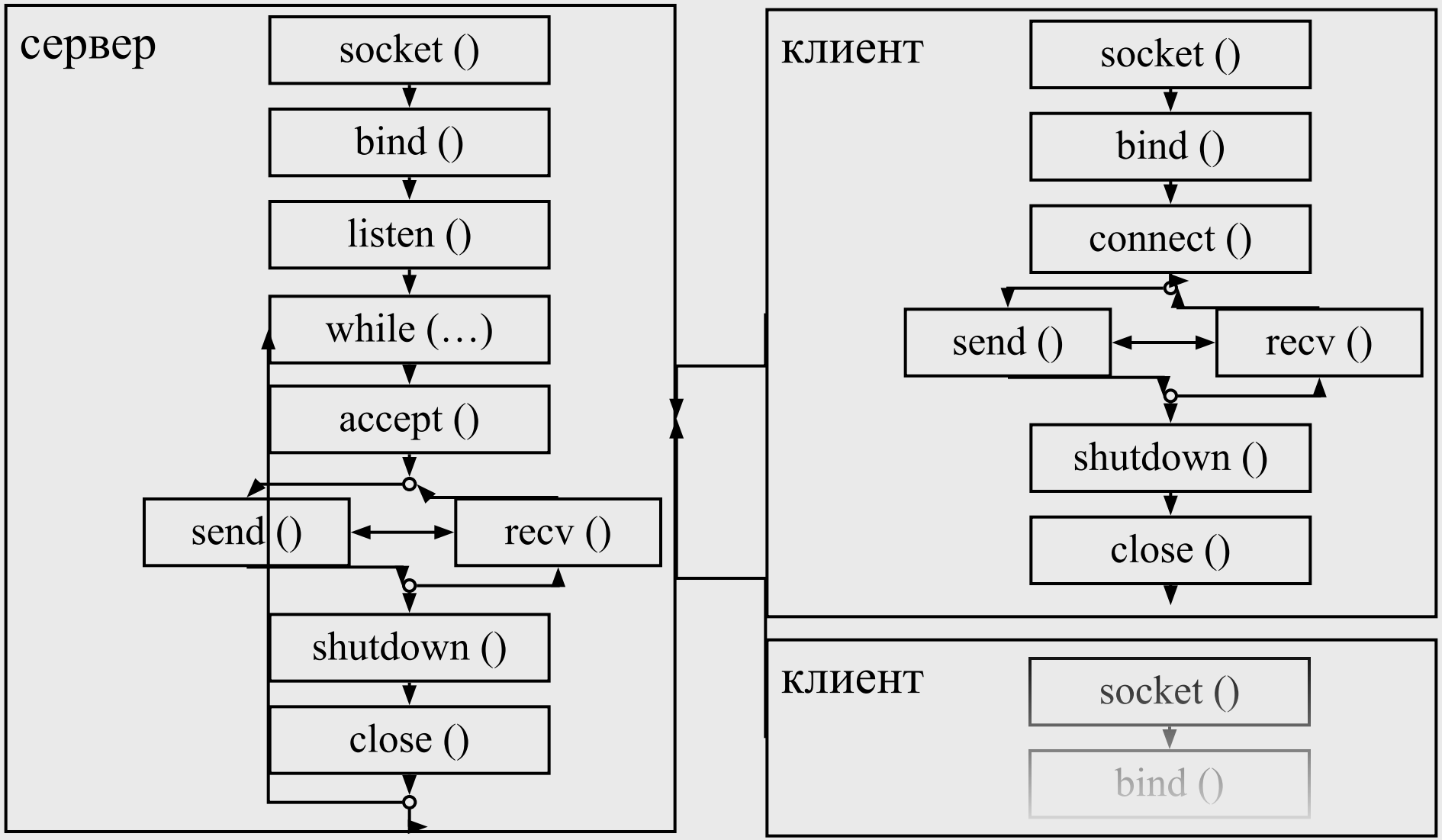
addrlen — размер структуры **sockaddr** (**«доменный» адрес сокета**).

Возвращаемое значение

В случае успешного связывания **bind** возвращает 0, в случае ошибки — –1.

Предварительное установление соединения

(тип сокета — виртуальный канал или датаграмма)



Прослушивание сокета

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

Параметры

sockfd — дескриптор сокета

backlog — максимальный размер очереди запросов на соединение

Возвращаемое значение

В случае успешного обращения функция возвращает 0, в случае ошибки — –1. Код ошибки заносится в **errno**.

Запрос на соединение

Необходимые заголовочные файлы и прототип

#include <sys/types.h>

#include <sys/socket.h>

int connect (int sockfd, struct sockaddr * serv_addr, int addrlen) ;

Параметры

sockfd — дескриптор сокета

serv_addr — указатель на структуру, содержащую адрес сокета, с которым производится соединение

addrlen — реальная длина структуры

Возвращаемое значение

В случае успешного связывания функция возвращает 0, в случае ошибки — –1. Код ошибки заносится в **errno**.

Подтверждение соединения

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int accept ( int sockfd, struct sockaddr * addr, int *
addrlen ) ;
```

Параметры

sockfd — дескриптор сокета

addr — указатель на структуру, в которой возвращается адрес клиентского сокета, с которым установлено соединение (если адрес клиента не интересует, передается NULL).

addrlen — возвращается реальная длина этой структуры. максимальный размер очереди запросов на соединение.

Возвращаемое значение

- дескриптор нового сокета, соединенного с сокетом клиентского процесса.

Прием и передача данных

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int send ( int  sockfd, const
      void * msg, int len,
      unsigned int flags ) ;
```

```
int recv ( int sockfd, void *
      buf, int len, unsigned
      int flags ) ;
```

Параметры

sockfd — дескриптор сокета, через который передаются данные

msg — сообщение

len — длина сообщения

buf — указатель на буфер для приема данных

len — первоначальная длина буфера

Прием и передача данных

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int send ( int  sockfd, const
      void * msg, int len,
      unsigned int flags ) ;
```

```
int recv ( int sockfd, void *
      buf, int len, unsigned
      int flags ) ;
```

Параметры

flags — может содержать комбинацию специальных опций.

MSG_OOB — флаг сообщает ОС, что процесс хочет осуществить прием/передачу экстренных сообщений.

MSG_PEEK — При вызове **recv ()** процесс может прочитать порцию данных, не удаляя ее из сокета. Последующий вызов recv вновь вернет те же самые данные.

Прием и передача данных

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int send ( int  sockfd, const
      void * msg, int len,
      unsigned int flags ) ;
```

```
int recv ( int sockfd, void *
      buf, int len, unsigned
      int flags ) ;
```

Возвращаемое значение

Функция возвращает количество переданных байт в случае успеха и –1 в случае неудачи. Код ошибки при этом устанавливается в **errno**.

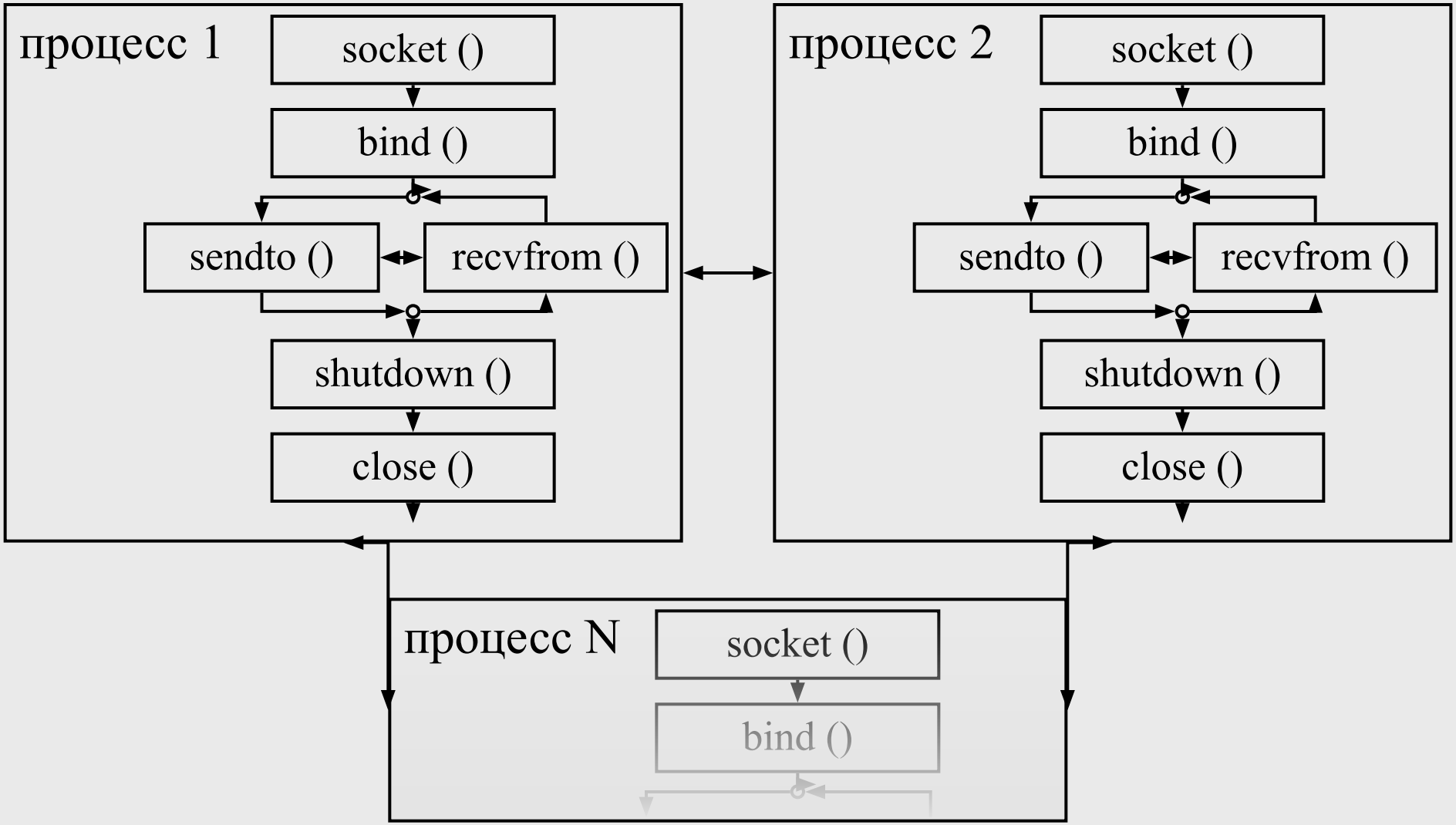
В случае успеха функция возвращает количество считанных байт, в случае неудачи –1.

Прием и передача данных

- **Read()**
- **Write()**

Сокеты без предварительного соединения

(тип сокета — датаграмма)



Прием и передача данных

Необходимые заголовочные файлы и прототип

```
#include <sys/types.h>
#include <sys/socket.h>
int sendto ( int sockfd,
    const void * msg,
    int len,
    unsigned  int flags,
    const struct sockaddr * to,
    int tolen ) ;

int recvfrom ( int sockfd,
    void * buf,
    int len,
    unsigned int flags,
    struct sockaddr * from,
    int * fromlen ) ;
```

Завершение работы с сокетом

Необходимые заголовочные файлы и прототип

#include <sys/types.h>

#include <sys/socket.h>

int shutdown (int sockfd, int mode) ;

(закрытие соединения)

int close (int fd) ;

(закрытие сокета)

Параметры

sockfd — дескриптор сокета

mode — режим закрытия соединения

= 0, сокет закрывается для чтения

= 1, сокет закрывается для записи

= 2, сокет закрывается и для чтения, и для записи

Схема работы с сокетами с установлением соединения

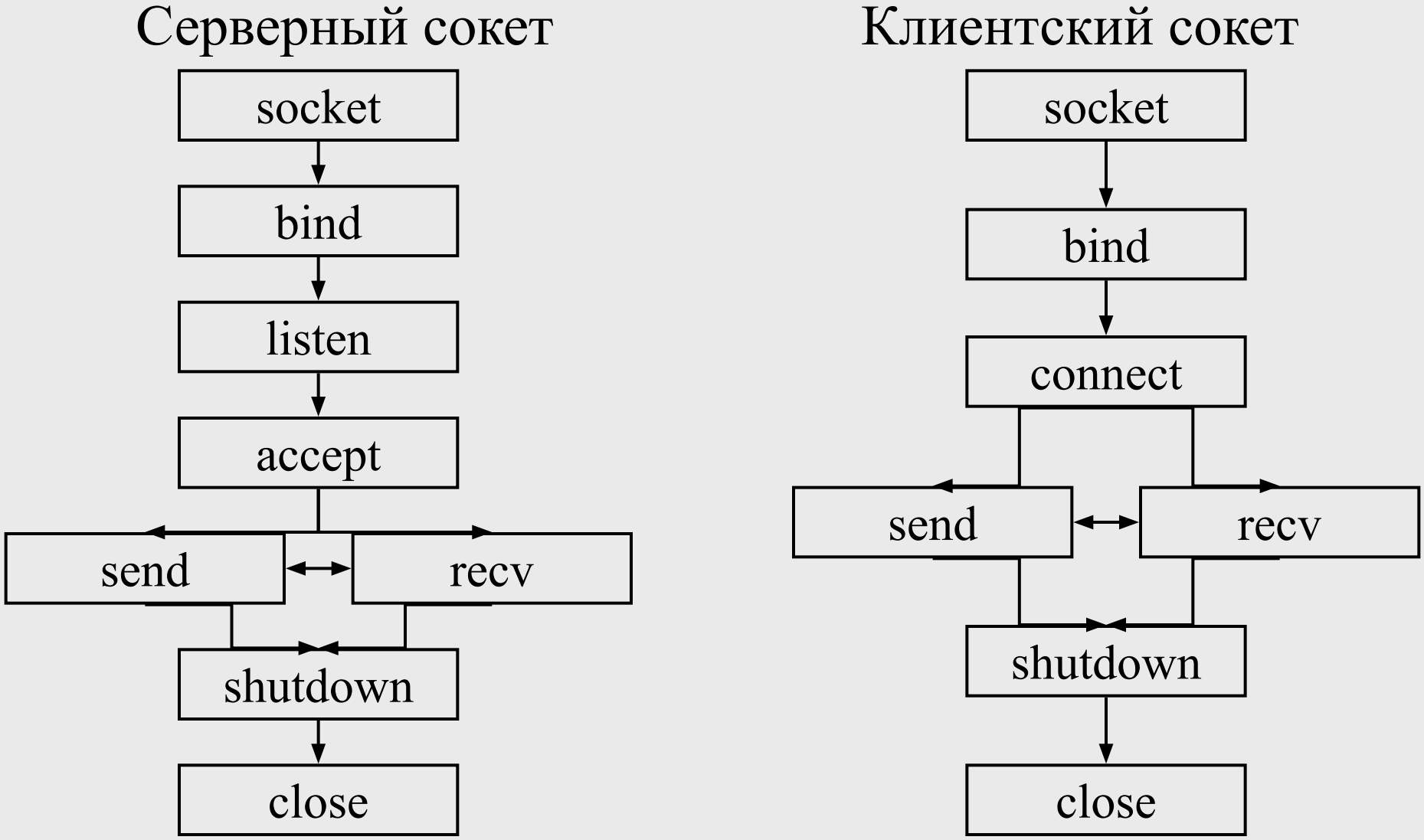
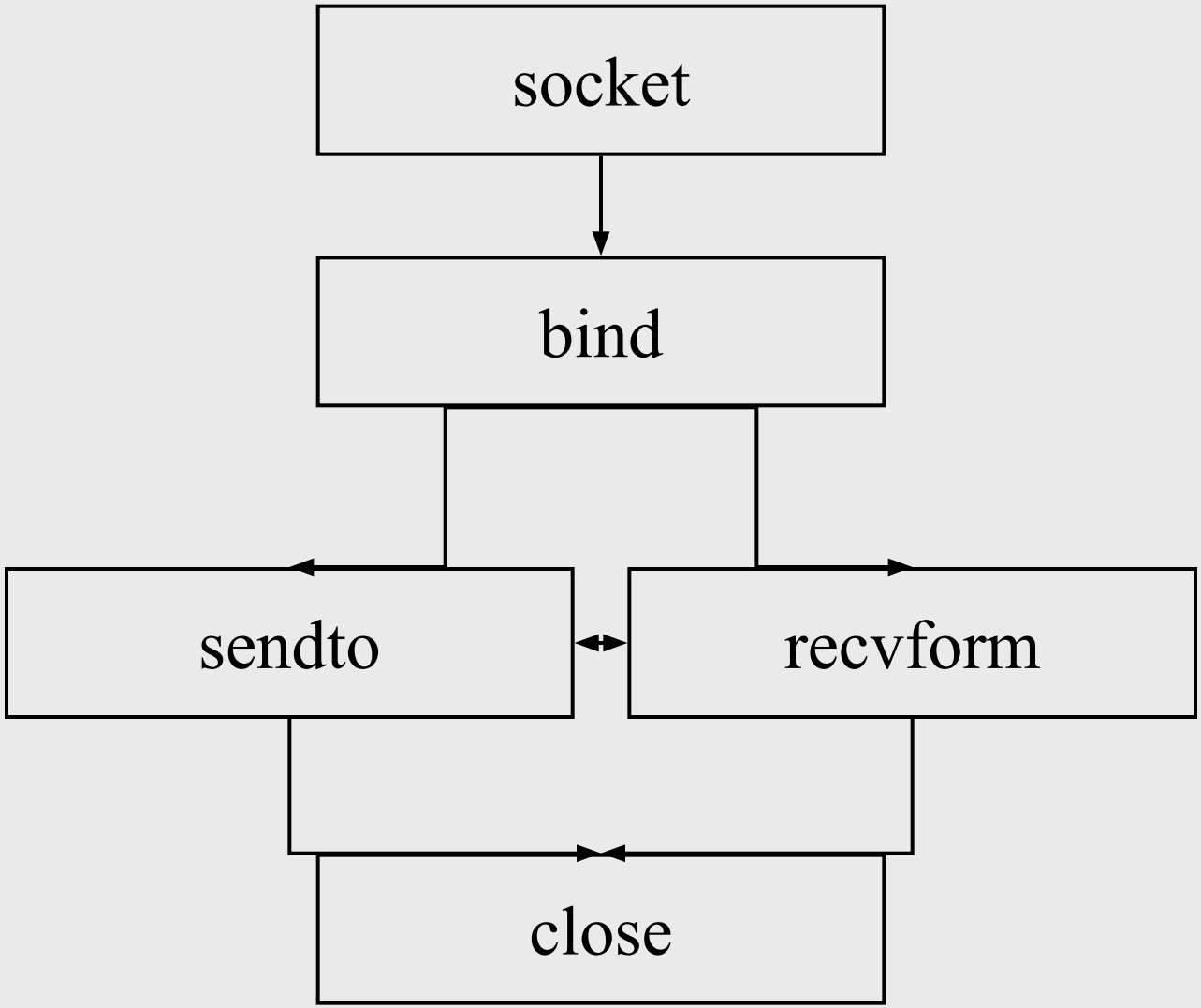


Схема работы с сокетами без установления соединения



Пример. Работа с локальными сокетами

AF_UNIX

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <string.h>
#define SADDRESS "mysocket"
#define CADDRESS "clientsocket"
#define BUFLEN  40
```

```
int main ( int argc, char ** argv)
{
    struct sockaddr_un
    party_addr, own_addr ;
    int sockfd ;
    int is_server ;
    char buf [ BUFLEN ] ;
    int party_len ;
    int quitting ;
    if (argc != 2) {
        printf( "Usage: %s
    client|server.\n", argv [ 0 ] ) ;
        return 0 ;
    }
    …
```

```
…
quitting = 1;
is_server = ! strcmp ( argv [ 1 ] ,
“server” ) ;
memset ( & own_addr, 0, sizeof (
own_addr ) ) ;
own_addr . sun_family =
AF_UNIX ;
strcpy ( own_addr . sun_path,
is_server ? SADDRESS :
CADDRESS ) ;
if ( ( sockfd = socket (
AF_UNIX, SOCK_DGRAM, 0 )
) < 0 ) {
    printf ( “can't create socket\n”
) ;
    return 0 ;
}
```

```
…
unlink ( own_addr . sun_path ) ;  /* связываем сокет */
if ( bind ( sockfd, ( struct sockaddr * ) & own_addr, sizeof (
own_addr . sun_family ) + strlen ( own_addr . sun_path ) ) < 0 )
{
    printf ( “can't bind socket!” ) ;
    return 0 ;
}
if ( ! is_server ) { /* это — клиент  */
    memset ( & party_addr, 0, sizeof ( party_addr ) ) ;
    party_addr . sun_family = AF_UNIX ;
    strcpy ( party_addr . sun_path, SADDRESS ) ;
    printf ( “type the string: ” ) ;
…
```

```
…
while ( gets ( buf ) ) { /* не пора ли выходить? */
    quitting = ( ! strcmp ( buf, “quit” ) ) ;
        /* считали строку и передаем ее серверу */
    if ( sendto ( sockfd, buf, strlen ( buf ) + 1, 0, ( struct
sockaddr * ) & party_addr, sizeof ( party_addr . sun_family ) +
strlen ( SADDRESS ) ) != strlen ( buf ) + 1 ) {
        printf ( “client: error writing socket!\n” ) ;
        return 0 ;
    }
    if ( recvfrom ( sockfd, buf, BUFLEN, 0, NULL, 0 ) <0 ) {
        printf ( "client: error reading socket!\n“ ) ;
        return 0 ;
    }
    …
```

```
        …
        printf ( "client: server answered: %s\n", buf ) ;
        if ( quitting )
            break ;
        printf ( “type the string: ” ) ;
    }               /* while */
    close ( sockfd ) ;
    return 0 ;
}               /* if (!is_server), клиент */
…
```

```
...
   while ( 1 ) { /* получаем строку от клиента и выводим на печать
   */
        party_len = sizeof ( party_addr ) ;
        if ( recvfrom ( sockfd, buf, BUFLEN, 0, ( struct sockaddr * ) &
   party_addr, & party_len ) < 0 )
        {
             printf ( “server: error reading socket!” ) ;
             return 0 ;
  }
        printf ( “server: received from client: %s \n”, buf ) ;
        /* не пора ли выходить? */
        quitting = ( ! strcmp ( buf, “quit” ) ) ;
        if ( quitting ) strcpy ( buf, “quitting now!” ) ;
        else
             if ( ! strcmp ( buf, “ping!” ) ) strcpy ( buf, “pong!” ) ;
             else strcpy ( buf, “wrong string!” ) ;
```

```
        ...
        /* посылаем ответ */
        if ( sendto ( sockfd, buf, strlen ( buf ) + 1, 0, ( struct sockaddr * )
    & party_addr, party_len ) != strlen ( buf ) + 1 )
        {
            printf ( “server: error writing socket!\n” ) ;
            return 0 ;
        }
        if ( quitting )
            break ;
    }   /* while ( 1 ) */
    close ( sockfd ) ;
    return 0 ;
}
```

Пример. Работа с локальными сокетами

AF_INET (GET /<имя файла>)

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

```
#define PORTNUM 8080
#define BACKLOG 5
#define BUFLEN 80
#define FNFSTR "404 Error File Not Found "
#define BRSTR "Bad Request "
```

```
int main ( int argc, char ** argv )
{
    struct sockaddr_in own_addr, party_addr ;
    int sockfd, newsockfd, filefd ;
    int party_len ;
    char buf [ BUFLEN ] ;
    int len ;
    int i ;
    /* создаем сокет */
    if ( ( sockfd = socket ( AF_INET, SOCK_STREAM, 0 ) ) < 0 )
    {
        printf ( “can't create socket\n” ) ;
        return 0 ;
    }
    …
```

```
…
    /* связываем сокет */
    memset ( & own_addr, 0, sizeof ( own_addr ) ) ;
    own_addr . sin_family = AF_INET ;
    own_addr . sin_addr . s_addr = INADDR_ANY ;
    own_addr . sin_port = htons ( PORTNUM ) ;
    if ( bind ( sockfd, ( struct sockaddr * ) & own_addr, sizeof (
    own_addr ) ) < 0 ) {
        printf ( “can't bind socket!” ) ;
        return 0 ;
    }
    /* начинаем обработку запросов на соединение */
    if ( listen ( sockfd, BACKLOG ) < 0 ) {
        printf ( “can't listen socket!” ) ;
        return 0 ;
    }
    …
```

```
…
while ( 1 ) {
    memset ( & party_addr, 0, sizeof ( party_addr ) ) ;
    party_len = sizeof ( party_addr ) ; /* создаем соединение */
    if ( ( newsockfd = accept ( sockfd, ( struct sockaddr * ) &
party_addr, & party_len ) ) < 0 ) {
        printf ( “error accepting connection!” ) ;
        return 0 ;
    }
    if ( ! fork () ) {
        /*это — сын, он обрабатывает запрос и посылает ответ*/
        close ( sockfd ) ; /*  этот сокет сыну не нужен */
        if ( ( len = recv ( newsockfd, & buf, BUFLEN, 0 ) ) < 0 ) {
            printf ( “error reading socket!” ) ;
            return 0 ;
   }
        …
```

```
…
/* разбираем текст запроса */
printf ( "received: %s \n", buf ) ;
if ( strncmp ( buf, "GET /", 5 ) ) { /* плохой запрос! */
    if ( send ( newsockfd, BRSTR, strlen ( BRSTR ) + 1,
0 ) != strlen ( BRSTR ) + 1 ) {
        printf ( “error writing socket!” ) ;
        return 0 ;
    }
    shutdown ( newsockfd, 1 ) ;
    close ( newsockfd ) ;
    return 0 ;
}
…
```

```
…
for ( i =5; buf [ i ] && ( buf [ i ] > ‘ ’ ) ; i++ ) ;
buf [ i ] = 0 ;
/* открываем файл */
if ( ( filefd = open ( buf + 5, O_RDONLY ) ) < 0) {
    /* нет файла! */
    if ( send ( newsockfd, FNFSTR, strlen ( FNFSTR ) +
1, 0) != strlen ( FNFSTR ) + 1 ) {
        printf ( “error writing socket!” ) ;
        return 0 ;
    }
    shutdown ( newsockfd, 1 ) ;
    close ( newsockfd ) ;
    return 0 ;
}
…
```

```
        …
        /* читаем из файла порции данных и посылаем их
клиенту */
        while ( len = read ( filefd, & buf, BUFLEN ) )
            if ( send ( newsockfd, buf, len, 0 ) < 0 ) {
                printf ( “error writing socket!” ) ;
                return 0 ;
            }
        close ( filefd ) ;
        shutdown ( newsockfd, 1 ) ;
        close ( newsockfd ) ;
        return 0 ;
    } /* процесс — отец. Он закрывает новый сокет и
продолжает прослушивать старый */
    close ( newsockfd ) ;
    }    /* while (1) */
}
```