МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» 12 2017

ПРОГРАММИРОВАНИЕ МЕЖПРОЦЕССНОГО ВЗАИМОДЕЙСТВИЯ

Методические указания для проведения лабораторных занятий и выполнения самостоятельной внеаудиторной работы по дисциплине «Теория языков программирования и методы трансляции» для студентов направления подготовки 09.03.04 «Программная инженерия»

Курск 2017

УДК 681.3

Составитель: А.В. Малышев

Рецензент

Кандидат технических наук, начальник отдела информатизации ГУ КРО ФСС РФ А.Ф. Рубанов

Программирование межпроцессного взаимодействия : методические указания для проведения лабораторных занятий и выполнения самостоятельной внеаудиторной работы по дисциплине «Теория языков программирования и методы трансляции» / Юго-Зап. гос. ун-т; сост. А.В. Малышев. Курск, 2017. 15 с.: ил. 18. Библиогр.: с. 13

Содержат сведения по вопросам программирования сокетов для серверных и клиентских приложений. Приведены соответствующие примеры программного кода и задания для самостоятельной работы.

Предназначены для студентов направления подготовки 09.03.04.

Текст печатается в авторской редакции

Подписано в печать . Формат 60x84 1/16
Усл. печ. л. . Уч.-изд. л. . Тираж экз. Заказ. Бесплатно.

Юго-Западный государственный университет.

305040, г. Курск, ул. 50 лет Октября, 94.

1. Цель работы

Изучить интерфейс сокетов, процедуры создания удаления сокетов, установки соединений, приёма и передачи данных для написания серверных и клиентских сетевых приложений.

2. Основы программирования межпроцессного взаимодействия

Интерфейс сокетов был впервые реализован в операционной системе Berkley UNIX. Сейчас этот программный интерфейс доступен практически в любой модификации Unix, в том числе в Linux. Хотя все реализации отличаются друг от друга, основной набор функций в них совпадает. Сокеты были разработаны для программ на C, но в настоящее время средства для работы с ними предоставляют многие языки (Perl, Java и др.).

Сокеты предоставляют весьма мощный и гибкий механизм межпроцессного взаимодействия (IPC). Они могут использоваться для организации взаимодействия программ на одном компьютере, по локальной сети или через Internet, что позволяет создавать распределённые приложения различной сложности. Кроме того, с их помощью можно организовать взаимодействие с программами, работающими под управлением других операционных систем.

Сокеты поддерживают многие стандартные сетевые протоколы (конкретный их список зависит от реализации) и предоставляют унифицированный интерфейс для работы с ними. Наиболее часто сокеты 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, можно написать собственный Web-броузер или Web-сервер, способный обслуживать одновременно множество клиентов.

2.1. Понятие сокета

Сокет (socket) - это конечная точка сетевых коммуникаций. Он является чем-то вроде "портала", через которое 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.

В программе сокет идентифицируется *дескриптором* - это просто переменная типа int. Программа получает дескриптор от операционной системы при создании сокета, а затем передаёт его сервисам socket API для указания сокета, над которым необходимо выполнить то или иное действие.

2.2. Атрибуты сокета

С каждым сокет связываются три атрибута: *домен*, *тип* и *протокол*. Эти атрибуты задаются при создании сокета и остаются неизменными на протяжении всего времени его существования. Для создания сокета используется функция socket, имеющая следующий прототип (рис. 1).

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

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

Рис. 1. Формат функции socket

Домен определяет пространство адресов, в котором располагается сокет, и множество протоколов, которые используются для передачи данных. Чаще других используются домены Unix и Internet, задаваемые константами AF_UNIX и AF_INET соответственно (префикс AF означает "address family" - "семейство адресов"). При задании AF_UNIX для передачи данных используется файловая система ввода/вывода Unix. В этом случае сокеты используются для межпроцессного взаимодействия на одном компьютере и не годятся для работы по сети. Константа AF_INET соответствует Internet-домену. Сокеты, размещённые в этом домене, могут использоваться для работы в любой IP-сети. Существуют и другие домены (AF_IPX для протоколов Novell, AF_INET6 для новой модификации протокола IP - IPv6 и т.д.), но в этой статье мы не будем их рассматривать.

Тип сокета определяет способ передачи данных по сети. Чаще других применяются:

- SOCK_STREAM. Передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных, при котором фрагменты отправленного блока не теряются, не переупорядочиваются и не дублируются. Поскольку этот тип сокетов является самым распространённым, до конца раздела мы будем говорить только о нём. Остальным типам будут посвящены отдельные разделы.
- SOCK_DGRAM. Передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться в пути, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (multicasting) и широковещательная передача (broadcasting).
- SOCK_RAW. Этот тип присваивается низкоуровневым (т. н. "сырым") сокетам. Их отличие от обычных сокетов состоит в том, что с их помощью программа может взять на себя формирование некоторых заголовков, добавляемых к сообщению.

Не все домены допускают задание произвольного типа сокета. Например, совместно с доменом Unix используется только тип SOCK_STREAM. С другой стороны, для Internet-домена можно задавать любой из перечисленных типов. В этом случае для реализации SOCK_STREAM используется протокол TCP, для реализации SOCK_DGRAM - протокол UDP, а тип SOCK_RAW используется для низкоуровневой работы с протоколами IP, ICMP и т. д.

Наконец, последний атрибут определяет протокол, используемый для передачи данных. Часто протокол однозначно определяется по домену и типу сокета. В этом случае в качестве третьего параметра функции socket можно передать 0, что соответствует протоколу по умолчанию. Тем не менее, иногда (например, при работе с низкоуровневыми сокетами) требуется задать протокол явно. Числовые идентификаторы протоколов зависят от выбранного домена; их можно найти в документации.

Прежде чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене (эту процедуру называют именованием сокета). Иногда связывание осуществляется неявно (внутри функций connect и accept), но выполнять его необходимо во всех случаях. Вид адреса зависит от выбранного домена. В Unix-домене это текстовая строка - имя файла, через который происходит обмен данными. В Internet-домене адрес задаётся комбинацией IP-адреса и 16-битного номера порта. IP-адрес определяет хост в сети, а порт - конкретный сокет на этом хосте. Протоколы TCP и UDP используют различные пространства портов.

Для явного связывания сокета с некоторым адресом используется функция bind. Её прототип имеет вид (рис. 2):

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

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

Рис. 2. Формат функции bind

В качестве первого параметра передаётся дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр, addr, содержит указатель на структуру с адресом, а третий - длину этой структуры. Посмотрим, что она собой представляет (рис. 3).

```
struct sockaddr {
    unsigned short    sa_family;    // Семейство адресов, AF_xxx
    char              sa_data[14];  // 14 байтов для хранения адреса
};
```

Рис. 3. Формат структуры sockaddr

Поле sa_family содержит идентификатор домена, тот же, что и первый параметр функции socket. В зависимости от значения этого поля по-разному интерпретируется содержимое массива sa_data. Разумеется, работать с этим массивом напрямую не очень удобно, поэтому можно использовать вместо sockaddr одну из альтернативных структур вида sockaddr_XX (XX - суффикс, обозначающий домен: "un" - Unix, "in" - Internet и т. д.). При передаче в функцию bind указатель на эту структуру приводится к указателю на sockaddr. Рассмотрим для примера структуру sockaddr_in (рис. 4).

```
struct sockaddr_in {
    short int          sin_family;  // Семейство адресов
    unsigned short int sin_port;    // Номер порта
    struct in_addr     sin_addr;    // IP-адрес
    unsigned char      sin_zero[8]; // "Дополнение" до размера структуры sockaddr
};
```

Рис. 4. Формат структуры sockaddr_in

Здесь поле sin_family соответствует полю sa_family в sockaddr, в sin_port записывается номер порта, а в sin_addr - IP-адрес хоста. Поле sin_addr само является структурой, которая имеет вид (рис. 5):

```
struct in_addr {
    unsigned long s_addr;
};
```

Рис. 5. Формат структуры in_addr

Зачем понадобилось заключать всего одно поле в структуру? Дело в том, что раньше in_addr представляла собой объединение (union), содержащее гораздо большее число полей. Сейчас, когда в ней осталось всего одно поле, она продолжает использоваться для обратной совместимости.

И ещё одно важное замечание. Существует два порядка хранения байтов в слове и двойном слове. Один из них называется *порядком хоста* (host byte order), другой - *сетевым порядком* (network byte order) хранения байтов. При указании IP-адреса и номера порта необходимо преобразовать число из порядка хоста в сетевой. Для этого используются функции htons (Host TO Network Short) и htonl (Host TO Network Long). Обратное преобразование выполняют функции ntohs и ntohl.

2.3. Установка соединения (сервер)

Установка соединения на стороне сервера состоит из четырёх этапов, ни один из которых не может быть опущен. Сначала сокет создаётся и привязывается к локальному адресу. Если компьютер имеет несколько сетевых интерфейсов с различными IP-адресами, можно принимать соединения только с одного из них, передав его адрес функции bind. Если же необходимо соединяться с клиентами через любой интерфейс, то нужно задать в качестве адреса константу INADDR_ANY. Что касается номера порта, можно задать конкретный номер или 0 (в этом случае система сама выберет произвольный неиспользуемый в данный момент номер порта).

На следующем шаге создаётся очередь запросов на соединение. При этом сокет переводится в режим ожидания запросов со стороны клиентов. Всё это выполняет функция listen (рис. 6).

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

Рис. 6. Формат функции listen

Первый параметр - дескриптор сокета, а второй задаёт размер очереди запросов. Каждый раз, когда очередной клиент пытается соединиться с сервером, его запрос ставится в очередь, так как сервер может быть занят обработкой других запросов. Если очередь заполнена, все последующие запросы будут игнорироваться. Когда сервер готов обслужить очередной запрос, он использует функцию accept (рис. 7).

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

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

Рис. 7. Формат функции accept

Функция accept создаёт для общения с клиентом *новый* сокет и возвращает его дескриптор. Параметр sockfd задаёт слушающий сокет. После вызова он остаётся в слушающем состоянии и может принимать другие соединения. В структуру, на которую ссылается addr, записывается адрес сокета клиента, который установил соединение с сервером. В переменную, адресуемую указателем addrlen, изначально записывается размер структуры; функция accept записывает туда длину, которая реально была использована. Если адрес клиента не имеет значения, то можно передать NULL в качестве второго и третьего параметров.

Обратите внимание, что полученный от accept новый сокет связан с тем же самым адресом, что и слушающий сокет. Сначала это может показаться странным. Но дело в том, что адрес TCP-сокета не обязан быть уникальным в Internet-домене. Уникальными должны быть только *соединения*, для идентификации которых используются *два* адреса сокетов, между которыми происходит обмен данными.

2.4. Установка соединения (клиент)

На стороне клиента для установления соединения используется функция connect, которая имеет следующий прототип (рис. 8):

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

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

Рис. 8. Формат функции connect

Здесь sockfd - сокет, который будет использоваться для обмена данными с сервером, serv_addr содержит указатель на структуру с адресом сервера, а addrlen - длину этой структуры. Обычно сокет не требуется предварительно привязывать к локальному адресу, так как функция connect

сделает это автоматически, подобрав подходящий свободный порт. Можно принудительно назначить клиентскому сокету некоторый номер порта, используя bind перед вызовом connect. Делать это следует в случае, когда сервер соединяется с только с клиентами, использующими определённый порт (примерами таких серверов являются rlogind и rshd). В остальных случаях системе выбирает порт автоматически.

2.5. Обмен данными

После того как соединение установлено, можно начинать обмен данными. Для этого используются функции send и recv. В Unix для работы с сокетами можно использовать также файловые функции read и write, но они обладают меньшими возможностями, а кроме того не будут работать на других платформах (например, под Windows), поэтому лучше ими пользоваться.

Функция send используется для отправки данных и имеет следующий прототип (рис. 9).

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

Рис. 9. Формат функции send

Здесь sockfd - это, как всегда, дескриптор сокета, через который мы отправляем данные, msg - указатель на буфер с данными, len - длина буфера в байтах, а flags - набор битовых флагов, управляющих работой функции (если флаги не используются, передайте функции 0). Вот некоторые из них (полный список можно найти в документации):

- MSG_OOB. Предписывает отправить данные как *срочные* (out of band data, OOB). Концепция срочных данных позволяет иметь два параллельных канала данных в одном соединении. Иногда это бывает удобно. Например, Telnet использует срочные данные для передачи команд типа Ctrl+C. В настоящее время использовать их не рекомендуется из-за проблем с совместимостью (существует два разных стандарта их использования, описанные в RFC793 и RFC1122). Безопаснее просто создать для срочных данных отдельное соединение.
- MSG_DONTROUTE. Запрещает маршрутизацию пакетов. Нижележащие транспортные слои могут проигнорировать этот флаг.

Функция send возвращает число байтов, которое на самом деле было отправлено (или -1 в случае ошибки). Это число может быть меньше указанного размера буфера. Если необходимо отправить весь буфер целиком, то нужно написать свою функцию и вызывать в ней send, пока все данные не будут отправлены. Она может выглядеть примерно так (рис. 10).

```
int sendall(int s, char *buf, int len, int flags)
{
   int total = 0;
   int n;

   while(total < len)
   {
      n = send(s, buf+total, len-total, flags);
      if(n == -1) { break; }
      total += n;
   }

   return (n==-1 ? -1 : total);
}
```

Рис. 10. Формат функции sendall

Использование sendall ничем не отличается от использования send, но она отправляет весь буфер с данными целиком. Для чтения данных из сокета используется функция recv (рис. 11).

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

Рис. 11. Формат функции recv

В целом её использование аналогично send. Она точно так же принимает дескриптор сокета, указатель на буфер и набор флагов. Флаг MSG_OOB используется для приёма срочных данных, а MSG_PEEK позволяет "подсмотреть" данные, полученные от удалённого хоста, не удаляя их из системного буфера (это означает, что при следующем обращении к recv возвращаются те же самые данные). Полный список флагов можно найти в документации. По аналогии с send функция recv возвращает количество прочитанных байтов, которое может быть меньше размера буфера. Аналогичным образом можно написать собственную функцию recvall, заполняющую буфер целиком. Существует ещё один особый случай, при котором recv возвращает 0. Это означает, что соединение было разорвано.

2.6. Закрытие сокета

Закончив обмен данными, закройте сокет с помощью функции close. Это приведёт к разрыву соединения (рис. 12).

```
#include <unistd.h>

int close(int fd);
```

Рис. 12. Формат функции close

Можно запретить передачу данных в каком-то одном направлении, используя shutdown (рис. 13).

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

Рис. 13. Формат функции shutdown

Параметр how может принимать одно из следующих значений:
0 - запретить чтение из сокета
1 - запретить запись в сокет
2 - запретить и то и другое

Хотя после вызова shutdown с параметром how, равным 2, больше нельзя использовать сокет для обмена данными, всё равно потребуется вызвать close, чтобы освободить связанные с ним системные ресурсы.

2.7. Эхо-клиент и эхо-сервер

Эхо-клиент посылает сообщение "Hello there!" и выводит на экран ответ сервера. Его код приведён на рис. 14. Эхо-сервер читает всё, что передаёт ему клиент, а затем просто отправляет полученные данные обратно. Его код содержится на рис. 15.

2.8. Обмен датаграммами

Как уже говорилось, датаграммы используются в программах довольно редко. В большинстве случаев надёжность передачи критична для приложения, и вместо изобретения собственного надёжного протокола поверх UDP программисты предпочитают использовать TCP. Тем не менее, иногда датаграммы оказываются полезны. Например, их удобно использовать при транслировании звука или видео по сети в реальном времени, особенно при широковещательном транслировании.

Поскольку для обмена датаграммами не нужно устанавливать соединение, использовать их гораздо проще. Создав сокет с помощью socket и bind, можно тут же использовать его для отправки или получения данных. Для этого понадобятся функции sendto и recvfrom (рис. 16).

Функция sendto очень похожа на send. Два дополнительных параметра to и tolen используются для указания адреса получателя. Для задания адреса используется структура sockaddr, как и в случае с функцией connect. Функция recvfrom работает аналогично recv. Получив очередное сообщение, она записывает его адрес в структуру, на которую ссылается from, а записанное количество байт - в переменную, адресуемую указателем fromlen.

Некоторую путаницу вносят *присоединённые датаграммные сокеты* (connected datagram sockets). Дело в том, что для сокета с типом SOCK_DGRAM тоже можно вызвать функцию connect, а затем использовать send и recv для обмена данными. Нужно понимать, что никакого соединения при этом не устанавливается. Операционная система просто запоминает адрес, который был передан функции connect, а затем использует его при отправке данных. Обратите внимание, что присоединённый сокет может получать данные *только* от сокета, с которым он соединён.

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

char message[] = "Hello there!\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(3425); // или любой другой порт...
    addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
    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;
}
```

Рис. 14. Эхо-клиент

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.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(3425);
   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;
         send(sock, buf, bytes_read, 0);
      }
         close(sock);
   }
      return 0;
}
```

Рис. 15. Эхо-сервер

```
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);
```

Рис. 16. Формат функций sendto и recvfrom

Некоторую путаницу вносят *присоединённые датаграммные сокеты* (connected datagram sockets). Дело в том, что для сокета с типом SOCK_DGRAM тоже можно вызвать функцию connect, а затем использовать send и recv для обмена данными. Нужно понимать, что никакого соединения при этом не устанавливается. Операционная система просто запоминает адрес, который был передан функции connect, а затем использует его при отправке данных. Обратите внимание, что присоединённый сокет может получать данные *только* от сокета, с которым он соединён.

Для иллюстрации процесса обмена датаграммами приводятся две небольшие программы - sender (рис. 17) и receiver (рис. 18). Первая отправляет сообщения "Hello there!" и "Bye bye!", а вторая получает их и печатает на экране. Программа sender демонстрирует применение как обычного, так и присоединённого сокета, а receiver использует обычный.

3. Задания на лабораторную работу

3.1. Написать эхо-сервер, использующий протокол TCP и возвращающий принятую информацию построчно.
3.2. Написать эхо-клиент, использующий протокол TCP и возвращающий принятую информацию от сервера построчно.
3.3. Написать эхо-сервер, использующий протокол UDP и возвращающий принятую информацию построчно.
3.4. Написать эхо-клиент, использующий протокол UDP и возвращающий принятую информацию от сервера построчно.

4. Вопросы для самопроверки

4.1. IP-адрес. Классы IP-адресов. Номер порта.
4.2. Передача данных при помощи потоковых сокетов.
4.3. Передача данных при помощи дейтаграммных сокетов.
4.4. Широковещательная и групповая передача данных с помощью дейтаграммных сокетов.
4.5. Системная библиотека сокетов.

Библиографический список

1. Вирт, Никлаус. Алгоритмы и структуры данных. Новая версия для Оберона [Текст] : учебник / пер. с англ. Ф. В. Ткачева. - М.: ДМК-Пресс, 2012. - 272 с.
2. Серебряков, Владимир Алексеевич. Теория и реализация языков программирования [Текст] / В.А. Серебряков. – М.: Физматлит, 2012. - 236 с.

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

char msg1[] = "Hello there!\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(3425);
   addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
   sendto(sock, msg1, sizeof(msg1), 0,
         (struct sockaddr *)&addr, sizeof(addr));

   connect(sock, (struct sockaddr *)&addr, sizeof(addr));
   send(sock, msg2, sizeof(msg2), 0);

   close(sock);

   return 0;
}
```

Рис. 17. Программа sender

```
#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(3425);
   addr.sin_addr.s_addr = htonl(INADDR_ANY);
   if(bind(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
   {
      perror("bind");
      exit(2);
   }
   while(1)
   {
      bytes_read = recvfrom(sock, buf, 1024, 0, NULL, NULL);
      buf[bytes_read] = '\0';
      printf(buf);
   }
      return 0;
}
```

Рис. 18. Программа receiver