Разработка клиент-серверных приложений в Linux

Составил: А.А.Штанюк

20 апреля 2006 г.

Аннотация

В данной статье рассматриваются вопросы разработки программ, использующих технологию "клиент-сервер" .

1 Сокеты

Сокет представляет собой программную абстракцию, механизм для организации взаимодействия программ между собой. Эти программы могут располагаться на одном компьютере, могут выполняться на разных машинах в локальной сети, а могут быть вообще разнесены по разным континентам и взаимодействовать через всемирную сеть. Программа использует сокеты, записывая в них и считывая из них информацию.

Процесс использования сокетов в значительной мере стандартизирован и основан на нескольких библиотечных функциях.

1.1 Создание сокета

Сокеты создаются функцией socket:

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

В качестве параметров функции выступают три значения:

- domain - определяет адреса и протоколы, используемые при взаимодействии сокетов. Чаще всего используются значения **AF_UNIX**, когда необходимо связать процессы на локальной машине и можно задействовать юниксовую систему ввода/вывода и **AF_INET**, - когда необходимо обеспечить связь через Интернет.
- type - 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. Этот тип сокетов является самым распространённым.

  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.
- protocol определяет протокол, используемый для передачи данных. Как мы только что видели, 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.

Результатом работы функции является целое число, которое носит название *дескриптор сокета*. Это значение должно быть положительным. Если функция возвращает -1, то создать сокет не удалось.

После создания сокета функцией **socket**, необходимо произвести его связывание с адресом в выбранном домене (именование сокета). Для этой цели используют функцию **bind**.

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

Вид адреса зависит от выбранного вами 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.

Первый параметр функции - дескриптор сокета.Второй - адрес структуры, задающей параметры для связи сокета:

```
struct sockaddr_in  {
short int sin_family;
unsigned short int  sin_port;
struct in_addr  sin_addr;
unsigned char sin_zero[8];
};
```

Описание полей структуры:

- **sin_family** - Семейство адресов.
- **sin_port** - Номер порта
- **sin_addr** - IP-адрес
- **sin_zero** - "Дополнение" до размера структуры sockaddr

Третий параметр - размер структуры с параметрами.

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**.

1.2 Установка соединения на стороне сервера

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, вы можете принимать соединения только с одного из них, передав его адрес функции **bind**. Если же вы готовы соединяться с клиентами через любой интерфейс, задайте в качестве адреса константу **INADDR_ANY**. Что касается номера порта, вы можете 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**.

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

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**.

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

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 вас не интересует адрес клиента, вы можете просто передать NULL в качестве второго и третьего параметров.

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.

1.3 Установка соединения (клиент)

На стороне клиента для установления соединения используется функция **connect**, которая имеет следующий прототип.

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

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 за вас, подобрав подходящий свободный порт. Вы можете 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 проще и надёжнее предоставить системе выбрать порт за вас.

1.4 Пример: подготовка к передаче данных

Теперь мы можем собрать воедино все подготовительные действия по установлению соединения на стороне сервера и на стороне клиента. Сначала рассмотрим сервер:

```
int sock, listener;
struct sockaddr_in addr;
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);
```

После вызова listen, сервер готов к приему и обработке запросов от клиентов.
Код на стороне клиента:

```
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);
addr.sin_addr.s_addr = htonl(INADDR_LOOPBACK);
if(connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0)
{
       perror("connect");
       exit(2);
}
```

1.5 Обмен данными

После того как соединение установлено, можно начинать обмен данными. Для этого используются функции send и recv.

Функция send используется для отправки данных и имеет следующий прототип.

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

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).

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 вы хотите отправить весь буфер целиком, вам придётся написать свою функцию и вызывать в ней send, пока все данные не будут отправлены.

Для чтения данных из сокета используется функция recv.

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

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 вы получите 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. Вы без труда сможете написать собственную функцию recvall, заполняющую буфер целиком. Существует ещё один особый случай, при котором recv возвращает 0. Это означает, что соединение было разорвано.

Значение, возвращаемое send может отличаться от размера буфера. В этом случае необходимо написать функцию, отправляющую все данные из буфера:

```
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);
}
```

1.6 Закрытие сокета

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

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

Вы также можете запретить передачу данных в каком-то одном направлении, используя shutdown.

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

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, вы больше не сможете использовать сокет для обмена данными, вам всё равно потребуется вызвать close, чтобы освободить связанные с ним системные ресурсы.

1.7 Пример простого приложения без выхода в сеть

В следующем примере мы создаем простой сервер, который принимает от клиентов строку и передает ее им обратно. Получив строку от сервера, клиент выводит ее на терминал. В качестве адреса сервера в клиенте используется константа **INADDR_LOOPBACK**.

Код сервера:

```
// простой сервер

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

```
#include <stdlib.h>
#include <unistd.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;
}
```

Код клиента:

```
// простой клиент

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

1.8 Пример простого приложения с выходом в сеть

В качестве примера с выходом в сеть мы возьмем за основу предыдущую программу и внесем в нее изменения. Во-первых, мы должны добавить новый заголовочный файл

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

для поддержки функции inet_addr. Эта функция нам будет необходима для преобразования строки, содержащей IP-адрес сервера в число, помещаемое в поле sin_addr структуры sockadr_in.

Во-вторых, определим адрес сервера и порт через define:

```
#define SERVER_ADRESS "192.168.254.34"
#define SERVER_PORT 3425
```

В-третьих, добавим возможность для сервера вести журнал регистрации сообщений.

Код сервера:

```
// простой сервер с поддержкой интернета

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

```
#define SERVER_ADRESS "192.168.254.34"
#define SERVER_PORT 3425

int main()
{
    int sock, listener;
    struct sockaddr_in addr;
    char buf[1024];
    int bytes_read;
    FILE *fp;

    listener = socket(AF_INET, SOCK_STREAM, 0);
    if(listener < 0)
    {
        perror("socket");
        exit(1);
    }
    addr.sin_family = AF_INET;
    addr.sin_port = htons(SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(SERVER_ADRESS);
    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;
    fp=fopen("server.log","a");
    fprintf(fp,"%s",buf);
    fclose(fp);
            send(sock, buf, bytes_read, 0);
        }
        close(sock);
    }
    return 0;
}
```

Код клиента:

```
// простой клиент с поддержкой интернета

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

```
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SERVER_ADRESS "192.168.254.34"
#define SERVER_PORT 3425

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(SERVER_PORT);
    addr.sin_addr.s_addr = inet_addr(SERVER_ADRESS);
    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;
}
```