УДК 004.75

Эккерт И.С.,
студент магистратуры
кафедра «Системы обработки информации и управления»
МГТУ им. Н.Э. Баумана
Россия, г. Москва
Тюмин М.О.,
студент магистратуры
кафедра «Системы обработки информации и управления»
МГТУ им. Н.Э. Баумана
Россия, г. Москва
Водолазский И.А.,
студент магистратуры
кафедра «Системы обработки информации и управления»
МГТУ им. Н.Э. Баумана
Россия, г. Москва

ФУНКЦИОНИРОВАНИЕ И РЕАЛИЗАЦИЯ UNIX SOCKETS

Аннотация: *Статья посвящена исследованию существующих возможностей реализации сокетов в unix системах, надежности и основам обмена данными через сокеты.*

Ключевые слова: *передача данных, операционные системы, сокеты, клиент, сервер.*

Annotation: *The article is devoted to the research of existing approaches of implementing sockets in unix systems, reliability and the basics of data exchange through sockets.*

Key words: *data transfer, operating systems, sockets, clients, server.*

Введение

Socket API 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 чем-то отличаются друг от друга, основной набор функций в них совпадает. Изначально сокеты использовались в программах на C/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 вам создавать распределённые приложения различной сложности. Кроме того, с их помощью можно организовать взаимодействие с программами, работающими под управлением других операционных систем. Например, под Windows существует интерфейс Window Sockets, спроектированный на основе socket API. Ниже мы увидим, насколько легко можно адаптировать существующую Unix-программу для работы под Windows [1, с. 420].

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, вы можете написать собственный Web-браузер или Web-сервер, способный обслуживать одновременно множество клиентов.

Как видно, сокеты - весьма мощное и удобное средство для сетевого программирования. В этой статье я покажу, как ими пользоваться. Начав с понятия сокета и самых основных функций для работы с ним, мы постепенно перейдём к обсуждению более сложных тем. В частности, мы рассмотрим использование низкоуровневых сокетов, различные способы организации

параллельного обслуживания клиентов, использование стандартных протоколов Internet и взаимодействие с программами, работающими под управлением операционной системы Microsoft Windows.

Основы socket API

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, с. 236].

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.

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

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 [2, с. 245].

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.

Адреса

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 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.

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

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

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.

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

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, 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) [3, с. 456].

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.

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.

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 сама, 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 проще и надёжнее предоставить системе выбрать порт самостоятельно [3, с. 480].

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) [3, с. 491].

Закрытие сокета

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

#include <unistd.h>

int close(int fd);

Также можно 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

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.

Использование низкоуровневых сокетов

Низкоуровневые сокеты открывают перед нами новые горизонты. Они предоставляют программисту полный контроль над содержимым пакетов, которые отправляются в путешествие по сети. С другой стороны, они сложнее в использовании и обладают плохой переносимостью. Вот почему использовать их следует только в случае необходимости. Например, без них не обойтись при разработке системных утилит типа ping и traceroute [4, с. 494].

Первым делом выясним, чем низкоуровневые сокеты отличаются от

обычных. Работая с обычными сокетами, системе передаются "чистые" данные, а она сама заботится о добавлении к ним необходимых заголовков (а иногда ещё и концевиков). Например, когда посылается сообщение через UDP-сокет, к нему добавляется сначала UDP-заголовок, потом IP-заголовок, а в самом конце - заголовок аппаратного протокола, который используется в вашей локальной сети (например, Ethernet). В результате получается кадр, показанный на рисунке 1.

Рисунок 1. Формат кадра

Низкоуровневые сокеты позволяют включать в буфер с данными заголовки некоторых протоколов. Например, можно включить в сообщение TCP- или UDP-заголовок, предоставив системе сформировать для IP-заголовок, а можно вообще сформировать все заголовки самостоятельно. Разумеется, при этом придётся изучить работу соответствующих протоколов и строго соблюсти формат их заголовков, иначе программа работать не будет [4, с. 502].

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Феннер, Б. UNIX. Разработка сетевых приложений / Б. Феннер. - СПб.: Питер, 2007 – 1040 с.
2. Стивенс, У.Р. Unix. Взаимодействие процессов / У.Р. Стивенс. – СПб.: Питер, 2002 – 1104 с.
3. Чан, Т. Системное программирование на C++ для Unix / Т. Чан. – СПб.: БХВ-Петербург, 1999. Грэй, Д – 520 с.
4. Системное программирование в UNIX. Руководство программиста по разработке ПО / Д. Грэй. – М.: ДМК, 2000 – 670 с.