**ТЕМА: "Основы использования сокетов"**

Цель лабораторной работы: ознакомиться c принципами разработки и функционирования сетевых приложений, использующих сокеты.

## Теоретические основы использования сокетов"

Источник: Жиганов Е.Д. "Разработка сетевых приложений на основе протокола TCP/IP в среде Unix-подобных операционных систем"  
(Методические указания к лабораторному практикуму)

#### **Предварительные замечания о сетевом порядке байтов**

Как правило, в современных компьютерах минимальный элемент оперативной памяти, имеющий уникальный адрес, имеет длину 8 бит (1 байт). И, кроме того, процессоры умеют манипулировать как целым несколькими байтами: двумя, четырьмя, восемью, в зависимости от разрядности процессора. Хранение в памяти двух-, четырех- и восьмибайтовых слов, рассматриваемых как знаковые или беззнаковые целые числа, можно организовать по-разному. Именно, можно хранить самый младший (наименее значимый) байт числа по меньшему адресу, а можно наоборот, по меньшему адресу хранить самый старший (наиболее значимый) байт. Например, в процессорах семейства Intel используется первый способ, а в процессорах Motorola - второй.

Поэтому для того, чтобы компьютеры с разными в этом смысле процессорами могли обмениваться данными по сети, нужно договориться о том, в каком порядке байты будут передаваться по сети. Например, в семействе протоколов TCP/IP принят порядок, обратный по сравнению с тем, какой используется в процессорах Intel, то есть 2-х и 4-х байтовые числа должны передаваться, начиная с самого старшего байта. В этих протоколах в сетевом порядке байтов хранятся, в частности, IP-адрес и номер TCP-порта. Забота о преобразовании данных от локального порядка байтов к сетевому при передаче в сеть и от сетевого к локальному при приеме из сети лежит на программном обеспечении TCP/IP и на прикладном программисте.

Как правило, среди функций, входящих в состав интерфейса прикладных программ, имеются функции для преобразования чисел из локального порядка байтов к сетевому и наоборот. К таким функциям относятся (прототипы описаны в заголовочном файле netinet/in.h):

Изображение выглядит как текст

Автоматически созданное описание

Первая из этих функций преобразует длинное целое (32 бита) от локального порядка байтов к сетевому, вторая выполняет такое же преобразование над коротким целым (16 бит), а вторая пара функций преобразует, соответственно, длинное и короткое целое от сетевого порядка байтов к локальному.

Если вы пишете программу для процессора, у которого порядок байт совпадает с сетевым, то не следует думать, что не нужно использовать функции преобразования, если вы, конечно, специально не хотите сделать вашу программу непереносимой между аппаратными платформами с разным порядком байтов.

### **Разработка сервера**

Основные действия, которые должна выполнить программа-сервер:

* создать сокет (программное гнездо)
* дать сокету имя (адрес)
* объявить сокет могущим принимать соединения
* принять соединение
* по окончании работы закрыть сокет

Ниже последовательно и подробно описаны все эти шаги.

#### **Создание сокета**

Эта операция производится посредством вызова функции socket(), имеющей следующий прототип:



Функция возвращает целое положительное число, называемое дескриптором сокета, вполне аналогичное дескриптору файла. При ошибке: -1. Дескриптор нужно запомнить для последующих операций над сокетом. Первый параметр задает семейство протоколов, по которым будет происходить сетевое общение. Он может принимать, в частности, следующие значения:

* AF\_UNIX, AF\_LOCAL - протокол Unix для локального взаимодействия
* AF\_INET - IP версии 4
* AF\_INET6 - IP версии 6
* AF\_IPX - протоколы IPX, используемые в сетях Novell

Сокет имеет тип, указанный вторым параметром. Тип сокета определяет семантику взаимодействия и может принимать следующие значения:

* SOCK\_STREAM - обеспечивает надежный двусторонний обмен потоками байтов, основанный на установлении соединения. При этом гарантируется правильный порядок байтов (не в смысле сетевого порядка байтов), то есть байты будут приняты в том порядке, в каком они были посланы. Например, для семейства AF\_INET это фактически означает использование протокола транспортного уровня TCP.
* SOCK\_DGRAM - ненадежный обмен на основе передачи дейтаграмм без установления соединения. Например, для семейства AF\_INET это означает использование протокола транспортного уровня UDP.
* SOCK\_SEQPACKET - обеспечивает основанный на установлении соединения надежный упорядоченный двусторонний обмен дейтаграммами фиксированного максимального размера. От получателя требуется, чтобы от читал весь пакет целиком за один системный вызов.
* SOCK\_RAW - непонятно
* SOCK\_RDM - надежная передача дейтаграмм, но без гарантии упорядочения.

Следует заметить, что может оказаться так, что в рамках данного семейства протоколов реализованы не все типы сокетов.

Третий параметр указывает номер конкретного протокола в рамках указанного семейства для указанного типа сокета. Как правило, существует единственный протокол для каждого типа сокета внутри каждого семейства, однако, их может быть и больше. В таких случаях для получения информации о протоколах можно воспользоваться функциями getprotobyname(), getprotobynumber() или группой setprotoent(), getprotoent(), endprotoend().

Чтобы создать, например, TCP-сокет, используем следующий код:

Изображение выглядит как текст

Автоматически созданное описание

#### **Именование сокета**

После создания сокет еще не способен принимать и посылать данные, так как он, хотя уже и существует в определенном пространстве имен, но имени пока не имеет. Именованием сокета называется процедура связывания его с адресом в выбранном домене. Иногда связывание осуществляется неявно (внутри функций connect и accept), но выполнять его необходимо во всех случаях. Для явного именования сокета используется функция bind() со следующим прототипом:



При успешном завершении функция возвращает 0, при ошибке - -1. Первый параметр представляет собой дескриптор сокета, возвращенный функцией socket(). Второй - это то имя (локальный адрес), который мы хотим дать сокету. Третий - длина второго параметра в байтах. Форматы адресов различаются для различных семейств протоколов и различных семейств адресов. Структура sockaddr – это 'родовая' (generic) структура, которая выглядит вот так

Изображение выглядит как текст

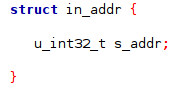
Автоматически созданное описание

Здесь sa\_family - семейство адресов (не путать с семейством протоколов), а массив sa\_data - данные об адресе сокета, специфичные для конкретного семейства адресов. Поле sa\_family - общее для всех семейств протоколов и адресов (правда, оно может по-разному называться). Рассмотрим в качестве примера формат структуры sock\_addr для семейства протоколов AF\_INET:

Изображение выглядит как текст

Автоматически созданное описание

Тип sa\_family\_t эквивалентен типу unsigned short int. Структура in\_addr состоит всего из одного элемента и имеет следующий формат



Поле sin\_family может принимать только одно значение, а именно AF\_INET. Следующее поле в структуре sockaddr\_in - это TCP-порт. Поле s\_addr, входящее в состав структуры in\_addr - IP-адрес (если вы готовы соединяться с клиентами через любой интерфейс, с любым IP-адресом, задайте в качестве адреса константу **INADDR\_ANY)**. Наконец, массив sin\_zero дополняет структуру sockaddr\_in до размера структуры sock\_addr. Отметим, что sin\_port и s\_addr хранятся в сетевом порядке байтов.

Таким образом, для именования созданного TCP-сокета программа должна заполнить структуру sockaddr\_in и вызвать функцию bind(), передав ей указатель на эту структуру вторым параметром:

Изображение выглядит как текст

Автоматически созданное описание

#### **Переключение сокета в режим прослушивания**

Как правило, программы-серверы исполняются в фоновом режиме, ожидая соединений (слушая сеть) от программ-клиентов. После создания и именования сокета он еще не готов принимать соединения от клиентов. Чтобы программа-сервер могла это делать, сокет нужно перевести в прослушивающий режим, что осуществляется посредством вызова функции listen(), имеющей следующий прототип:



При успешном завершении эта функция возвращает 0, при ошибке -1. Первый параметр - дескриптор сокета, второй - максимальное число соединений в очереди соединений, ожидающих обработки.

Таким образом, после вот такого кода

Изображение выглядит как текст

Автоматически созданное описание

наш сокет готов принимать соединения от клиентов.

#### **Принятие запросов на соединение от клиентов**

В ответ на попытку установления соединения со стороны клиента сервер должен принять это соединение. Это делается с помощью функции accept(), прототип которой выглядит так:



Первый параметр – это дескриптор сокета, который должен быть связан с именем посредством функции bind() и переведен в прослушивающий режим вызовом функции listen() до вызова accept().

Функция accept() выполняет следующие действия: извлекает из очереди соединений, ожидающих обработки, первый запрос и создает новый сокет с такими же свойствами, как и sockfd. Если в момент вызова accept() в очереди не было запросов на соединение, то поведение функции зависит от того, в каком режиме находится сокет, блокирующем или неблокирующем. В первом случае программа блокируется до прихода запроса на соединение, во втором функция accept() возвращается с ошибкой (errno будет EWOULDBLOCK или EAGAIN).

Функция возвращает дескриптор вновь созданного сокета, который нужно использовать для обмена данными, но нельзя для приема соединений. Первоначальный сокет sockfd остается открытым и служит для принятия последующих соединений на этом порту.

Второй параметр заполняется самой функцией и по завершении вызова будет содержать информацию об адресе того, кто присоединился (имя удаленного сокета). Третий параметр одновременно является как входным, так и выходным. При вызове он должен содержать размер объекта, на который показывает указатель remote\_addr, а по завершении он будет содержать фактическую длину адреса. Если вас не интересует адрес клиента (вы не планируете устанавливать соединение со стороны сервера), вы можете просто передать NULL в качестве второго и третьего параметров.

Изображение выглядит как текст

Автоматически созданное описание

#### **Чтение из сокета**

Прием данных из сети можно осуществлять посредством функций recvfrom(), recvmsg(), recv() и read(). Первые две функции можно использовать для чтения данных вне зависимости от того, является ли сокет ориентированным на соединение или нет. Последние две используются для приема данных из сокета, ориентированного на соединение.

Функция read() - это обычная функция чтения, с помощью которой мы читаем из файлов и т.п. По сравнению с ней функция recv() ориентирована на работу исключительно с сокетами и обладает более богатыми возможностями. Рассмотрим подробнее функцию recv(). Она имеет следующий прототип



и возвращает при успешном завершении число прочитанных байт, а при ошибке - -1, при этом, как обычно, переменная errno принимает соответствующее значение. Первый параметр функции - сокет, из которого нужно прочитать данные, второй - указатель на область памяти, в которую нужно записать принятые данные, третий - сколько байт читать. С помощью четвертого параметра можно управлять поведением функции. Например, указав в качестве флага MSG\_PEEK, мы прочитаем данные из начала очереди, но после чтения они останутся в очереди. Разные флаги можно комбинировать, объединяя соответствующие константы посредством операции побитного ИЛИ.

Отметим, что по умолчанию только что созданный сокет является блокирующим. В отношении функции recv() это означает, что если в момент ее вызова данных нет, она блокируется до тех пор, пока они не придут из сети.

#### **Запись в сокет**

Посылку данных в сеть можно осуществлять посредством функций sendto(), sendmsg(), send() и write(). Первые две функции можно использовать для записи данных вне зависимости от того, является ли сокет ориентированным на соединение или нет. Последние две используются для записи данных в сокет, ориентированный на соединение.

Функция write() - это обычная функция записи, с помощью которой мы пишем в файлы и т.п. По сравнению с ней функция send() ориентирована на работу исключительно с сокетами и обладает более богатыми возможностями. Рассмотрим подробнее функцию send(). Она имеет следующий прототип



и возвращает при успешном завершении число записанных байт, а при ошибке - -1, при этом, как обычно, переменная errno принимает соответствующее значение. Первый параметр функции - сокет, в который нужно записать данные, второй - указатель на область памяти, из которой нужно взять данные, третий - сколько байт записать. С помощью четвертого параметра можно управлять поведением функции. Например, указав в качестве флага MSG\_DONTROUTE, мы заставим TCP/IP посылать данные в обход обычных средств маршрутизации непосредственно на сетевой интерфейс получателя, что используется, например, различными диагностическими программами и маршрутизаторами. Разные флаги можно комбинировать, объединяя соответствующие константы посредством операции побитного ИЛИ.

#### **Закрытие сокета**

После окончания обмена данными программа должна закрыть сокет(ы), вызвав функцию close(). Прототип этой функции описан в файле unistd.h и имеет вид



Функции нужно передать дескриптор сокета, который нужно закрыть. При успешном завершении функция возвращает 0, при ошибке - -1.

В нижеследующем примере демонстрируется использование всех описанных выше функций работы с сокетом.

Изображение выглядит как текст

Автоматически созданное описание  
Изображение выглядит как текст

Автоматически созданное описание

### **Разработка клиента**

Минимальный набор действий для клиента:

* создать сокет
* установить соединение
* по окончании работы закрыть сокет

Создание сокета описано выше. Чтобы обмениваться данными с сервером, клиент должен установить с ним соединение.

#### **Установление соединения**

Данная операция осуществляется посредством вызова функции connect(), имеющей следующий прототип



При успешном завершении функция возвращает 0, при ошибке - -1. Первый параметр функции - дескриптор сокета, возвращенный вызовом socket(), второй - указатель на структуру, содержащую адрес удаленного сокета, которую нужно заполнить перед вызовом connect() и третий - длина структуры, на которую указывает remote\_addr в байтах. Отметим, что вызов bind() не является необходимым для клиента, так как назначение порта сокету функция connect() сделает сама, а клиенту, вообще говоря, все равно, какой у него порт.

Ниже приведены фрагменты кода, содержащие основные шаги, которые должна выполнить программа-клиент для того, чтобы обменяться данными с программой-сервером.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание  
В этом отрывке используется не описанная ранее функция inet\_aton(). Эта функция наряду с другими применяется для манипуляций с IP-адресами. Прототипы всех таких функций определены в заголовочном файле arpa/inet.h следующим образом:

Изображение выглядит как текст

Автоматически созданное описание

Функция inet\_aton() преобразует IP-адрес, задаваемый первым аргументом из стандартной формы в виде десятичных чисел, разделенными точками в бинарную в сетевом порядке байтов. При успешном завершении возвращается ненулевое значение, а если адрес неправильный, то 0. Результат преобразования помещается в структуру, на которую указывает второй параметр.

Функция inet\_ntoa() выполняет обратное преобразование, то есть преобразует IP-адрес, заданный в двоичном виде в сетевом порядке байтов в стандартную форму числа-точки. Результат хранится в статическом буфере (возвращается указатель), поэтому при последующих вызовах он будет переписан.

Функция inet\_addr() преобразует IP-адрес, задаваемый первым аргументом из стандартной формы в виде десятичных чисел, разделенными точками в бинарную в сетевом порядке байтов. При успешном завершении возвращается результат преобразования, в противном случае INADDR\_NONE (-1). Эта функция устаревшая, поскольку -1=255.255.255.255 представляет собой корректный IP-адрес. Поэтому пользуйтесь inet\_aton().

Функция inet\_network() извлекает из IP-адреса в стандартной текстовой записи числа-точки номер сети в двоичном виде в локальном порядке байтов. При некорректном адресе возвращается -1.

Функция inet\_makeaddr() составляет из номера сети net и номера узла host, заданных в локальном порядке байтов, IP-адрес в сетевом порядке байтов.

Функция inet\_lnaof() извлекает из IP-адреса, заданного ее аргументом, часть, соответствующую узлу (в локальном порядке байтов).

Функция inet\_netof() извлекает из IP-адреса, заданного ее аргументом, часть, соответствующую сети (в локальном порядке байтов).

#### **Отладка программ**

При отладке сервер-клиентного приложения можно обойтись без сети. Достаточно запустить клиента и сервера на одной машине, а затем использовать для соединения адрес *интерфейса внутренней петли* (loopback interface). В программе ему соответствует константа INADDR\_LOOPBACK (не забудьте применять к ней функцию **htonl**!). Пакеты, направляемые по этому адресу, в сеть не попадают. Вместо этого они передаются стеку протоколов TCP/IP как только что принятые. Таким образом моделируется наличие виртуальной сети, в которой вы можете отлаживать ваши сетевые приложения.

**Пример эхо-сервера**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Пример эхо-клиента**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Выше приведены две небольшие демонстрационные программы. Эхо-клиент посылает сообщение "Hello there!" и выводит на экран ответ сервера. Эхо-сервер читает всё, что передаёт ему клиент, а затем просто отправляет полученные данные обратно.

### (источник: http://rsdn.org/article/unix/sockets.xml)

### **Ход работы**

Создается серверное и клиентское приложение. Серверная и клиентская часть должны иметь возможность получать от пользователя номер порта для соединения через параметр командной строки. Если пользователь ничего не ввел, то используется стандартный номер порта, определяемый в исходном тексте программы как константа.

Для сборки приложений создается makefile, в котором должны присутствовать как минимум три цели для сборки: client, server, clean. Последняя цель для сборки производит очистку каталога с исходными текстами от всех уже скомпилированных файлов, оставляя только файлы с исходными текстами программы.

**Серверное и клиентское приложения выполняют следующие действия**

* Серверное приложение садится на определенный порт и ждет соединений по этому порту. Порт должен задаваться при запуске серверной части в командной строке приложения. Если порт не указан, сервер должен садиться на определенный порт по умолчанию.
* Клиентское приложение, используя аутентификацию (протокол обмена между клиентским и серверным приложением, которое позволит однозначно отделить "чужое" клиентское приложение от своего), пытается подключиться к серверному приложению по заданному в командной строке порту. Аутентификация включает в себя авторизацию клиента (по имени пользователя и паролю, которые клиент должен запросить у пользователя) – правильность пары проверяется на сервере.
* При удачном подключении клиентское приложение запрашивает у пользователя команду и входные параметры, необходимые для вычисления задания (смотри варианты ниже) и передает их для вычисления серверу, иначе клиент выводит сообщение об ошибке и закрывается; помимо перечисленных должна быть предусмотрена команда, завершающая работу с сервером.
* Серверное приложение производит необходимые вычисления и передает результаты вычисления клиентскому приложению, после чего ожидает ввода следующей команды. Вычислительная часть должна быть отделена от логики работы сервера (набор функций либо класс(ы) в отдельном файле-библиотеке, заголовочный файл которой подключается в основном коде серверной части).
* Необходимо предусмотреть корректную обработку ошибок работы с протоколом TCP/IP и вывод на консоль диагностирующей информации с завершением работы приложения.

**Вычислительная часть**

На сервере хранятся экспериментальные данные - два файла: BD.txt и BD\_Coords.txt.

Первый файл содержит информацию о физических характеристиках тела в каждой точке (заданной номером ID), изменяющихся во времени, в формате 4 колонок с заголовками:

Time Point ID Temperature Displacement X

Колонки отделены друг от друга символами табуляции. Данные в файле отсортированы по 1, а затем (для равных значений в 1 колонке) по 2 колонке.

Второй файл содержит информацию о координатах каждой точки в формате 4 колонок с заголовками:

Point ID X Y Z

Колонки отделены друг от друга символами табуляции. Данные в файле отсортированы по 1 колонке.

По запросу клиента пользователю сервер пересылает выборку из экспериментальных данных в виде массива либо структуры

|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание | Условие | Формат вывода |
|  | Информация по температуре и перемещению в заданный момент времени | Time равен значению, заданному пользователем | X Y Z Displacement\_X Temperature  (через табуляцию или иной разделитель) |
|  | Информация о заданной точке тела во все моменты времени | Координаты точки X, Y, Z задаются пользователем | Time Displacement\_X Temperature  (через табуляцию или иной разделитель) |
|  | Информация о среднем, максимальном либо минимальном перемещении в каждый момент времени | Пользователь задает, нужно среднее или максимальное значение.  Для каждого значения Time рассчитать среднее арифметическое значение Displacement X по всем точкам | Time Max\_Displacement\_X  Time Mean\_Displacement\_X  Time Min\_Displacement\_X  (через табуляцию или иной разделитель) |
|  | Информация о средней, максимальной либо минимальной температуре в каждый момент времени | Пользователь задает, нужно минимальное, среднее или максимальное значение.  Для каждого значения Time рассчитать максимальное значение Temperature по всем точкам | Time Max\_Temperature  Time Mean\_Temperature  Time Min\_Temperature  (через табуляцию или иной разделитель) |
|  | Информация о температуре и перемещению на некоторой линии, параллельной оси X, в последний момент времени | Координаты Y и Z вводятся пользователем, Time максимально | X Displacement\_X Temperature  (через табуляцию или иной разделитель) |
|  | Информация о температуре и перемещению на некоторой линии, параллельной оси Z, в последний момент времени | Координаты X и Y вводятся пользователем, Time максимально | Z Displacement\_Z Temperature  (через табуляцию или иной разделитель) |
|  | Информация о температуре и перемещению на некоторой линии, параллельной оси Y, в последний момент времени | Координаты X и Z вводятся пользователем, Time максимально | Y Displacement\_Z Temperature  (через табуляцию или иной разделитель) |
|  | Информация о средней, максимальной либо минимальной температуре по слою на некоторой линии, параллельной оси Y, в последний момент времени | Пользователь задает, нужно минимальное, среднее или максимальное значение.  Time максимально | Y Max\_Temperature  Y Mean\_Temperature  Y Min\_Temperature    (через табуляцию или иной разделитель) |

|  |  |  |  |
| --- | --- | --- | --- |
| № варианта | №№ команд для реализации | № варианта | №№ команд для реализации |
| 1 | 1, 3 | 7 | 1, 6 |
| 2 | 2, 3 | 8 | 2, 6 |
| 3 | 1, 4 | 9 | 1, 7 |
| 4 | 2, 4 | 10 | 2, 7 |
| 5 | 1, 5 | 11 | 1, 8 |
| 6 | 2, 5 | 12 | 2, 8 |