Лекция 2. Socket интерфейс

Мы рассматриваем сетевое программирование как использование транспортного уровня для обеспечения прикладных задач. То есть нам надо передавать/получать через сеть некоторые данные, рассматриваемые с точки зрения транспортного уровня как абстрактный массив байтов. Подготовка и использование этих данных — это задача уровня приложений. А нам нужно просто передать эти данные на транспортный уровень. Для решения этих задач можно использовать так называемый socket интерфейс, реализованный в виде набора библиотречных функции.

Прежде чем рассматривать назначение таких функций, уточним круг вопросов, решаемых на транспортном уровне, и также данные, необходимые для связи с удаленными станциями, на сетевом уровне.

Как уже говорилось, для идентификации сторон сетевого обмена на транспортном уровне используется номер порта, а на сетевом уровне — IP адрес. Таким образом, должны быть средства задания этих адресов при передаче и получения их при приеме (чтобы иметь возможность сформировать ответное сообщение). Кроме этого, на транспортном уровне могут применяться разные стратегии доставки, реализуемые в различных протоколах. На практике, транспортный уровень обычно использует два основных протокола: UDP и TCP.

UDP (User Datagram Protocol) реализует передачу данных без каких-либо гарантий. То есть массив данных передается в сеть, но дальнейшая судьба этого сообщения не отслеживается, протокол не предусматривает проверок, контрольных сообщений и т.п. За счет этого протокол работает очень быстро, но надежность доставки не гарантируется. Таким образом, заботы о том, чтобы все необходимые данные дошли до пункта назначения перекладывается на приложение, использующее этот протокол. По логике работы этот протокол напоминает обычную почту, когда отправитель письма уже не может отслеживать процесс его доставки и надеется только на надежность работы самой почтовой службы.

TCP (Transmission Control Protocol) реализует надежную передачу данных. Это означает, что при работоспособной сети все отправленные данные дойдут до пункта назначения. Если какие-то фрагменты данных будут повреждены или потеряются в процессе доставки, то протокол сам организует их повторную передачу. Таким образом, протокол предусматривает разнообразные проверки, повторные перечачи, таймеры ожидания и другие механизмы. Кроме этого TCP вводит понятие соединения как сеанса связи в рамках которого происходит передача данных. Пока соединение является установленным (действующим), протокол гарантирует доставку данных от отправителя к получателю. Соответственно, установление и разрыв соединения являются отдельными процедурами в рамках TCP.

Под транспортным уровнем находится сетевой уровень. И этот уровень оперирует своими протоколами, в частности, IP. В настоящее время параллельно используются две версии IP, это версии 4 и 6. Для наших простейших нужд достаточно знать, что они отличаются форматом IP адреса. В четвертой версии IP адрес занимает 4 байта, в 6 версии - 32 байта. Пока что мы здесь для простоты рассмотрим интерфейс относительно 4 версии IP протокола.

Итак, приложение, собирающееся использовать транспотрный уровень для обмена данными, должно предоставить socket интерфейсу следующую информацию:

— указание на протокол, который надо использовать при обмене данными

— сетевой адрес другого абонента обмена (IP адрес)

— номер порта, который будет использовать транспортный уровень
— сами данные в виде массива байтов

В свою очередь, socket интерфейс предоставляет услуги по заданию адресов отправителя и получателя, отправления и приема пользовательских массивов байтов, диагностике по успешности выполняемых операций.

Рассматриваем socket для Linux. Для Windows есть ряд отличий и при этом надо опрелиться с компилятором. Потом дам пояснения и комментарии.

Для манипуляций с первыми тремя параметрами (собственно, параметрами транспортного и сетевого уровней) в интерфейса используется структура

struct sockaddr_in

Эта структура содержит несколько полей, нас будут интересовать те, которые отвечают за семейство адресов, порт , IP адрес . Они так и называются

```
short    sin_family;
u_short  sin_port;
struct in_addr sin_addr;
```

При отправлении сообщений мы должны заполнить эту структуру необходимыми данными. При получении сообщения мы также имеем возможность получить такую структуру с данными, касающимися отправителя сообщения, и тем самым извлечь и использовать эти данные в дальнейшей работе приложения.

Семество адресов у нас всегда будет AF_INET — интернет адреса. Порт — это просто целое число. А вот с адресом ситуация чуть сложнее, и чтобы не напрягаться, можно воспользоваться специальной функцией, которая даст нам представление адреса в нужной форме.

```
        unsigned short port = 5555;
struct sockaddr_in addr;
struct hostent *hostinfo;
hostinfo = gethostbyname ("127.0.0.1");
if (hostinfo == NULL) ......
addr.sin_family = AF_INET;
addr.sin_port = htons (port);   //  htons(INADDR_ANY)
addr.sin_addr = *(struct in_addr*) hostinfo->h_addr;
```

Вообще говоря, заполнить поле IP адреса можно разными способами. В учебных примерах из Интернета можно их увидеть.

Теперь рассмотрим основные функции socket интерфейса.

Сокет — это канал обмена данными, идентифицируемый целым числом (в точности как дескриптор канала вода/вывода) Перед отправкой сообщений нужно заполнить структуру sockaddr_in в отношении получателя сообщения.

```
socket  (PF_INET, SOCK_STREAM, 0)  (или SOCK_DGRAM)
```

— создать канал для сетевого обмена, задать семейство протоколов и конкретный протокол обмена.

```
bind  (sock, (struct sockaddr*)&addr, sizeof(addr))
```

— ассоциировать сокет с конкретным сетевым интерфейсом (обычно для UDP)

```
close (sock)
```

— закрыть сокет

```
shutdown ()
```

— прекратить всю сетевую активность на сокете

```
connect (sock, (struct sockaddr*)&addr, sizeof(addr))
```

— запросить TCP соединение

```
listen (sock, n)
```

— установить длину очереди необработанных запросов на входящие TCP соединения

```
accept (sock,(struct sockaddr*)&client,&size)
```

— принять и подтвердить запрос на TCP соединение

```
sendto (sock,buf, buflen, 0, (struct sockaddr*)&addr_to, sizeof(addr_to))
```

— отправить сообщение (обычно UDP)

```
recvfrom(sock, buf, BUFLEN, 0, (struct sockaddr*)&addr_from, &size)
```

— прочитать сообщение (обычно UDP)
еще вызовы для TCP:

```
send (sock, buf, buflen, 0)
```

— отправить сообщение TCP

```
recv(sock, buf, BUFLEN, 0)
```

— прочитать сообщение TCP
а также чтение и запись через общие функции ввода/вывода (для TCP)

```
read (sock, buffer, length)
```

```
write (sock, buffer, length)
```

Вспомогательные функции

```
setsockopt   getsockopt
```

```
htonl htons ntohl ntohs
```

преобразование litte|big endian в формат, принятый при сетевой передаче

Примитивный пример.
Одна “станция” отправляет сообщение. Другая его принимает и отправляет обратно добавив в конце слово Echo. Первая станция распечатывает этот ответ. По ходу дела станции также печатают на экран всякую служебную информацию. Примеры для протоколов UDP и TCP.