Полное руководство по сетевому программированию для разработчиков игр. Часть 3. UDP

Автор: **x84**

"Раньше мои волосы были сухие и безжизненные, теперь...
они МОКРЫЕ и ШЕВЕЛЯТСЯ!!"
реклама шампуня "два в одном"

Да! Раньше наш сокет был сухой и безжизненный, теперь мы заставим его шевелиться! В этой части мы (НАКОНЕЦ-ТО!!!) узнаем, как отсылать и принимать данные (пока только по протоколу UDP).

Одноранговый обмен данными

Логически правильно начинать с простого и переходить к более сложному, поэтому мы начнем с UDP. Это, однако, вовсе не означает, что UDP менее важен, чем TCP, вовсе нет. Все зависит от того, что за игру ты хочешь сделать. Для некоторых игр/жанров больше подходит UDP, для других - TCP. Главное сделать правильный выбор ("И почему я не послал его подальше и не взял синюю таблетку?! Расскажи он мне тогда все полностью - я бы засунул красную пилюлю ему в задницу!!" (c) Warner Bros - Матрица).

UDP... Что такое "одноранговая передача данных"? Это когда взаимодействуют компьютеры, которые имеют одинаковые права по управлению обменом данными и одинаковые приоритеты (peer-to-peer - "равный равному", не путать с point-to-point protocol). Взаимодействие по протоколу UDP можно сравнить с отправкой и приемом почты. Отправив пакет, остановить его движение нельзя. Как уже было сказано, этот протокол не гарантирует доставку данных, то есть, отправляя данные, надо иметь ввиду, что адресат может их и не получить. Что самое

Общий алгоритм сетевого приложения, работающего через UDP выглядит так:

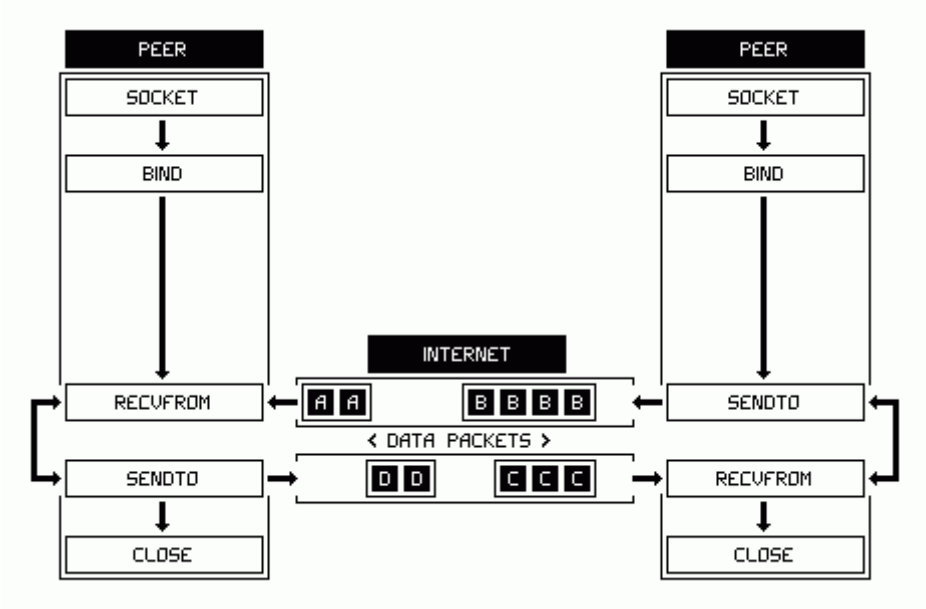
1. Инициализируем (если надо) необходимые библиотеки и создаем сокет. В качестве второго параметра сокета надо передать константу SOCK_DGRAM - мы используем дэйтаграммы.

2. Если необходимо, привязываем сокет к определенному адресу и порту.

3. Отправляем/принимаем пакеты

4. Закрываем сокет и производим очистку

Допустим, нам надо, чтобы один peer передал другому две строки текста (две последовательности байт) – "AA" и "BBBB". А другой peer передал первому другие две строки – "CCC" и "DD". Вот, как это выглядит на схеме:



Обратите внимание, на то, что происходит с пакетами в Интернет. Как видим, каждая строка передается в своем собственном "конверте"-пакете. То есть они вполне самостоятельны и идут по сети отдельно друг от друга. Все логично.

Первый, второй и четвертый пункты нам уже ясны, и мы знаем как произвести соответствующие действия.

данные, адрес - это IP адрес получателя, имя - это порт который прослушивает получатель на наличие входящих данных. Мы уже знаем, что адреса и порты у нас хранятся в адресных структурах. Однако наша обертка для адресных структур еще не завершена, мы должны добавить в нее возможность смены порта и адреса после вызова конструктора.

Вот как выглядят соответствующие методы:

```
// Listing 3.01 win & nix

void _sock_addr::set_port (unsigned short port)
{
  address->sin_port = htons (port);
}

void _sock_addr::set_ip (const char * ip)
{
  address->sin_addr.s_addr = inet_addr (ip)
  if (address->sin_addr.s_addr == INADDR_NONE)
    throw
    _sock_exception ("_sock_addr::set_ip - the provided IP address seems to be invalid");
}
```

Для чего нам эти методы? Щас узнаем! :) Дело в том, что они позволяют нам использовать одну и ту же адресную структуру для отправки дэйтаграмм разным адресатам, мы просто задаем нужные порт и адрес - и отправляем. Это позволяет нам избежать ненужных операций выделения/освобождения памяти.

Чтобы отправить кому-то дэйтаграмму надо знать адрес получателя. Для этого с получателем либо "договориться заранее" или отправителю должен быть известен адрес постоянного проживания. Если на компьютере получателя несколько сетевых интерфейсов – то он должен при помощи bind() выбрать один из них, иначе при автоматической привязке ему будет назначен какой-то из адресов, по которому пришла первая дэйтаграмма.

Вот так выглядит код для отправки сообщений по протоколу UDP на языке C:

```
// Linux & FreeBSD

int sendto ( int s, const void * msg, int len, int flags,
             const struct sockaddr * to, int tolen);

// Windows

int sendto ( SOCKET s, const char * buf, int len, int flags,
             const struct sockaddr * to, int tolen);
```

Теперь посмотрим, что к чему... Первый параметр — это дескриптор сокета, через который надо отправлять данные. Следом идет указатель на последовательность байтов (второй параметр), которые составляют тело сообщения (наше письмо другому компьютеру). Дальше нам надо указать длину нашего сообщения (третий параметр). Затем идет число, которое определяет, каким образом система будет обслуживать данную дэйтаграмму (в большинстве случаев нас устроит способ обслуживания по умолчанию - значение 0). Как видно, четвертый параметр формируется из набора флагов, которые мы рассмотрим далее. Пятый параметр - это адресная структура, содержащая адрес и порт получателя дэйтаграммы. И, наконец, шестой параметр (tolen) - это длина указанной адресной структуры (мы уже знаем, что адресные структуры бывают разные, и, соответственно размер у них тоже варьируется). В качестве шестого параметра мы должны указать sizeof(struct sockaddr_in) (равно 16-ти на архитектуре Intel x86). Для чего указывать размер структуры? Очень просто - эта функция должна работать со всеми уровнями модели OSI, поэтому она не знает, с каким семейством адресов имеет дело - мы должны ей сказать об этом.

sendto возвращает количество отосланных байт. "Как?! Разве она не отсылает данные целиком и полностью?!" - спросишь ты. Ответ: да, этот вызов по отношению к дэйтаграммам всегда возвращает число, равное указанной в третьем параметре длине сообщения. В самом простом случае (который мы сейчас рассматриваем) функция sendto() возвратит либо длину отосланного сообщения, либо -1 (*nix) или SOCKET_ERROR (Windows) в случае ошибки при отсылке (напомню, что -1 и SOCKET_ERROR - одно и то же, но константу SOCKET_ERROR определяют разработчики winsock, и мы не будем этому противиться). Также возможно указать 0 в качестве длины сообщения, тогда и возврат будет равен нулю, что абсолютно легально для операционной системы. Но какой смысл в этом? Смысл, оказывается, все-таки есть, причем он вовсе не тривиален, поэтому мы отложим это на потом...

Для того, чтобы понять, что же все-таки делает sendto, мы вспомним, что когда мы создали сокет, система выделила место для буфера исходящих сообщений (пакетов). Функция sendto на самом деле ничего не отсылает, она лишь только помещает сообщение в вышеуказанный буфер. Дальнейшая судьба сообщения полностью зависит от системы. Размер буфера тоже определяется системой, исходя из пропускной способности интерфейса (сетевой карты), размера свободной памяти и других факторов. Мы имеем шанс получить на выходе из sendto -1, когда в

Ранее мы говорили о том, что сокету вовсе не обязательно назначать "имя", система сама осуществит привязку при первом обращении к сокету. Если мы до сих пор не вызвали bind(), то во время вызова sendto() система выберет подходящий интерфейс (подходящий адрес с точки зрения оптимального маршрута сообщения, исходя из адреса получателя, указанного в sendto) и любой доступный порт из списка эфемерных (временных). Узнать порт и адрес, к которым привязан сокет можно при помощи функции getsockname().

Выглядит она так:

```
// Linux & FreeBSD

int getsockname (int s, struct sockaddr * name, int * namelen);

// Windows

int getsockname (SOCKET s, struct sockaddr * name, int * namelen);
```

s - это дескриптор сокета, чье "имя" нам надо узнать. name - указатель на адресную структуру, куда будут записаны все данные об "имени" сокета. namelen - это длина структуры, которая была записана в name. После этого мы можем использовать поля этой структуры, чтоб узнать порт и адрес нашего сокета.

Все очень просто:

```
// если мы уверены в том, что сокет был создан в семействе протоколов PF_INET, то
// мы можем безбоязненно использовать следующий код:

struct sockaddr_in * name = new struct sockaddr_in;
int namelen = sizeof (struct sockaddr_in);

int error = getsockname (sd, (struct sockaddr *) name, &namelen);
if (error == -1)
{
    // обработать ошибки
}

cout << "The socket IP address is: " << ntohs (name->sin_port) << endl
  << "The socket port number is: " << inet_ntoa (name->sin_addr) << endl;
```

Итак, теперь мы знаем, что успешный вызов sendto вовсе не означает успешную доставку сообщения адресату. Мы просто помещаем его в исходящую очередь. С этого момента мы уже не можем повлиять на его судьбу (вернее, можем, но пока еще не умеем :)) После этого система берет на себя обязательство по дальнейшему распоряжению отправкой дэйтаграммы. После отправки (после того, как сообщение покинет компьютер отправителя) сообщение будет зависеть от промежуточных компьютеров, до тех пор, пока не достигнет адресата.

Страницы: **1** 2 Следующая »

#UDP, #сетевое программирование, #сокеты

31 июля 2003 (Обновление: 24 сен 2009)