Сетевое программирование

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

19 мая 2018 г.

Версия: 0.10

Аннотация

В данной заметке показано писать сетевые приложения (программы). Приложение состоит из двух программ-частей: сервера и клиента. В данной заметке показано как каждую их частей писать. Программы основаны на TCP/IP архитектуре.

Детали того что такое сетевое программирование является предметом отдельной заметки. В данной даны детали именно кодинга.

Предварительная версия!

1 Сетевое программирование

Главное что необходимо написать две программы. Одна отвечает на серверную часть, а другая за клиентскую.

1.1 Клиентская часть

Рассмотрим клиентскую часть сетевого приложения. Клиент подключается к серверу, посылает свои запросы и ждет от сервера ответа.

Дабы не зависеть от серверной части будем подключатся к существующим серверам, а именно – к HTTP серверам.

Исходный код данной части выложен на сервере.

Создание сокета Сокет обеспечивает пересылку данных от данного компьютера на другой. Аналог файлового дескриптора (FILE), но для взаимодействия по сети.

```
1 int s = socket(AF_INET, // IPv4
2   SOCK_STREAM, // потоковый, т.е. TCP
3   0 /* протокол по умолчанию  */);
```

AF_INET указывает, что речь идет о современных IP сетях по протоколу версии 4. SOCK_STREAM указывает на потоковое подсоединение, т.е. применительно к IP это подразумевает протокол TCP.

Данная операция фактически имеет отношение к операционной системе, т.е. именно она в конце концов создает необходимый объект. Последнее означает, что, если, например, у пользователя права на сетевое взаимодействие отсутствуют, то сокет не удасться создать. Поэтому необходимо обязательно проверить успешность создания сокета:

```
1 if( s == -1 )
2 {
3   printf("Сокет не удалось создать.");
4   return -1;
5 }
```

Формирование адреса сервера Клиент должен подключится к серверу, а не наоборот. Поэтому у сервера есть адрес (ip адрес плюс номер порта), который необходимо знать для подключения к нему.

Для задания данных параметров используется структура `struct sockaddr_in`. В поле `sin_addr.s_addr` указывается ip адрес сервера в шестнадцатеричной системе, а в поле `sin_port` номер порта. Ввиду того, что данные поля должны быть указаны в совместимом с другими системами формате, для заполнения этих полей принято использовать вспомогательные функции. Для заполнения поля `sin_addr.s_addr` используется функция `inet_addr`, а – функция `htons`. Данные функции обеспечивают, что порядок следования байт в этих числах один и тот же. Функция `inet_addr` естественно делает ещё и некий разбор текстовой строчки и элементарные преобразования.

Цель нашего тестового примера подключится к серверу `машинноезрение.рф` и считать страницу `html` по протоколу `http`. IP адрес сервера `машинноезрение.рф` является `178.140.230.40`, а порт для взаимодействия по протоколу `http` является `80`. В таком случае код будет таким:

```
1 // Структура для указания адреса сервера.
2 struct sockaddr_in srv;
3 // Сначала лучше структуру обнулить.
4 memset( &srv, 0, sizeof( srv ) );
```

```
5
6  // Строчка содержащая ip адрес.
7  const char *ip_str =
8    // Адрес сервера машинноезрение.рф:
9    "178.140.230.40";
10   // Локальный адрес:
11   //"127.0.0.1";
12 srv.sin_family = AF_INET; // Для IPv4.
13 srv.sin_port = htons( 80 ); // Порт отвечающий за HTTP.
```

Подключение После того как структура с полным адресом (ip4 и порт) задана можно попытаться выполнить подсоединение к серверу. Делается это посредством функции `connect`,

```
1 // Подключаемся к серверу.
2 if( connect(s, (struct sockaddr *)&srv, sizeof(srv) ) < 0 )
3 {
4   printf("Не удалось подключится к серверу.\n");
5   return -1;
6 }
```

где передается сокет (`s`), который будет использоваться для подключения, указатель на структуру содержащую адрес куда подключаемся (`srv`) и размер структуры (`sizeof(srv)`). Ввиду того, что функция `connect` носить общий характер, необходимо преобразовать указатель к общему виду, а также передать размер используемой структуры.

Данная функция также может выдать ошибку (тут важны не только права в данной системе, а и то что сервер не заблокирован и включен). Поэтому обязательно необходимо делать проверку на успешность.

После успешного подключения можно обмениваться данными с сервером.

Обмен данными После того как выполнены все формальности, т.е. создан сокет и выполнено подключение к нужному серверу, можно посылать и принимать сообщения. Последнее можно делать как с помощью стандартных функций, используемых с файлами (`read`, `write`), так и посредством специализированных для сетевого обмена (`recv`, `send`). Последние обладают большими возможностями, например, вызов можно сделать как блокирующим, так и не блокирующим.

В нашем примере будет применять специализированные, но расширенными возможностями не будет пользоваться (соответствующее поле будет равно `0`).

Для пересылке сообщения используется функция `send`.

```
1  const char *msg = "GET /test_socket.html HTTP/1.1\r\n\
2  Host: xn--80akaaied0aladi2a9h.xn--p1ai\r\n\r\n";
3  int msg_len = strlen( msg ); // Длина сообщения.
4
5  // Отсылаем запрос.
6  if( send(s, msg, msg_len, 0) < 0)
7  {
8    printf("Запрос не удался.\n");
9    return -1;
10 }
```

Первый аргумент указывает соект (`s`) через который будет передано сообщение. Второй и третий аргумент задают сообщение, т.е. указывают на область памяти, где сообщение находится, а также передают его размер. Последний аргумент задает дополнительные возможности, которыми мы не будет пользоваться.

Сообщение составлено в согласии с протоколом HTTP и является запросом на страницу `test_socket.html` с сервера `xn80akaaied0aladi2a9h.xnp1ai`, где последнее является переводом `машиноезрение.рф` на латиницу.

Не вдаваясь в детали протокола HTTP, можно считать, что сообщение задано корректно и понятно. Ответом на данное сообщение является текст запрашиваемой страницы.

Он считывается следующим образом:

```
1  char reply[ 2001 ];
2  while( ( ret = recv(s, reply, 2000, 0) ) > 0 )
3  {
4    reply[ret] = 0;// Создаем нуль строку.
5    printf("%s", reply);
6    if( ret == 0 )
7      break;
8  }
```

При считывании данных и потокового соединения (SOCK_STREAM) возвращено может быть меньший объем данных, чем был запрошен. Поэтому считывание обычно делается в цикле. В данном случае при считывании очередного фрагмента данные выводятся в консоль.

Тогда вся страница будет выведена в консоль. Чтобы можно было эту страницу потом увидеть, нужно запускать программу так:

```
1  ./a.out >> test.html
```

1.2 Серверная часть

Серверная часть ожидает подключения клиентов. После успешного подключения клиента сервер отвечает на его запросы.

Исходный код данной части выложен на сервере.

Создание сокета Создает по полной аналогии с клиентской частью. Единственное, что следует подчеркнуть, что сначала создается сокет самого сервера:

```
1  int s_srv = socket(AF_INET, // IPv4
2    SOCK_STREAM, // потоковый, т.е. TCP
3    0 /* протокол по умолчанию  */);
4
5  // Проверяем успешность создания серверного сокета.
6  if( s_srv == -1 )
7  {
8    printf("Сокет не удалось создать.");
9    return -1;
10 }
```

Данный сокет отвечает за ожидание подключения от клиентов. При выполнении подключения для каждого клиента будет создан отдельный сокет.

Привязка порта Далее необходимо привязать порт к нашей программе, а точнее к нашему сокету. Последнее означает, что когда будут поступать соединения, то они будут появляться на данном сокете.

Для начала необходимо заполнить структуру указывающею параметры нашего сервера:

```
1  struct sockaddr_in srv;
2  memset( &srv, 0, sizeof( srv ) );
3
4  srv.sin_family = AF_INET; // Для IPv4.
5  srv.sin_addr.s_addr = INADDR_ANY; // Любой сетевой интерфейс.
6  srv.sin_port = htons( 1234 ); // Наш порт.
```

Наиболее значимое поле является `srv.sin_port`, которое и задает порт, к которому будут подключатся клиенты. Поле `srv.sin_addr.s_addr` задает наш внутренний адрес. В общем случае его можно настраивать (если больше одного сетевого интерфейса). В нашем случае нам все-равно, поэтому указывается параметр `INADDR_ANY`, который обозначает соединение с любого сетевого интерфейса.

После того как структура заполнена привязка осуществляется так:

```
1 if( bind( s_srv, (struct sockaddr *)&srv, sizeof(srv) ) < 0 )
2 {
3   printf("Порт не удалось привязать к сокету.");
4   return -1;
5 }
```

Привязка является системной операцией, поэтому в ней может быть дан отказ. В частности, если такой порт уже кем-то зарезервирован. Поэтому очень важно выполнить проверку на успешность выполненной операции.

Теперь любые соединения с компьютером, на котором запущена данная программа, по данному порту будет поступать на сокет `s_srv`.

Настройка Серверный сокет можно достаточно тонко настраивать. Наиболее важной традиционной настройкой является указание того, сколько клиентов могут быть в очереди на подключение.

Последнее достигается вызовом `listen`:

```
1 listen( s_srv, 3);
```

Теперь очередь состоит из трех ожидающих подключение. При превышении данного количества клиентам будет отказано в доступе к серверу.

Ожидание подключения После того как все настроено сервер ожидает подключение от клиентов. Для этого необходимо вызвать функцию `accept`.

```
1 int c = sizeof(struct sockaddr_in);
2 struct sockaddr_in clnt;
3 // Ожидаем и принятие подключения от клиента.
4 int s_clnt = accept(s_srv, (struct sockaddr *)&clnt, (socklen_
5 if( s_clnt < 0 )
6 {
7   printf("Не удалось установить соединение с клиентом.\n");
8   return 1;
9 }
```

Функция возвращается при поступлении подключения (или ошибки). Наиболее важным является именно возвращенное значение. В случае успеха оно задает сокет, который используется для общения с клиентом.

В идеале здесь необходимо вызвать создать нить с переданным сокетом. Тогда данная нить продолжит общаться с клиентом. В основной же программе (нити)

можно опять вызвать функцию accept, для ожидания следующего клиента. Если же обрабатывать запросы клиента в основной нити (тем самым не вызывать функцию accept), то больше никто не сможет подсоединится к серверу.

В качестве примера приведем код сервера, который будет каждую переданную ему строчку превращать в заглавные буквы.

```
1  int rcv_sz;
2  char clnt_msg[ 2000 ];
3  while( ( rcv_sz = recv( s_clnt , clnt_msg , 2000 , 0) ) > 0 )
4  {
5    // Получили сообщение размера rcv_sz.
6    // Если получили строчку с нужным кючевым словом STOP
7    if( strncmp( clnt_msg, "STOP", rcv_sz < 4 ? rcv_sz : 4 ) ==
8      break;// то выходим из цикла.
9    // Иначе обрабатываем строчку. В данном случае
10   for( int i = 0; i < rcv_sz; i++)// каждый символ
11     clnt_msg[ i ] = toupper( clnt_msg[ i ] ); // делается
   заглавным.
12   // Передаем сообщение обратно клиенту. Размер совпадает
   с полученной строчкой.
13   write( s_clnt , clnt_msg , rcv_sz );
14 }
```