**Л25. Работа с WinSocket в Visual C++**

Socket (гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью. При использовании протоколов TCP/IP можно говорить, что socket является средством подключения прикладной программы к порту (см. выше) локального узла сети.

Socket-интерфейс представляет собой просто набор системных вызовов и/или библиотечных функций языка программирования СИ, разделенных на четыре группы:

Ниже рассматривается подмножество функций socket-интерфейса, достаточное для написания сетевых приложений, реализующих модель "клиент-сервер" в режиме с установлением соединения.

1. Функции локального управления

Функции локального управления используются, главным образом, для выполнения подготовительных действий, необходимых для организации взаимодействия двух программ-партнеров. Функции носят такое название, поскольку их выполнение носит локальный для программы характер.

## *1.2. Связывание socket'а*

Для подключения socket'а к коммуникационной среде, образованной вычислительной сетью, необходимо выполнить системный вызов bind, определяющий в принятом для сети формате локальный адрес канала связи со средой. В сетях TCP/IP socket связывается с локальным портом. Системный вызов bind имеет следующий синтаксис:

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

int bind (s, addr, addrlen)

int s;

struct sockaddr \*addr;

int addrlen;

Аргумент s задает дескриптор связываемого socket'а.

Аргумент addr в общем случае должен указывать на структуру данных, содержащую локальный адрес, приписываемый socket'у. Для сетей TCP/IP такой структурой является sockaddr\_in.

Аргумент addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

Структура sockaddr\_in используется несколькими системными вызовами и функциями socket-интерфейса и определена в include-файле in.h следующим образом:

struct sockaddr\_in {

short sin\_family;

u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8];

};

Поле sin\_family определяет используемый формат адреса (набор протоколов), в нашем случае (для TCP/IP) оно должно иметь значение AF\_INET.

Поле sin\_addr содержит адрес (номер) узла сети.

Поле sin\_port содержит номер порта на узле сети.

Поле sin\_zero не используется.

Определение структуры in\_addr (из того же include-файла) таково:

struct in\_addr {

union {

u\_long S\_addr; /\*

другие (не интересующие нас)

члены объединения \*/

} S\_un;

#define s\_addr S\_un.S\_addr

};

Структура sockaddr\_in должна быть полностью заполнена перед выдачей системного вызова bind. При этом, если поле sin\_addr.s\_addr имеет значение INADDR\_ANY, то системный вызов будет привязывать к socket'у номер (адрес) локального узла сети.

В случае успеха bind возвращает 0, в противном случае - "-1".

## 2. Функции установления связи

Для установления связи "клиент-сервер" используются системные вызовы listen и accept (на стороне сервера), а также connect (на стороне клиента). Для заполнения полей структуры socaddr\_in, используемой в вызове connect, обычно используется библиотечная функция gethostbyname, транслирующая символическое имя узла сети в его номер (адрес).

## *2.1. Ожидание установления связи*

Системный вызов listen выражает желание выдавшей его программы-сервера ожидать запросы к ней от программ-клиентов и имеет следующий вид:

#include <sys/socket.h>

int listen (s, n)

int s;

int n;

Аргумент s задает дескриптор socket'а, через который программа будет ожидать запросы к ней от клиентов. Socket должен быть предварительно создан системным вызовом socketи обеспечен адресом с помощью системного вызова bind.

Аргумент n определяет максимальную длину очереди входящих запросов на установление связи. Если какой-либо клиент выдаст запрос на установление связи при полной очереди, то этот запрос будет отвергнут.

Признаком удачного завершения системного вызова listen служит нулевой код возврата.

## *2.2. Запрос на установление соединения*

Для обращения программы-клиента к серверу с запросом на установление логической соединения используется системный вызов connect, имеющий следующий вид #include <sys/types.h> #include <sys/socket.h> #include <netinet/in.h> int connect (s, addr, addrlen) int s; struct sockaddr\_in \*addr; int addrlen;

Аргумент s задает дескриптор socket'а, через который программа обращается к серверу с запросом на соединение. Socket должен быть предварительно создан системным вызовом socketи обеспечен адресом с помощью системного вызова bind.

Аргумент addr должен указывать на структуру данных, содержащую адрес, приписанный socket'у программы-сервера, к которой делается запрос на соединение. Для сетей TCP/IP такой структурой является sockaddr\_in. Для формирования значений полей структуры sockaddr\_in удобно использовать функцию gethostbyname.

Аргумент addrlen задает размер (в байтах) структуры данных, указываемой аргументом addr.

Для того, чтобы запрос на соединение был успешным, необходимо, по крайней мере, чтобы программа-сервер выполнила к этому моменту системный вызов listen для socket'а с указанным адресом.

При успешном выполнении запроса системный вызов connect возвращает 0, в противном случае - "-1" (устанавливая код причины неуспеха в глобальной переменной errno).

Примечание. Если к моменту выполнения connect используемый им socket не был привязан к адресу посредством bind ,то такая привязка будет выполнена автоматически.

Примечание. В режиме взаимодействия без установления соединения необходимости в выполнении системного вызова connect нет. Однако, его выполнение в таком режиме не является ошибкой - просто меняется смысл выполняемых при этом действий: устанавливается адрес "по умолчанию" для всех последующих посылок дейтаграмм.

## *2.3. Прием запроса на установление связи*

Для приема запросов от программ-клиентов на установление связи в программах-серверах используется системный вызов accept, имеющий следующий вид:

#include <sys/socket.h>

#include <netinet/in.h>

int accept (s, addr, p\_addrlen)

int s;

struct sockaddr\_in \*addr;

int \*p\_addrlen;

Аргумент s задает дескриптор socket'а, через который программа-сервер получила запрос на соединение (посредством системного запроса listen ).

Аргумент addr должен указывать на область памяти, размер которой позволял бы разместить в ней структуру данных, содержащую адрес socket'а программы-клиента, сделавшей запрос на соединение. Никакой инициализации этой области не требуется.

Аргумент p\_addrlen должен указывать на область памяти в виде целого числа, задающего размер (в байтах) области памяти, указываемой аргументом addr.

Системный вызов accept извлекает из очереди, организованной системным вызовом listen, первый запрос на соединение и возвращает дескриптор нового (автоматически созданного) socket'а с теми же свойствами, что и socket, задаваемый аргументом s. Этот новый дескриптор необходимо использовать во всех последующих операциях обмена данными.

Кроме того после удачного завершения accept:

* область памяти, указываемая аргументом addr, будет содержать структуру данных (для сетей TCP/IP это sockaddr\_in), описывающую адрес socket'а программы-клиента, через который она сделала свой запрос на соединение;
* целое число, на которое указывает аргумент p\_addrlen, будет равно размеру этой структуры данных.

Если очередь запросов на момент выполнения accept пуста, то программа переходит в состояние ожидания поступления запросов от клиентов на неопределенное время (хотя такое поведение accept можно и изменить).

Признаком неудачного завершения accept служит отрицательное возвращенное значение (дескриптор socket'а отрицательным быть не может).

Примечание. Системный вызов accept используется в программах-серверах, функционирующих только в режиме с установлением соединения.

## *2.4. Формирование адреса узла сети*

Для получения адреса узла сети TCP/IP по его символическому имени используется библиотечная функция

#include <netinet/in.h>

#include <netdb.h>

struct hostent \*gethostbyname (name)

char \*name;

Аргумент name задает адрес последовательности литер, образующих символическое имя узла сети.

При успешном завершении функция возвращает указатель на структуру hostent, определенную в include-файле netdb.h и имеющую следующий вид struct hostent { char \*h\_name; char \*\*h\_aliases; int h\_addrtype; int h\_lenght; char \*h\_addr; };

Поле h\_name указывает на официальное (основное) имя узла.

Поле h\_aliases указывает на список дополнительных имен узла (синонимов), если они есть.

Поле h\_addrtype содержит идентификатор используемого набора протоколов, для сетей TCP/IP это поле будет иметь значение AF\_INET.

Поле h\_lenght содержит длину адреса узла.

Поле h\_addr указывает на область памяти, содержащую адрес узла в том виде, в котором его используют системные вызовы и функции socket-интерфейса.

Пример обращения к функции gethostbyname для получения адреса удаленного узла в программе-клиенте, использующей системный вызов connect для формирования запроса на установления соединения с программой-сервером на этом узле, рассматривается ниже.

## *3. Функции обмена данными*

В режиме с установлением логического соединения после удачного выполнения пары взаимосвязанных системных вызовов connect (в клиенте) и accept (в сервере) становится возможным обмен данными.

Этот обмен может быть реализован обычными системными вызовами read и write, используемыми для работы с файлами (при этом вместо дескрипторов файлов в них задаются дескрипторы socket'ов).

Кроме того могут быть дополнительно использованы системные вызовы send и recv, ориентированные специально на работу с socket'ами.

Примечание. Для обмена данными в режиме без установления логического соединения используются, как правило, системные вызовы sendtoи recvfrom. Sendto позволяет специфицировать вместе с передаваемыми данными (составляющими дейтаграмму) адрес их получателя. Recvfrom одновременно с доставкой данных получателю информирует его и об адресе отправителя.

## *3.1. Посылка данных*

Для посылки данных партнеру по сетевому взаимодействию используется системный вызов send, имеющий следующий вид

#include <sys/types.h>

#include <sys/socket.h>

int send (s, buf, len, flags)

int s;

char \*buf;

int len;

int flags;

Аргумент s задает дескриптор socket'а, через который посылаются данные.

Аргумент buf указывает на область памяти, содержащую передаваемые данные.

Аргумент len задает длину (в байтах) передаваемых данных.

Аргумент flags модифицирует исполнение системного вызова send. При нулевом значении этого аргумента вызов send полностью аналогичен системному вызову write.

При успешном завершении send возвращает количество переданных из области, указанной аргументом buf, байт данных. Если канал данных, определяемый дескриптором s, оказывается "переполненным", то send переводит программу в состояние ожидания до момента его освобождения.

## *3.2. Получение данных*

Для получения данных от партнера по сетевому взаимодействию используется системный вызов recv, имеющий следующий вид

#include <sys/types.h>

#include <sys/socket.h>

int recv (s, buf, len, flags)

int s;

char \*buf;

int len;

int flags;

Аргумент s задает дескриптор socket'а, через который принимаются данные.

Аргумент buf указывает на область памяти, предназначенную для размещения принимаемых данных.

Аргумент len задает длину (в байтах) этой области.

Аргумент flags модифицирует исполнение системного вызова recv. При нулевом значении этого аргумента вызов recv полностью аналогичен системному вызову read.

При успешном завершении recv возвращает количество принятых в область, указанную аргументом buf, байт данных. Если канал данных, определяемый дескриптором s, оказывается "пустым", то recv переводит программу в состояние ожидания до момента появления в нем данных.

## 4. Функции закрытия связи

Для закрытия связи с партнером по сетевому взаимодействию используются системные вызовы close и shutdown.

## *4.1. Системный вызов close*

Для закрытия ранее созданного socket'а используется обычный системный вызов close, применяемый в ОС UNIX для закрытия ранее открытых файлов и имеющий следующий вид

int close (s)

int s;

Аргумент s задает дескриптор ранее созданного socket'а.

Однако в режиме с установлением логического соединения (обеспечивающем, как правило, надежную доставку данных) внутрисистемные механизмы обмена будут пытаться передать/принять данные, оставшиеся в канале передачи на момент закрытия socket'а. На это может потребоваться значительный интервал времени, неприемлемый для некоторых приложений. В такой ситуации необходимо использовать описываемый далее системный вызов shutdown.

## *4.2. Сброс буферизованных данных*

Для "экстренного" закрытия связи с партнером (путем "сброса" еще не переданных данных) используется системный вызов shutdown, выполняемый перед close и имеющий следующий вид

int shutdown (s, how)

int s;

int how;

Аргумент s задает дескриптор ранее созданного socket'а.

Аргумент how задает действия, выполняемые при очистке системных буферов socket'а:

0 - сбросить и далее не принимать данные для чтения из socket'а;

1 - сбросить и далее не отправлять данные для посылки через socket;

2 - сбросить все данные, передаваемые через socket в любом направлении.

5. Пример использования socket-интерфейса

В данном разделе рассматривается использование socket-интерфейса в режиме взаимодействия с установлением логического соединения на очень простом примере взаимодействия двух программ (сервера и клиента), функционирующих на разных узлах сети TCP/IP.

Содержательная часть программ примитивна:

сервер, приняв запрос на соединение, передает клиенту вопрос "Who are you?";

клиент, получив вопрос, выводит его в стандартный вывод и направляет серверу ответ "I am your client" и завершает на этом свою работу;

сервер выводит в стандартный вывод ответ клиента, закрывает с ним связь и переходит в состояние ожидания следующего запроса к нему.

Примечание. Предлагаемые ниже тексты программ предназначены только для иллюстрации логики взаимодействия программ через сеть, поэтому в них отсутствуют такие атрибуты программ, предназначенных для практического применения, как обработка кодов возврата системных вызовов и функций, анализ кодов ошибок в глобальной переменной errno, реакция на асинхронные события и т.п.

## *5.1. Программа-сервер*

Текст программы-сервера на языке программирования СИ выглядит следующим образом

1 #include <sys/types.h>

2 #include <sys/socket.h>

3 #include <netinet/in.h>

4 #include <netdb.h>

5 #include <memory.h>

6 #define SRV\_PORT 1234

7 #define BUF\_SIZE 64

8 #define TXT\_QUEST "Who are you?\n"

9 main () {

10 int s, s\_new;

11 int from\_len;

12 char buf[BUF\_SIZE];

13 struct sockaddr\_in sin, from\_sin;

14 s = socket (AF\_INET, SOCK\_STREAM, 0);

15 memset ((char \*)&sin, '\0', sizeof(sin));

16 sin.sin\_family = AF\_INET;

17 sin.sin\_addr.s\_addr = INADDR\_ANY;

18 sin.sin\_port = SRV\_PORT;

19 bind (s, (struct sockaddr \*)&sin, sizeof(sin));

20 listen (s, 3);

21 while (1) {

22 from\_len = sizeof(from\_sin);

23 s\_new = accept (s, &from\_sin, &from\_len);

24 write (s\_new, TXT\_QUEST, sizeof(TXT\_QUEST));

25 from\_len = read (s\_new, buf, BUF\_SIZE);

26 write (1, buf, from\_len);

27 shutdown (s\_new, 0);

28 close (s\_new);

29 };

30 }

Строки 1...5 описывают включаемые файлы, содержащие определения для всех необходимых структур данных и символических констант.

Строка 6 приписывает целочисленной константе 1234 символическое имя SRV\_PORT. В дальнейшем эта константа будет использована в качестве номера порта сервера. Значение этой константы должно быть известно и программе-клиенту.

Строка 7 приписывает целочисленной константе 64 символическое имя BUF\_SIZE. Эта константа будет определять размер буфера, используемого для размещения принимаемых от клиента данных.

Строка 8 приписывает последовательности символов, составляющих текст вопроса клиенту, символическое имя TXT\_QUEST. Последним символом в последовательности является символ перехода на новую строку '\n'. Сделано это для упрощения вывода текста вопроса на стороне клиента.

В строке 14 создается (открывается) socket для организации режима взаимодействия с установлением логического соединения (SOCK\_STREAM) в сети TCP/IP (AF\_INET), при выборе протокола транспортного уровня используется протокол "по умолчанию" (0).

В строках 15...18 сначала обнуляется структура данных sin, а затем заполняются ее отдельные поля. Использование константы INADDR\_ANY упрощает текст программы, избавляя от необходимости использовать функцию gethostbyname для получения адреса локального узла, на котором запускается сервер.

Строка 19 посредством системного вызова bind привязывает socket, задаваемый дескриптором s, к порту с номером SRV\_PORT на локальном узле. Bind завершится успешно при условии, что в момент его выполнения на том же узле уже не функционирует программа, использующая этот номер порта.

Строка 20 посредством системного вызова listen организует очередь на три входящих к серверу запроса на соединение.

Строка 21 служит заголовком бесконечного цикла обслуживания запросов от клиентов.

На строке 23, содержащей системный вызов accept, выполнение программы приостанавливается на неопределенное время, если очередь запросов к серверу на установление связи оказывается пуста. При появлении такого запроса accept успешно завершается, возвращая в переменной s\_new дескриптор socket'а для обмена информацией с клиентом.

В строке 24 сервер с помощью системного вызова write отправляет клиенту вопрос.

В строке 25 с помощью системного вызова read читается ответ клиента.

В строке 26 ответ направляется в стандартный вывод, имеющий дескриптор файла номер 1. Так как строка ответа содержит в себе символ перехода на новую строку, то текст ответа будет размещен на отдельной строке дисплея.

Строка 27 содержит системный вывод shutdown, обеспечивающий очистку системных буферов socket'а, содержащих данные для чтения ("лишние" данные могут там оказаться в результате неверной работы клиента).

В строке 28 закрывается (удаляется) socket, использованный для обмена данными с очередным клиентом.

Примечание. Данная программа (как и большинство реальных программ-серверов) самостоятельно своей работы не завершает, находясь в бесконечном цикле обработки запросов клиентов. Ее выполнение может быть прервано только извне путем посылки ей сигналов (прерываний) завершения. Правильно разработанная программа-сервер должна обрабатывать такие сигналы, корректно завершая работу (закрывая, в частности, посредством close socket с дескриптором s).

## *5.2. Программа-клиент*

Текст программы-клиента на языке программирования СИ выглядит следующим образом

1 #include <sys/types.h>

2 #include <sys/socket.h>

3 #include <netinet/in.h>

4 #include <netdb.h>

5 #include <memory.h>

6 #define SRV\_HOST "delta"

7 #define SRV\_PORT 1234

8 #define CLNT\_PORT 1235

9 #define BUF\_SIZE 64

10 #define TXT\_ANSW "I am your client\n"

11 main () {

12 int s;

13 int from\_len;

14 char buf[BUF\_SIZE];

15 struct hostent \*hp;

16 struct sockaddr\_in clnt\_sin, srv\_sin;

17 s = socket (AF\_INET, SOCK\_STREAM, 0);

18 memset ((char \*)&clnt\_sin, '\0',

sizeof(clnt\_sin));

19 clnt\_sin.sin\_family = AF\_INET;

20 clnt\_sin.sin\_addr.s\_addr = INADDR\_ANY;

21 clnt\_sin.sin\_port = CLNT\_PORT;

22 bind (s, (struct sockaddr \*)&clnt\_sin,

sizeof(clnt\_sin));

23 memset ((char \*)&srv\_sin, '\0',

sizeof(srv\_sin));

24 hp = gethostbyname (SRV\_HOST);

25 srv\_sin.sin\_family = AF\_INET;

26 memcpy ((char

\*)&srv\_sin.sin\_addr,hp->h\_addr,hp->h\_length);

27 srv\_sin.sin\_port = SRV\_PORT;

28 connect (s, &srv\_sin, sizeof(srv\_sin));

29 from\_len = recv (s, buf, BUF\_SIZE, 0);

30 write (1, buf, from\_len);

31 send (s, TXT\_ANSW, sizeof(TXT\_ANSW), 0);

32 close (s);

33 exit (0);

34 }

В строках 6 и 7 описываются константы SRV\_HOST и SRV\_PORT, определяющие имя удаленного узла, на котором функционирует программа-сервер, и номер порта, к которому привязан socket сервера.

Строка 8 приписывает целочисленной константе 1235 символическое имя CLNT\_PORT. В дальнейшем эта константа будет использована в качестве номера порта клиента.

В строках 17...22 создается привязанный к порту на локальном узле socket.

В строке 24 посредством библиотечной функции gethostbyname транслируется символическое имя удаленного узла (в данном случае "delta"), на котором должен функционировать сервер, в адрес этого узла, размещенный в структуре типа hostent.

В строке 26 адрес удаленного узла копируется из структуры типа hostent в соответствующее поле структуры srv\_sin, которая позже (в строке 28) используется в системном вызове connect для идентификации программы-сервера.

В строках 29...31 осуществляется обмен данными с сервером и вывод вопроса, поступившего от сервера, в стандартный вывод.

Строка 32 посредством системного вызова close закрывает (удаляет) socket.