**Лекция 24. Сетевое программирование**

1. Сокеты Windows

2. Серверные функции

3. Клиентские функции

*Сокеты* - название программного интерфейса, который обеспечивает обмен данными между процессами. Процессы, участвующие в обмене данными, могут исполняться как на локальном компьютере, так и на разных компьютерах, объединенных в компьютерную сеть. *Сокет* - абстрактный объект, представляющий конечную точку соединения.  
Выделяют клиентские и слушающие (серверные) сокеты. Различия между ними очевидны: клиентские подключаются к процессу (к слушающему сокету), слушающие сокеты, соответственно, обрабатывают эти подключения. Передача данных между процессами происходит через клиентские сокеты.

**Общая схема работы с сокетами в Windows**

**Клиент**

* Инициализация WSA
* Создание сокета
* Присоединение к серверу
* Прием/передача данных
* Разрыв соединения

**Сервер**

* Инициализация WSA
* Создание слушающего сокета и привязка к порту
* Прослушивание порта
* Обработка входящих подключений
* Прием/передача данных
* Разрыв соединения

В последующих статьях представлены описания необходимых методов и исходный код реализации этих схем.

**Шаблон программы**

Этот код содержит основные включения WSA.

#include < winsock2.h >

#include < ws2tcpip.h >

#include < iostream >

#pragma comment(lib, "Ws2\_32.lib")

int main()

{

return 0;

}

**1. Сокеты Windows.**

Возможность взаимодействия с другими системами обеспечивается в Windows поддержкой сокетов (sockets) Windows Sockets.

Winsock API разрабатывался как расширение Berkley Sockets API для среды Windows и поэтому поддерживается всеми системами Windows. К преимуществам Winsock можно отнести следующее:

• Перенос уже имеющегося кода, написанного для Berkeley Sockets API, осуществляется непосредственно.

• Системы Windows легко встраиваются в сети. IPv6 допускает использование более длинных IP-адресов, преодолевая существующий 4-байтовый адресный барьер версии IPv4.

• Сокеты могут использоваться совместно с перекрывающимся вводом/выводом Windows, что, помимо всего прочего, обеспечивает возможность масштабирования серверов при увеличении количества активных клиентов.

• Сокеты можно рассматривать как дескрипторы (типа HANDLE) файлов при использовании функций ReadFile и WriteFile и, с некоторыми ограничениями, при использовании других функций, точно так же, как в качестве дескрипторов файлов сокеты применяются в UNIX. Эта возможность оказывается удобной в тех случаях, когда требуется использование асинхронного ввода/вывода и портов завершения ввода/вывода.

• Существуют также дополнительные, непереносимые расширения.

Winsock API поддерживается библиотекой DLL (WS2\_32.DLL), для получения доступа к которой следует подключить к программе библиотеку WS\_232 .LIB. Эту DLL следует инициализировать с помощью нестандартной, специфической для Winsock функции WSAStartup, которая должна быть первой из функций Winsock, вызываемых программой. Когда необходимость в использовании функциональных возможностей Winsock отпадает, следует вызывать функцию WSACleanup.

int WSAStartup (

WORD

wVersionRequired,

LPWSADATA

lpWSAData);

wVersionRequired — указывает старший номер версии библиотеки DLL, который вам требуется и который вы можете использовать.

Функция возвращает ненулевое значение, если запрошенная версия данной DLL не поддерживается.

Младший байт параметра wVersionRequired указывает основной номер версии, а старший байт — дополнительный. Обычно используют макрос MAKEWORD; таким образом, выражение MAKEWORD (2,0) представляет версию 2.0.

lpWSAData — указатель на структуру WSADATA, которая возвращает информацию о конфигурации DLL, включая старший доступный номер версии.

По окончании работы программы, а также в тех случаях, когда необходимости в использовании сокетов больше нет, следует вызывать функцию WSACleanup, чтобы библиотека WS\_32. DLL, обслуживающая сокеты, могла освободить ресурсы, распределенные для этого процесса.

Инициализировав Winsock DLL, можно использовать стандартные (Berkeley Sockets) функции для создания сокетов и соединений, обеспечивающих взаимодействие серверов с клиентами или взаимодействие равноправных узлов сети между собой.

Используемый в Winsock тип данных SOCKET аналогичен типу данных HANDLE в Windows, и его даже можно применять совместно с функцией ReadFile и другими функциями Windows, требующими использования дескрипторов типа HANDLE. Для создания (или открытия) сокета служит функция socket.

SOCKET socket (int af, int type, int protocol);

af — обозначает семейство адресов, или протокол; для указания протокола IP.

type — указывает тип взаимодействия: ориентированное на установку соединения (connection-oriented communication), или потоковое (SOCK\_STREAM), и дейтаграммное (datagram communication) (SOCK\_DGRAM),

protocol — является излишним, если параметр af установлен равным AF\_INET; используйте значение 0.

В случае неудачного завершения функция socket возвращает значение INVALID\_SOCKET.

Winsock можно использовать совместно с протоколами, отличными от TCP/IP.

**2. Серверные функции сокета.**

Под *сервером* будет пониматься процесс, который принимает запросы на образование соединения через заданный порт. Несмотря на то что сокеты, подобно именованным каналам, могут использоваться для создания соединений между равноправными узлами сети, введение указанного различия между узлами является весьма удобным и отражает различия в способах, используемых обеими системами для соединения друг с другом.

Следующий шаг заключается в привязке сокета к его адресу и *конечной точке* (endpoint) (направление канала связи от приложения к службе). Вызов socket, за которым следует вызов bind, аналогичен созданию именованного канала. Однако не существует имен, используя которые можно было бы различать сокеты данного компьютера. Вместо этого в качестве конечной точки службы используется *номер порта* (port number). Любой заданный сервер может иметь несколько конечных точек.

int bind (

SOCKET

s,

const struct sockaddr \*saddr,

int namelen);

s — несвязанный сокет, возвращенный функцией socket.

saddr — заполняется перед вызовом и задает протокол и специфическую для протокола информацию.

namelen — присвойте значение sizeof (sockaddr).

В случае успешного выполнения функция возвращает значение 0, иначе — SOCKET\_ERROR. Структура sockaddr определяется следующим образом:

struct sockaddr {

u\_short sa\_family;

char sa\_data [14];

}; typedef struct sockaddr SOCKADDR,

\*PSOCKADDR;

Первый член этой структуры, safamily, обозначает протокол. Второй член, sa\_data, зависит от протокола. Internet-версией структуры sa\_data является структура sockaddr\_in:

struct sockaddr\_in {

short sin\_family; /\* AF\_INET \*/

u\_short sin\_port;

struct in\_addr sin\_addr; /\* 4-байтовый IP-адрес \*/

char sin\_zero [8];

};

typedef struct sockaddr\_in

SOCKADDR\_IN, \*PSOCKADDR\_IN;

Для преобразования текстовой строки с IP-адресом к требуемому формату можно использовать функцию inet\_addr, поэтому член sin\_addr.s\_addr переменной sockaddr\_in инициализируется следующим образом:

sa.sin\_addr.s\_addr = inet\_addr ("192.13.12.1");

О связанном сокете, для которого определены протокол, номер порта и IP-адрес, иногда говорят как об *именованном сокете* (named socket).

***Перевод связанного сокета в состояние прослушивания***

Функция listen делает сервер доступным для образования соединения с клиентом. Аналогичной функции для именованных каналов не существует.

int listen (SOCKET s, int nQueueSize);

Параметр nQueueSize указывает число запросов на соединение, которые вы намерены помещать в очередь сокета.

***Прием клиентских запросов соединения***

Сервер может ожидать соединения с клиентом, используя функцию accept, возвращающую новый подключенный сокет, который будет использоваться в операциях ввода/вывода.

SOCKET accept (

SOCKET s,

LPSOCKADDR

lpAddr, LPINT

lpAddrLen) ;

s — прослушивающий сокет.

lpAddr — указатель на структуру sockaddr\_in, предоставляющую адрес клиентской системы.

lpAddrLen — указатель на переменную, которая будет содержать размер возвращенной структуры sockaddr\_\_in.

***Отключение и закрытие сокетов***

ДЛЯ отключения сокетов применяется функция shutdown (s, how).

Аргумент how может принимать одно из двух значений: 1, указывающее на то, что соединение может быть разорвано только для посылки сообщений, и 2, соответствующее разрыву соединения как для посылки, так и для приема сообщений. Функция shutdown не освобождает ресурсы, связанные с сокетом, но гарантирует завершение посылки и приема всех данных до закрытия сокета. Тем не менее, после вызова функции shutdown приложение уже не должно использовать этот сокет.

Когда работа с сокетом закончена, его следует закрыть, вызвав функцию closesocket (SOCKET s). Сначала сервер закрывает сокет, созданный функцией accept, а не прослушивающий сокет, созданный с помощью функции socket. Сервер должен закрывать прослушивающий сокет только тогда, когда завершает работу или прекращает принимать

клиентские запросы соединения.

**3. Клиентские функции сокета.**

Клиентская станция, которая желает установить соединение с сервером, также должна создать сокет, вызвав функцию socket.

Следующий шаг заключается в установке соединения сервером, а, кроме того, необходимо указать номер порта, адрес хоста и другую информацию. Имеется только одна дополнительная функция-connect.

***Установление клиентского соединения с сервером***

Клиент может соединиться с сервером ним при помощи функции

connect.

int connect (

SOCKET

s,

LPSOCKADDR

lpName, int

nNameLen);

s — сокет, созданный с использованием функции socket.

lpName — указатель на структуру sockaddr\_in, инициализированную значениями номера порта и IP-адреса системы с сокетом, связанным с указанным портом, который находится в состоянии прослушивания.

Инициализируйте nNameLen значением sizeof (struct sockaddr\_in).

***Отправка и получение данных***

Программы, использующие сокеты, обмениваются данными с помощью функций send и recv, прототипы которых почти совпадают (перед указателем буфера функции send помещается модификатор const).

Прототип функции send.

int send (

SOCKE

T s,

const char \* lpBuffer,

int nBufferLen, int

nFlags);

Возвращаемым значением является число фактически переданных байтов. Значение SOCKET\_ERROR указывает на ошибку.

nFlags — может использоваться для обозначения степени срочности сообщений (например, экстренных сообщений), а значение MSG\_PEEK позволяет просматривать получаемые данные без их считывания.

Функции send и recv *не являются атомарными* (atomic), и поэтому нет никакой гарантии, что затребованные данные будут действительно отправлены или получены.

С сокетами могут использоваться также функции ReadFile и WriteFile, только в этом случае при вызове функции необходимо привести сокет к типу HANDLE.

**4. Сравнение именованных каналов и сокетов.**

Именованные каналы, очень похожи на сокеты, но в способах их использования имеются значительные различия.

• Именованные каналы могут быть ориентированными на работу с сообщениями, что значительно упрощает программы.

• Именованные каналы требуют использования функций ReadFile и Write-File, в то время как сокеты могут обращаться также к функциям send и recv.

• В отличие от именованных каналов сокеты настолько гибки, что предоставляют пользователям возможность выбрать протокол для использования с сокетом, например, TCP или UDP. Кроме того, пользователь имеет возможность выбирать протокол на основании характера предоставляемой услуги или иных факторов.

• Сокеты основаны на промышленном стандарте, что обеспечивает их совместимость с системами, отличными от Windows.

Имеются также различия в моделях программирования сервера и клиента.

***Сравнение серверов именованных каналов и сокетов***

Установка соединения с несколькими клиентами при использовании сокетов требует выполнения повторных вызовов функции accept. Каждый из вызовов возвращает очередной подключенный сокет. По сравнению с именованными каналами имеются следующие отличия:

• В случае именованных каналов требуется, чтобы каждый экземпляр именованного канала и дескриптор типа HANDLE создавались с помощью функции CreateNamedPipe, тогда как для создания экземпляров сокетов применяется функция accept.

• Допустимое количество клиентских сокетов ничем не ограничено (функция listen ограничивает лишь количество клиентов, помещаемых в очередь), в то время как количество экземпляров именованных каналов, в зависимости от того, что было указано

при первом вызове функции CreateNamedPipe, может быть ограниченным,

• Не существует вспомогательных функций для работы с сокетами, аналогичных функции TransactNamedPipe.

• Именованные каналы не имеют портов с явно заданными номерами и различаются по именам.

В случае сервера именованных каналов получение пригодного для работы дескриптора типа HANDLE требует вызова двух функций (CreateNaraedPipe и ConnectNamedPipe), тогда как сервер сокета требует вызова четырех функций (socket,bind,listen и accept).

***Сравнение клиентов именованных каналов и сокетов***

В случае именованных каналов необходимо последовательно вызывать функции WaitNamedPipe и CreateFile. Если же используются сокеты, этот порядок вызовов обращается, поскольку можно считать, что функция socket создает со-кет, а функция connect — блокирует.

Дополнительное отличие состоит в том, что функция connect является функцией клиента сокета, в то время как функция ConnectNamedPipe используется сервером именованного канала.

**5. Дейтаграммы.**

Дейтаграммы аналогичны почтовым ящикам и используются при сходных обстоятельствах. Соединение между отправителем и получателем отсутствует, а получателей может быть несколько. Ни почтовые ящики, ни дейтаграммы не гарантируют доставку данных получателю, а последовательные сообщения не обязательно будут получены в той же очередности, в которой они были отправлены.

Первым шагом при использовании дейтаграмм является создание сокета посредством вызова функции socket с указанием значения SOCK DGRAM в поле type.

Далее необходимо использовать функции sendto и recvf rom, которые принимают те же аргументы, что и функции send и recv, но имеют по два дополнительных аргумента, относящихся к станции-партнеру. Так, функция sendto имеет следующий прототип:

int sendto ( SOCKET

s, LPSTR

lpBuffer, int

nBufferLen, int

nFlags,

LPSOCKADDR

lpAddr, int

nAddrLen);

lpAddr — указывает на адресную структуру, в которой можно задать имя конкретной системы и номер порта или же указать на необходимость рассылки дейтаграммы заданной совокупности систем.

Используя функцию reef rom, указывают систему или системы (возможно, все), от которых нужно принимать дейтаграммы.

Обычно дейтаграммы применяются для реализации RPC. В самых распространенных ситуациях клиент посылает запрос серверу, используя дейтаграммы. Поскольку доставка запроса не гарантируется, клиент должен повторно передать запрос, если по истечении заданного периода ожидания ответ от сервера (для посылки которого также используются дейтаграммы) не получен. Сервер должен быть готов к тому, что один и тот же запрос может направляться ему несколько раз.

Важно отметить, что ни клиенту, ни серверу RPC служебные сигналы, которые, например, необходимы при образовании соединения через потоковый со-кет, не требуются; вместо этого они связываются друг с другом посредством запросов и ответов. В качестве дополнительной возможности RPC может гарантировать надежность взаимодействия путем повторной передачи запросов по истечении периода ожидания, что упрощает разработку приложений. Выражаясь иначе, часто говорят о том, что клиент и сервер RPC *не имеют состояния* (они не хранят никакой информации относительно состояния текущего запроса или запросов, на которые еще не получен ответ). Отсюда следует, что результат обработки на сервере множества идентичных клиентских запросов будет тем же, что и результат обработки одиночного запроса. Это также значительно упрощает проектирование приложений и реализацию их логики.