СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

ЛЕКЦИЯ № 10

ПРЕПОДАВАТЕЛЬ: ХУСТОЧКА А.В.

СОКЕТЫ

- Технология сокетов (sockets) лежит в основе современного сетевого программирования. На ней базируются в настоящее время основные операционные среды (Unix, Windows).
- Эта технология была разработана в университете г. Беркли (США) для системы Unix, поэтому сокеты иногда называют сокетами Беркли. На идеологии сокетов реализуется механизм взаимодействия не только партнеров по телекоммуникациям, но и процессов в ЭВМ вообще.
- В стандарте семиуровневой модели OSI сокеты лежат на транспортном уровне. Сокет является пограничным понятием между протоколами телекоммуникаций и операционной системой ЭВМ.
- Интерфейс WinSock API предоставляет средства организации передачи данных с использованием дейтаграмм и каналов связи между узлами сети. Он позволяет передавать данные не только с использованием протокола TCP/IP, но и других протоколов, например, IPX/SPX.
- Как известно, в локальных и глобальных сетях существует два принципиально разных способа передачи данных: без установления соединения (дейтаграммный) и с установлением соединения (канала связи).

СОКЕТЫ

- Первый из них (без установления соединения) предполагает посылку пакетов данных от одного узла другому (или сразу нескольким узлам) без получения подтверждения о доставке и даже без гарантии того, что передаваемые пакеты будут получены в правильной последовательности.
- Примером такого протокола может служить протокол UDP, который используется в сетях TCP/IP, или протокол IPX, который является базовым в сетях Novell NetWare.
- Основные преимущества дейтаграммных протоколов заключаются в высоком быстродействии и возможности широковещательной передачи данных, когда один узел отправляет сообщения, а другие их получают, причем все одновременно.

СОКЕТЫ

- Второй способ передачи данных предполагает создание канала передачи данных между двумя различными узлами сети.
- При этом канал создается средствами дейтаграммных протоколов, однако доставка пакетов в канале является гарантированной. Пакеты всегда доходят в правильном порядке, хотя быстродействие получается в среднем ниже за счет посылки подтверждений. Примерами протоколов, использующих каналы связи, могут служить протоколы TCP и SPX. Протокол NETBIOS допускает передачу данных с использованием как дейтаграмм, так и каналов связи.

СОКЕТЫ

Для сокета необходимо указать три параметра:

- IP адрес, связанный с сокетом
- номер порта, для которого будут выполняться операции передачи данных
- тип сокета.

Что касается последнего параметра (тип сокета), то существуют сокеты двух типов.

Первый тип предназначен для передачи данных в виде дейтаграмм, второй - с использованием каналов связи.

МЕХАНИЗМ ПРИМЕНЕНИЯ СОКЕТОВ

- Все сетевые приложения построены на технологии клиент-сервер; это значит, что в сети существует по крайней мере одно приложение, являющее сервером, типичная задача которого - это ожидание запроса на подключение от приложений-клиентов, которых может быть теоретически сколько угодно, и выполнение всевозможных процедур в ответ на запросы клиентов.
- Для клиент-серверной технологии абсолютно неважно, где расположены клиент и сервер - на одной машине или на разных. Конечно, для успешного соединения клиента с сервером клиенту необходимо иметь минимальный набор данных о расположении сервера - для сетей TCP/IP это IP-адрес компьютера, где расположен сервер, и адрес порта, на котором сервер ожидает запросы от клиентов.

МЕХАНИЗМ ПРИМЕНЕНИЯ СОКЕТОВ

- Каждый из компьютеров в сети TCP/IP имеет свой уникальный IP адрес, который используется для обмена данными с другими компьютерами. Каждый посылаемый пакет от одного компьютера другому имеет адрес отправителя и получателя, что позволяет его однозначно идентифицировать.
- Однако в случае, если на компьютере работает множество приложений, одновременно использующих сеть, такого набора атрибутов явно недостаточно.
- Для разрешения неоднозначности, кроме адреса, каждое соединение (то есть каждый процесс) на каждом конце имеет идентификатор под названием "порт". Этот идентификатор представляет число от 0 до 65535. Таким образом, пара адрес+порт определяет сокет-канал, по которому два компьютера обмениваются данными друг с другом.
- Только одно приложение на одном компьютере в одно и то же время может использовать конкретный порт, однако для серверных частей возможно создание нескольких сокетов на одном порту для работы с несколькими клиентами.

МЕХАНИЗМ ПРИМЕНЕНИЯ СОКЕТОВ

- Для пользовательских программ обычно используются номера портов в диапазоне 1025 – 5000.
- Порты с меньшими номерами зарезервированы для таких известных служб, как telnet или ftp, а с большими предполагаются для использования другими стандартными службами.
- Значение порта не обязательно должно совпадать на сервере и клиенте - клиенту для соединения важно только знать порт сервера, порт клиента может выбираться клиентом произвольно и становится известен серверу в момент запроса клиента на соединение.
- Когда соединение будет установлено, ОС создаст для серверного приложения соответствующий сокет, с которым и будет работать приложение, так что порт клиента для сервера совершенно не важен.

МЕХАНИЗМ ПРИМЕНЕНИЯ СОКЕТОВ

- Механизм работы сокетов таков: на серверной стороне запускается серверный сокет, который после запуска сразу переходит в режим прослушивания (т.е. ожидания соединения клиентов).
- На стороне клиента создается сокет, для которого указывается IP-адрес и порт сервера и дается команда на соединение.
- Когда сервер получает запрос на соединение, ОС создает новый экземпляр сокета, с помощью которого сервер может обмениваться данными с клиентом. При этом сокет, который создан для прослушивания, продолжает находиться в режиме приема соединений; таким образом, программист может создать сервер, работающий с несколькими подключениями от клиентов.
- Работа с сокетами, по существу, это операции ввода-вывода, которые бывают синхронные и асинхронные. В терминологии сокетов работа в асинхронном режиме называется блокирующими сокетами, а в синхронном - неблокирующие сокеты. Попытка соединения или приема данных в блокирующем режиме (отправка всегда синхронна, так как фактически является постановкой в очередь) означает, что пока программа не соединится или не примет данные, передачи управления на следующий оператор не произойдет.

МИНИМАЛЬНЫЙ НАБОР ФУНКЦИЙ WINAPI

Рассмотрим минимальный набор функций из WinSock API, необходимых для написания элементарного клиента и сервера. Сами функции находятся в файле winsock32.dll.

Функция	Описание
WSAStartup	Функция сообщает ОС, что в любом процессе приложения могут быть использованы функции WinSock. Функция должна быть вызвана один раз при запуске приложения перед использованием любой функции WinSock.
WSACleanup	Функция сообщает ОС, что приложение более не использует WinSock. Должна быть вызвана перед завершением приложения.
Socket	Функция создает сокет. Входящие параметры: af - спецификация семейства сокетов в (AF_INET, AF_IPX и др.), Struct - спецификация типа нового сокета (принимает значение SOCK_STREAM или SOCK_DGRAM), protocol - специфический протокол, который будет использоваться сокетом (число). Если функция выполнена без ошибок, она возвращает дескриптор на новый сокет, если ошибки есть, возвращается INVALID_SOCKET.

МИНИМАЛЬНЫЙ НАБОР ФУНКЦИЙ WINAPI

Функция	Описание
Bind	Функция ассоциирует адрес с сокетом. Структура адреса содержит порт (необходимо привести функцией htons) и адрес (для сервера обычно указывается INADDR_ANY - любой).
Connect	Функция соединения для клиента. Структура адреса содержит порт (необходимо привести функцией htons) и адрес (для клиента необходимо привести из имени или спецификации ip4 - xxx.xxx.xxx.xxx).
Send	Функция отправки данных. Помещает в очередь сокета s кусок данных из buf, длиной len. Последний параметр отвечает за вид передачи сообщения. Может быть проигнорирован (0).
Recv	Функция получения данных.

ЭТАПЫ РАБОТЫ С СОКЕТАМИ

Итак, работа с сокетами содержит ряд этапов:

- сокет создается
- настраивается на заданный режим работы
- применяется для организации обмена
- и, наконец, ликвидируется.
- Для этого необходимо: - инициализировать приложение для работы с интерфейсом Windows Sockets; - создать сокет и убедиться, что он возвращает ненулевую величину, которая является дескриптором сокета; - заполнить структуру SOCKADDR_IN необходимыми данными, включающими формат адреса, порт и IP-адрес; - использовать bind() для привязки к определенному IP – адресу (если использовать inet_addr("0.0.0.0") или htonl(INADDR_ANY) в секции sin_addr структуры SOCKADDR_IN, можно привязать сокет к любому адресу).
- Затем можно предпринимать действия по созданию связи процессов с помощью сокета

ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

- Прежде всего в процессе инициализации приложение должно зарегистрировать себя в библиотеке WSOCK32.DLL, которая предоставляет приложениям интерфейс WinSock API в среде операционных систем Microsoft Windows. Для этого следует подключить к программе библиотеку Ws2_32.lib:

```
#include <winsock2.h>

#pragma comment(lib,"Ws2_32.lib")
```

- Эту DLL следует инициализировать с помощью нестандартной, специфической для WinSock функции WSAStartup, которая должна быть первой из функций WinSock, вызываемых программой. WSAStartup определяется следующим образом:

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

- В параметре wVersionRequested необходимо указать версию интерфейса Windows Sockets, необходимую для работы приложения. Старший байт параметра указывает младший номер версии (minor version), младший байт - старший номер версии (major version). Перед вызовом функции WSAStartup параметр lpWSAData должен содержать указатель на структуру типа WSADATA , в которую будут записаны сведения о конкретной реализации интерфейса Windows Sockets.
- В случае успеха функция WSAStartup возвращает нулевое значение

ИНИЦИАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

- Если происходит ошибка, возвращается одно из следующих значений

Ошибка	Описание
WSASYSNOTREADY	Сетевое программное обеспечение не готово для работы
WSAVERNOTSUPPORTED	Функция не поддерживается данной реализацией интерфейса Windows Sockets
WSAEINVAL	Библиотека DLL, обеспечивающая интерфейсе Windows Sockets, не соответствует версии, указанной в параметре wVersionRequested

ПРИМЕР ИНИЦИАЛИЗАЦИИ

- В операционных системах Microsoft Windows версий до Windows NT включительно использовалась система Windows Sockets версии 1.1. В настоящее время следует указывать версию 2.2:
- WSAData wsd;
- WSAStartup(0x0202,&wsd);
- Определение структуры WSADATA и указателя на нее выглядят следующим образом:

```
typedef struct WSAData {
        WORD                    wVersion;
        WORD                    wHighVersion;
        unsigned short          iMaxSockets;
        unsigned short          iMaxUdpDg;
        char FAR *              lpVendorInfo;
        char                    szDescription[WSADESCRIPTION_LEN+1];
        char                    szSystemStatus[WSASYS_STATUS_LEN+1];
} WSADATA;
```

ПРИМЕР ИНИЦИАЛИЗАЦИИ

- Использованные выше поля szDescription и szSystemStatus после вызова функции WSAStartup содержат, соответственно, описание конкретной реализации интерфейса Windows Socket и текущее состояние этого интерфейса в виде текстовых строк.
- В полях wVersion и wHighVersion записаны, соответственно, версия спецификации Windows Socket, которую будет использовать приложение, и версия спецификации, которой соответствует конкретная реализация интерфейса Windows Socket.
- Приложение может одновременно создавать несколько сокетов, например, для использования в разных подзадачах одного процесса. В поле iMaxSockets хранится максимальное количество сокетов, которое можно получить для одного процесса.
- В поле iMaxUdpDg записан максимальный размер пакета данных, который можно переслать с использованием дейтаграммного протокола UDP.
- И, наконец, поле lpVendorInfo содержит указатель на дополнительную информацию, формат которой зависит от фирмы-изготовителя конкретной реализации системы Windows Sockets.

ЗАВЕРШЕНИЯ РАБОТЫ ПРИЛОЖЕНИЯ

- Перед тем, как завершить свою работу, приложение должно освободить ресурсы, полученные у операционной системы для работы с Windows Sockets. Для выполнения этой задачи приложение должно вызвать функцию WSACleanup , определенную так, как это показано ниже:

```
int WSACleanup(void);
```

- Эта функция может возвратить нулевое значение при успехе или значение SOCKET_ERROR в случае ошибки. Для получения кода ошибки необходимо воспользоваться функцией с именем WSAGetLastError:

```
int WSAGetLastError(void);
```

ЗАВЕРШЕНИЯ РАБОТЫ ПРИЛОЖЕНИЯ

- Функция WSAGetLastError позволяет определить код ошибки при неудачном завершении практически всех функций интерфейса Windows Sockets.
- Если ошибка возникла при выполнении функции WSACleanup, функция WSAGetLastError может вернуть одно из следующих значений:

Ошибка	Описание
WSANOTINITIALISED	Интерфейс Windows Sockets не был проинициализирован функцией WSAStartup
WSAENETDOWN	Сбой сетевого программного обеспечения
WSAEINPROGRESS	Во время вызова функции WSACleanup выполнялась одна из блокирующих функций интерфейса Windows Sockets

- Сделаем небольшие пояснения относительно последней ошибки, приведенной в этом списке, имеющей код WSAEINPROGRESS . Некоторые функции интерфейса Windows Sockets способны блокировать работу приложения, так как они не возвращают управление до своего завершения.
- В операционных системах, использующих вытесняющую многозадачность, к которым относятся ОС Microsoft Windows, это не приводит к блокировке всей системы. Можно избежать использования блокирующих функций, так как для них в интерфейсе Windows Sockets существует замена.

СОЗДАНИЕ И ИНИЦИАЛИЗАЦИЯ СОКЕТА

- После инициализации интерфейса Windows Sockets приложение должно создать один или несколько сокетов, которые будут использованы для передачи данных.
- Сокет создается с помощью функции socket , имеющей следующий прототип:

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

- Параметр af определяет формат адреса. Для этого параметра вы должны указывать значение AF_INET , что соответствует формату адреса, принятому в Internet.
- Параметры type и protocol определяют, соответственно, тип сокета и протокол, который будет использован для данного сокета. Можно указывать сокеты следующих двух типов

Тип	Описание
SOCK_STREAM	Сокет будет использован для передачи данных через канал связи с использованием протокола TCP
SOCK_DGRAM	Передача данных будет выполняться без создания каналов связи через дейтаграммный протокол UDP

- Что же касается параметра protocol, то можно указать для него нулевое значение.

СОЗДАНИЕ И ИНИЦИАЛИЗАЦИЯ СОКЕТА

- В случае успеха функция socket возвращает дескриптор, который нужно использовать для выполнения всех операций над данным сокетом.
- Если же произошла ошибка, эта функция возвращает значение INVALID_SOCKET .
- Для анализа причины ошибки вы должны вызвать функцию WSAGetLastError, которая в данном случае может вернуть один из следующих кодов ошибки:

Ошибка	Описание
WSANOTINITIALISED	Интерфейс Windows Sockets не был проинициализирован функцией WSAStartup
WSAENETDOWN	Сбой сетевого программного обеспечения
WSAEAFNOSUPPORT	Указан неправильный тип адреса
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса Windows Sockets
WSAEMFILE	Израсходован весь запас свободных дескрипторов
WSAENOBUFS	Нет памяти для создания буфера
WSAEPROTONOSUPPORT	Указан неправильный протокол
WSAEPROTOTYPE	Указанный протокол несовместим с данным типом сокета
WSAESOCKTNOSUPPORT	Указанный тип сокета несовместим с данным типом адреса

УДАЛЕНИЕ СОКЕТА

- Для освобождения ресурсов приложение должно закрывать сокеты, которые ему больше не нужны, вызывая функцию closesocket:

```
int closesocket(SOCKET sock);
```

- Ниже перечислены коды ошибок для этой функции:

Ошибка	Описание
WSANOTINITIALISED	Перед использованием функции closesocket необходимо вызвать функцию WSAStartup
WSAENETDOWN	Сбой в сети
WSAENOTSOCK	Указанный в параметре дескриптор не является сокетом
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса Windows Sockets
WSAEINTR	Работа функции была отменена при помощи функции WSACancelBlockingCall

ПАРАМЕТРЫ СОКЕТА

- Перед использованием необходимо задать параметры сокета. Для этого нужно подготовить структуру типа **sockaddr** , определение которой показано ниже:

```
typedef struct sockaddr {
#if (_WIN32_WINNT < 0x0600)
    u_short sa_family;
#else
    ADDRESS_FAMILY sa_family;           // Address family.
#endif //(_WIN32_WINNT < 0x0600)
    CHAR sa_data[14];                   // Up to 14 bytes of direct address.
} SOCKADDR, *PSOCKADDR, FAR *LPSOCKADDR;
```

ПАРАМЕТРЫ СОКЕТА

- Для работы с адресами в формате Internet используется другой вариант этой структуры, в котором детализируется формат поля sa_data:

```
struct sockaddr_in {
        short   sin_family;
        u_short sin_port;
        struct  in_addr sin_addr;
        char    sin_zero[8];
};
```

ПАРАМЕТРЫ СОКЕТА

- Поле sin_family определяет тип адреса. В это поле записывается значение AF_INET, которое соответствует типу адреса, принятому в Internet:

```
srv_address.sin_family = AF_INET;
```

- Поле sin_port определяет номер порта, который будет использоваться для передачи данных. Порт - это просто идентификатор программы, выполняющей обмен на сети. На одном узле может одновременно работать несколько программ, использующих разные порты. Особенностью поля sin_port является использование так называемого сетевого формата данных. Этот формат требует, чтобы младшие байты данных хранились по старшим адресам памяти. Универсальный сетевой формат данных удобен при организации глобальных сетей, так как в узлах такой сети могут использоваться компьютеры с различной архитектурой. Для выполнения преобразований из обычного формата в сетевой и обратно в интерфейсе Windows Sockets предусмотрен специальный набор функций. В частности, для заполнения поля sin_port нужно использовать функцию htons, выполняющую преобразование 16-разрядных данных из формата Intel в сетевой формат.

ПАРАМЕТРЫ СОКЕТА

- При инициализации сокета в этой структуре вы должны указать адрес IP, с которым будет работать данный сокет.
- Если сокет будет работать с любым адресом (например, вы создаете сервер, который будет доступен из узлов с любым адресом), адрес для сокета можно указать следующим образом:

```
srv_address.sin_addr.s_addr = INADDR_ANY;
```

- В том случае, если сокет будет работать с определенным адресом IP (например, вы создаете приложение-клиент, которое будет обращаться к серверу с конкретным адресом IP), в указанную структуру необходимо записать реальный адрес.
- Дейтаграммный протокол UDP позволяет посылать пакеты данных одновременно всем рабочим станциям в широковещательном режиме. Для этого нужно указать адрес как INADDR_BROADCAST.
- Если вам известен адрес в виде четырех десятичных чисел, разделенных точкой (именно так его вводит пользователь), то вы можете заполнить поле адреса при помощи функции inet_addr :

```
dest_sin.sin_addr.s_addr = inet_addr("200.200.200.201");
```

- В случае ошибки функция возвращает значение INADDR_NONE , что можно использовать для проверки.

ПАРАМЕТРЫ СОКЕТА

- Обратное преобразование адреса IP в текстовую строку можно при необходимости легко выполнить с помощью функции inet_ntoa , имеющей следующий прототип:

```
char FAR * inet_ntoa(struct in_addr  in);
```

- При ошибке эта функция возвращает значение NULL.
- Однако чаще всего пользователь работает с доменными именами, используя сервер DNS или файл HOSTS . В этом случае вначале нужно воспользоваться функцией gethostbyname , возвращающей адрес IP, а затем записать полученный адрес в структуру sin_addr :

```
PHOSTENT  phe;

phe = gethostbyname("ftp.microsoft.com");

memcpy((char FAR *)&(dest_sin.sin_addr), phe->h_addr, phe->h_length);
```

- В случае ошибки функция gethostbyname возвращает NULL. При этом причину ошибки можно выяснить, проверив код возврата функции WSAGetLastError .

ПАРАМЕТРЫ СОКЕТА

- Если же указанный узел найден в базе DNS или в файле HOSTS, функция gethostbyname возвращает указатель на структуру hostent, описанную ниже:

```
struct  hostent {
    char  FAR * h_name;             // имя узла   char
    FAR * FAR * h_aliases;          // список альтернативных имен
    short h_addr type;              // тип адреса узла
    short h_length;                 // длина адреса
    char  FAR * FAR * h_addr _list; // список адресов
    #define h_addr   h_addr_list[0] // адрес
};
typedef struct hostent  *PHOSTENT;
typedef struct hostent  FAR *LPHOSTENT;
```

- Искомый адрес находится в первом элемента списка h_addr _list[0], на который можно также ссылаться при помощи h_addr. Длина поля адреса находится в поле h_length

ПРИВЯЗКА АДРЕСА К СОКЕТУ

- После того как подготовлена структура SOCKADDR и в нее записаны параметры сокета (в частности, адрес), следует выполнить привязку адреса к сокету при помощи функции bind :

```
int bind(SOCKET sock, const struct sockaddr FAR * addr, int namelen);
```

- Параметр sock должен содержать дескриптор сокета, созданного функцией socket . В поле addr следует записать указатель на подготовленную структуру SOCKADDR , а в поле namelen - размер этой структуры.
- В случае ошибки функция bind возвращает значение SOCKET_ERROR . Дальнейший анализ причин ошибки следует выполнять при помощи функции WSAGetLastError . Возможные коды ошибок перечислены ниже

Ошибка	Описание
WSANOTINITIALISED	Перед использованием функции необходимо вызвать функцию WSAStartup
WSAENETDOWN	Сбой в сети
WSAEADDRINUSE	Указанный адрес уже используется
WSAEFAULT	Значение параметра namelen меньше размера структуры sockaddr
WSAEINPROGRESS	Выполняется блокирующая функция интерфейса Windows Sockets
WSAEAFNOSUPPORT	Этот протокол не может работать с указанным семейством адресов
WSAEINVAL	Сокет уже привязан к адресу
WSAENOBUFS	Установлено слишком много соединений
WSAENOTSOCK	Указанный в параметре дескриптор не является сокетом

СОЗДАНИЕ КАНАЛА СВЯЗИ

- Как упоминалось ранее (слайд 12), работа с сокетами содержит ряд этапов: сокет создается, настраивается на заданный режим работы, применяется для организации обмена и, наконец, ликвидируется.
- Для передачи дейтаграммных сообщений при помощи протокола негарантированной доставки UDP канал связи не нужен. Сразу после создания сокетов и их инициализации можно приступать к передаче данных. Но для передачи данных с использованием протокола TCP необходимо создать канал связи.

СОЗДАНИЕ КАНАЛА СВЯЗИ. СЕРВЕР.

- Рассмотрим процедуру создания канала связи со стороны сервера.
- **1). Прежде всего необходимо переключить сокет в состояние прослушивания, то есть в режим ожидания соединения с клиентом, при помощи функции listen**, которая организует очередь запросов:

```
int listen(SOCKET sock, int backlog);
```

- Функция listen подготавливает сокет к обработке потока запросов.
- Через параметр sock функции необходимо передать дескриптор сокета, который будет использован для создания канала связи. Параметр backlog задает максимальный размер очереди для ожидания соединения (то есть, сколько запросов может быть принято на обслуживание без потерь; обычно можно указывать значения от 1 до 5, если используется версия WinSock 1.1; в версии 2.0 и выше нет ограничений сверху). При переполнении очереди будет послано сообщение об ошибке. Очередь содержит запросы на установку соединений для каждой пары значений (адрес IP, порт). Ожидающий сокет посылает каждому отправителю сообщение-отклик, подтверждающее получение запроса на соединение. Следует иметь в виду, что клиент, ориентированный на соединение, также должен прослушивать порт протокола, ожидая появления дейтограмм-откликов.

СОЗДАНИЕ КАНАЛА СВЯЗИ. СЕРВЕР.

- **2). Далее необходимо выполнить ожидание соединения**. Это можно выполнить двумя различными способами.
- Первый способ заключается в циклическом вызове функции accept до тех пор, пока не будет установлено соединение. Затем можно будет приступать к обмену данными. Оператор accept извлекает запросы на соединение из очереди. Функция accept имеет следующий прототип:

```
SOCKET accept(SOCKET sock, struct sockaddr FAR * addr, int FAR * addrlen);
```

Через параметр sock указывается дескриптор сокета, который прослушивает соединение (тот же, что и в listen); addr — указатель на структуру, которая содержит адрес буфера, в который будет записан адрес узла, подключившегося к серверу; addrlen — код длины адреса.

СОЗДАНИЕ КАНАЛА СВЯЗИ. СЕРВЕР.

- Оператор accept позволяет серверу принять запрос от клиента. Когда входная очередь сформирована, программа реализует процедуру accept и переходит в режим ожидания запросов. Программа извлекает первый элемент очереди, создает новый сокет со свойствами, идентичными sock, и при успешном выполнении возвращает дескриптор нового сокета. При возникновении ошибки возвращается код INVALID_SOCKET. По окончании обработки запроса сервер вновь вызывает accept, который возвращает ему дескриптор сокета очередного запроса, если таковой имеется. Если очередь пуста, accept блокирует программу до получения связи. Существуют серверы с параллельной и последовательной обработкой запросов. Параллельный обработчик запросов не ждет завершения обработки предшествующего запроса и вызывает оператор accept немедленно. В системе Unix используются обычно параллельные обработчики запросов.
- Если ожидание соединения в цикле нежелательно, можно предложить другой способ, основанный на использовании расширения программного интерфейса Windows Socket, предназначенного для выполнения асинхронных операций.

СОЗДАНИЕ КАНАЛА СВЯЗИ. СЕРВЕР.

- Вместо того чтобы ожидать соединение, вызывая в цикле функцию accept , приложение может вызвать один раз функцию WSAAsyncSelect , указав ей, что при получении запроса на установку соединения функция окна вашего приложения должна получить сообщение:

```
#define WSA_ACCEPT (WM_USER + 1)
```

- При попытке установки соединения главное окно приложения получит сообщение

```
WSA_ACCEPT rc = WSAAsyncSelect(srv_socket, hWnd, WSA_ACCEPT, FD_ACCEPT);
```

- В данном случае ожидание соединения выполняется для сокета srv_socket . Последний параметр функции имеет значение FD_ACCEPT . Это означает, что при попытке создания канала связи функция окна с идентификатором hWnd получит сообщение WSA_ACCEPT, определенное в приложении.
- Обработчик этого сообщения может выглядеть, например, следующим образом (см. следующий слайд)

```
void WndProc_OnWSAAccept(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    int rc;
    // При ошибке отменяем поступление извещений  в главное окно приложения
    if(WSAGETSELECTERROR(lParam) != 0) {
        WSAAsyncSelect (srv_socket , hWnd, 0, 0);
        return;
    }
    // Определяем размер адреса сокета
    acc_sin_len = sizeof(acc_sin);
    // Разрешаем установку соединения
    srv_socket = accept (srv_socket, (LPSOCKADDR)&acc_sin, (int FAR *)&acc_sin_len);
    if (srv_socket == INVALID_SOCKET) {
        return;
    }
    // Если на данном сокете начнется передача данных от клиента, в главное окно приложения
    // поступит сообщение WSA_NETEVENT. Это же сообщение поступит при разрыве соединения
    rc = WSAAsyncSelect (srv_socket , hWnd, WSA_NETEVENT, FD_READ|FD_CLOSE );
    if(rc > 0) {
        closesocket(srv_socket);
        return;
    }
}
```

В данном случае обработчик сообщения вначале вызывает функцию accept , выполняющую создание канала передачи данных. После этого функция WSAAsyncSelect вызывается еще один раз для того, чтобы установить асинхронную обработку приема данных от удаленного клиента, а также обработку ситуации разрыва канала связи.

СОЗДАНИЕ КАНАЛА СВЯЗИ. КЛИЕНТ.

- Вначале с помощью функции socket необходимо создать сокет.
- Затем заполненим адресной информацией структуру `SOCKADDR_IN`. Для получения адреса IP можно использовать функцию gethostbyname, для которой следует указать имя узла localhost .
- Это имя отображается в файле HOSTS на адрес 127.0.0.1 . Адрес 127.0.0.1 является локальным. Его можно использовать для тестирования приложений, выполняющих обмен данными при помощи протокола TCP/IP, запуская сервер и клиент на одном и том же компьютере.
- После заполнения структуры с адресной информацией функция connect должна создать канал связи с сервером.