Оглавление

1 Сетевые API

В Windows реализован набор сетевых API.

Важно иметь в виду, что выбор API для приложения определяется характеристиками API:

- какие протоколы используются,
- поддерживает ли он надежную и двустороннюю связь,
- переносим ли он на другие Windows-платформы, на которых может работать данное приложение.

Основные сетевые API:

именованные каналы (named pipes)

почтовые ящики (mailslots);

Windows Sockets (Winsock);

Remote Procedure Call (RPC);

Некоторые API, которые являются надстройками над перечисленными выше API и широко применяются в типичных системах под управлением Windows.

1.1 Именованные каналы.

Именованные каналы обеспечивают надежную двустороннюю связь,

Серверы назначают именованным каналам и их клиентам имена в соответствии с универсальными правилами именования (Universal Naming

Convention, UNC), которые обеспечивают независимый от протоколов способ идентификации ресурсов в Windows-сетях.

1.1.1 Функционирование именованных каналов

Коммуникационная связь по именованному каналу включает сервер именованного канала и клиента именованного канала.

Сервером именованного канала является приложение, создающее именованный канал, к которому подключаются клиенты.

Формат имени канала выглядит так:

\\Сервер\Pipe\Имя Канала.

Элемент **Сервер** указывает узел, на котором работает сервер именованного канала (сервер не может создать именованный канал на удаленном узле), и его имя может быть

- DNS-именем (например, mspress.microsoft.com),
- NetBIOS-именем (mspress)
- IP-адресом (255.0.0.0).

Элемент **Pipe** должен быть строкой «Pipe»,

Имя Канала — уникальное имя, назначенное именованному каналу.

Уникальная часть имени канала может включать подкаталоги.

Пример такого UNC-имени канала **\\MyComputer\Pipe\MyServerApp\ConnectionPipe**.

1.1.2 Создание именованного канала

Для создания именованного канала сервер использует Win32-функцию **CreateNamedPipe.**

Одним из входных параметров этой функции является указатель на имя канала в форме

\\.\Pipe\ИмяКанала,

где «\\.» — псевдоним локального компьютера, определенный в Win32.

Функция также принимает

- необязательный дескриптор защиты, запрещающий

несанкционированный доступ к именованному каналу,

- флаг, указывающий, должен ли канал быть двусторонним или односторонним,
- параметр, определяющий максимальное число одновременных соединений по данному каналу,
- флаг режима работы канала (побайтовой передачи или передачи сообщений).

Большинство сетевых API работает только в режиме побайтовой передачи. Это означает, что переданное сообщение может быть принято адресатом в виде нескольких фрагментов, из которых воссоздается полное сообщение.

Именованные каналы, работающие в режиме передачи сообщений, упрощают реализацию приемника, поскольку в этом случае число передач и приемов одинаково, а приемник, разом получая целое сообщение, не должен заботиться об отслеживании фрагментов сообщений.

- При первом вызове **CreateNamedPipe** с указанием какого-либо имени создается первый экземпляр именованного канала с этим именем и задается поведение всех последующих экземпляров этого канала.
- Повторно вызывая **CreateNamedPipe,** сервер может создавать дополнительные экземпляры именованного канала, максимальное число которых указывается при первом вызове **CreateNamedPipe**.

1.1.3 Установление связи сервера с клиентом.

Создав минимум один экземпляр именованного канала, сервер выполняет Win32-функцию **ConnectNamedPipe**, после чего именованный канал позволяет устанавливать соединения с клиентами.

Функция **ConnectNamedPipe** может выполняться как синхронно, так и асинхронно, и она не завершится, пока клиент не установит соединение через данный экземпляр именованного канала (или не возникнет ошибка).

Для подключения к серверу клиенты именованного канала используют Win32-функцию **CreateFile** или **CallNamedPipe**, указывая при вызове имя созданного сервером канала,

Если сервер вызывает функцию **ConnectNamedPipe**, профиль защиты клиента и запрошенные им права доступа к каналу (для чтения или записи) сравниваются с дескриптором защиты канала

Если клиенту разрешен доступ к именованному каналу, он получает описатель, представляющий клиентскую сторону именованного канала, и функция **ConnectNamedPipe**, вызванная сервером, завершается.

После того как соединение по именованному каналу установлено, клиент и сервер могут использовать его для чтения и записи данных через Win32-функции **ReadFile** и **WriteFile**. **Именованные каналы поддерживают как синхронную, так и асинхронную передачу сообщений.**

1.2 Почтовые ящики

Почтовые ящики обеспечивают ненадежную одностороннюю передачу данных.

Почтовые ящики предоставляют механизм **ненадежного одностороннего широковещания.**

Примером приложения, использующего этот тип коммуникационной связи, является сервис синхронизации времени, который каждые несколько секунд широковещательно рассылает в пределах домена сообщение с эталонным временем. Такие сообщения не критичны для работы в сети, поэтому они рассылаются через почтовые ящики.

Как и именованные каналы, почтовые ящики интегрированы с Win32 API.

Сервер почтового ящика создает почтовый ящик вызовом **CreateMailslot**.

Входным параметром этой функции является имя в форме

«\\.\Mailslot\ИмяПочтовогоЯщика».

Сервер может создавать почтовые ящики только на том узле, на которой он работает, а назначаемые им имена почтовых ящиков могут включать подкаталоги.

CreateMailslot также принимает необязательный дескриптор защиты, контролирующий доступ клиента к почтовому ящику.

Описатели, возвращаемые **CreateMailslot,** являются перекрытыми. Это означает, что операции с использованием таких описателей (например, рассылка и получение сообщений) выполняются асинхронно.

Поскольку почтовые ящики поддерживают одностороннюю ненадежную передачу, число параметров **CreateMailslot** меньше, чем у **CreateNamedPipe**.

После создания почтового ящика сервер просто отслеживает поступающие клиентские сообщения, вызывая функцию **ReadFile** и указывая описатель, представляющий почтовый ящик.

Клиенты почтового ящика используют формат именования, аналогичный применяемому клиентами именованных каналов, за исключением вариаций, необходимых для широковещательной передачи сообщений всем почтовым ящикам с данным именем в домене клиента или в другом указанном домене.

Чтобы послать сообщение в определенный экземпляр почтового ящика, клиент вызывает функцию **CreateFile**, указав имя, специфичное для компьютера, например **«\\Сервер\Mailslot\ИмяПочтовогоЯщика».** (Для представления локального компьютера клиент указывает «\\.\».)

Если клиент хочет получить описатель, представляющий все почтовые ящики с заданным именем в домене, членом которого он является, он указывает имя в формате **«*\Mailslot\ИмяПочтовогоЯщика».**

Для широковещательной передачи во все почтовые ящики с заданным именем в другом домене используется имя в формате

«**ИмяДомена\ Mailslot \ИмяПочтовогоЯщика».**

Получив описатель, представляющий клиентскую сторону почтового ящика, клиент посылает сообщения через функцию **WriteFile.**

Реализация почтовых ящиков допускает широковещательную передачу сообщений длиной не более 425 байт.

Если длина сообщения превышает 425 байт, почтовый ящик использует механизм надежной коммуникационной связи, требующий соединения клиента с сервером по типу «один к одному», что исключает возможность широковещательной передачи.

Другая (довольно странная) особенность почтовых ящиков — урезание сообщений с исходной длиной в 425 или 426 байт до 424 байт.

Таким образом, почтовые ящики непригодны для рассылки сообщений, длина которых превышает 424 байта.

1.3 Windows Sockets (Winsock)

Сокеты подобны каналам с тем отличием, что они при нормальном использовании соединяют процессы на разных машинах.

Например, один процесс пишет в сокет, а другой процесс на удаленной машине читает из него.

Сокеты также могут использоваться для соединения процессов на одной машине, но поскольку их использование влечет за собой большие накладные расходы, чем использование каналов, то, как правило, они применяются в контексте сети

Winsock поддерживает как надежные коммуникации, ориентированные на логические соединения, так и ненадежные коммуникации, не требующие логических соединений.

Механизм сокетов обеспечивает удобный и достаточно универсальный интерфейс обмена сообщениями, предназначенный для разработки сетевых распределенных приложений. Его универсальность обеспечивают следующие концепции.

1. Независимость от нижележащих сетевых протоколов и технологий. Для этого используется понятие ***коммуникационный домен (communication***

domain). Коммуникационный домен обладает некоторым набором коммуникационных свойств, определяющих

- **способ именования сетевых узлов и ресурсов,**
- **характеристики сетевых соединений (надежные, дейтаграммные, упорядоченные),**
- **способы синхронизации процессов и т. п.**

Одним из наиболее популярных доменов является домен Интернета с протоколами стека TCP/IP.

2. Использование абстрактной конечной точки соединения, получившей название *сокет (socket* — гнездо).

Сокет — это точка, через которую сообщения уходят в сеть или принимаются из сети. Сетевое соединение между двумя процессами осуществляется через пару сокетов. Каждый процесс пользуется своим сокетом, при этом сокеты могут находится как на разных компьютерах, так и на одном (в этом случае сетевое межпроцессное взаимодействие сводится к локальному).

3. Сокет может иметь как высокоуровневое символьное имя (адрес), так и низкоуровневое, отражающее специфику адресации определенного **коммуникационного домена**. Например, в домене Интернета низкоуровневое имя представлено парой (IP-адрес, порт).

4. Для каждого коммуникационного домена могут существовать сокеты различных типов. С помощью типа сокета можно задавать определенный вид взаимодействия, имеющий смысл для домена.

1.3.1 Функционирование Winsock

После инициализации Winsock API вызовом инициализирующей функции

первый шаг Winsock-приложения - создание сокета, представляющего **конечную точку коммуникационного соединения.**

вторым шагом приложения является привязка сокета к адресу локального компьютера. Winsock не зависит от конкретного протокола,

поэтому можно выбрать любой протокол (NetBEUI, TCP/IP, IPX), установленный в системе, где работает Winsock.

По окончании привязки дальнейшие действия клиента и сервера расходятся - равно как они различны и для сокетов разных типов (ориентированных и не ориентированных на логические соединения).

Winsock-сервер, ориентированный на логические соединения, выполняет на сокете операцию **listen**, указывая число соединений, которое он может поддерживать на этом сокете. Далее он выполняет операцию **accept**, чтобы клиент мог подключиться к сокету. При наличии ждущего запроса на соединение вызов **accept** завершается немедленно. В ином случае он завершается лишь после поступления запроса на соединение. После того как соединение установлено, функция **accept** возвращает новый сокет, представляющий серверную сторону соединения. Сервер может выполнять операции приема и передачи данных с помощью функций **recv** и **send**.

Клиенты, ориентированные на логические соединения, подключаются к серверу вызовом Winsock-функции **connect** с указанием адреса удаленного компьютера.

После того как соединение установлено, клиент может посылать и принимать сообщения через свой сокет.

После привязки к адресу сервер, не требующий логических соединений, ничем не отличается от аналогичного клиента; он посылает и получает сообщения через сокет, просто указывая удаленный адрес для каждого сообщения.

При использовании сообщений, не требующих логических соединений (т. е. дейтаграмм), отправитель получает код ошибки в ходе следующей операции приема, если предыдущее сообщение не было доставлено.

1.3.2 Расширения Winsock

С точки зрения программирования для Windows, сильной стороной Winsock API является его интеграция с механизмом Windows-сообщений.

Winsock-приложение может использовать преимущества такой интеграции для выполнения асинхронных операций с сокетом и приема уведомления о завершении операции через стандартное Windows-сообщение или функцию обратного вызова.

Это упрощает разработку Windows-приложений, поскольку позволяет отказаться от многопоточности и синхронизирующих объектов для поддержки сетевого ввода-вывода и реакции на пользовательский ввод или запросы диспетчера окон на обновление окон приложения.

Имена основанных на сообщениях версий Winsock-функций в стиле BSD обычно начинаются с префикса WSA, например, **WSAAccept.**

Кроме вспомогательных функций, прямо соответствующих функциям, реализованным в BSD Sockets, Microsoft добавила несколько функций, не входящих в стандарт Winsock.

Две из них, **AcceptEx** и **TransmitFile**, стоят того, чтобы привести здесь их описание, так как благодаря им многие Web-серверы под управлением Windows 2000 достигают высокой производительности.

AcceptEx является версией функции **accept**, которая в процессе установления соединения с клиентом возвращает адрес и первое сообщение клиента. А это позволяет Web-серверу избежать выполнения сразу нескольких Winsock-функций.

Установив соединение с клиентом, Web-сервер обычно посылает ему файл, например, Web-страницу

Реализация функции TransmitFile интегрирована с диспетчером кэша Windows, что позволяет серверу посылать файл непосредственно из кэша файловой системы. Такая пересылка данных называется нулевым копированием (zero-copy), поскольку в этом случае серверу не приходится обращаться к файловым данным: он просто указывает описатель файла и диапазон пересылаемых байт. Кроме того, функция TransmitFile позволяет серверу присоединять к началу или концу файла дополнительные данные. Это

дает ему возможность посылать заголовочную информацию, например, имя Web-сервера и поле, в котором указывается размер посылаемого сообщения.

Internet Information Services (IIS) 5-0, входящий в комплект поставки Windows, использует как **AcceptEx**, так и **TransmitFile**.

1.3.3 Расширение Winsock

Winsock является расширяемым API, поскольку сторонние разработчики могут добавлять компоненты доступа к транспортным сервисам (transport service providers, TSP), организующие интерфейсы между Winsock и другими протоколами, а также компоненты доступа к пространствам имен (namespace service providers), дополняющие механизмы разрешения имен в Winsock.

Такие компоненты подключаются к Winsock через его интерфейс компонентов доступа к сервисам (service provider interface, SPI). Если какой-то TSP регистрируется в Winsock, последний реализует на его основе функции сокета (вроде connect и accept) для тех типов адресов, которые указаны этим компонентом доступа как поддерживаемые. Никаких ограничений на то, как TSP реализует функции, не налагается, но такая реализация обычно требует взаимодействия с драйвером транспорта в режиме ядра.

Требование к любому клиент-серверному приложению, использующему Winsock, заключается в следующем: сервер должен сделать свой адрес доступным клиентам, чтобы они могли подключаться к серверу. Для стандартных сервисов, выполняемых в TCP/IP, с этой целью используются так называемые общеизвестные адреса. Если браузер знает имя компьютера, на котором работает Web-сервер, он может подключиться к нему, указав общеизвестный адрес Web-сервера (к IP-адресу сервера добавляется строка «:80» — номер HTTP-порта).

Компоненты доступа к пространствам имен позволяют серверам регистрировать сво< присутствие и другими способами. Например, компонент доступа к пространству имен мог бы на серверной стороне регистрировать

адрес сервера в **Active Directory**, а на клиентской — искать его в **Active Directory**.

Компоненты доступ; к пространствам имен обеспечивают эту функциональность Winsock, реализуя такие стандартные Winsock-функции разрешения имен, как **gethostbyaddr**, **getservbyname** и **getservbyport**.

1.4 Remote Procedure Call (RPC)

RPC — стандарт сетевого программирования, разработанный в начале 80-х.

Организация Open Software Foundation (теперь — The Open Group) сделала RPC частью стандарта OSF DCE (Distributed Computing Environment).

Несмотря на наличие второго стандарта RPC, SunRPC, реализация RPC от Microsoft совместима со стандартом OSF DCE.

RPC, опираясь на другие сетевые API (именованные каналы или Winsock), предоставляет альтернативную модель программирования, в какой-то мере скрывающую детали сетевого программирования от разработчика приложений.

Функционирование RPC

Механизм RPC позволяет создавать приложения, состоящие из произвольного числа процедур, часть которых выполняется локально, а часть — на удаленных компьютерах (через сеть).

RPC предоставляет модель работы с сетью, ориентированную на процедуры, а не на транспорты, что упрощает разработку распределенных приложений.

Сетевое программное обеспечение традиционно базируется на модели обработки ввода-вывода.

- В Windows сетевая операция начинается с того, что приложение выдает запрос на удаленный ввод-вывод.
- Операционная система обрабатывает запрос, передавая его редиректору, который действует в качестве удаленной

файловой системы, обеспечивая прозрачное взаимодействие клиента с удаленной файловой системой.

- Редиректор передает запрос удаленной файловой системе, а после того как удаленная система выполнит запрос и вернет результаты, локальная сетевая плата генерирует прерывание. Ядро обрабатывает это прерывание, и исходная операция ввода-вывода завершается, возвращая результаты вызывающей программе.

RPC использует совершенно другой подход. Приложения RPC похожи на другие структурированные приложения: у них есть основная программа, которая для выполнения специфических задач вызывает процедуры или библиотеки процедур.

Отличие приложений RPC от обычных программ в том, что некоторые библиотеки процедур в приложениях RPC выполняются на удаленных компьютерах, а некоторые - на локальном.

Для приложения RPC все процедуры кажутся локальными. Иначе говоря, вместо того чтобы заставлять программиста писать код для передачи запросов на вычисления или ввод-вывод по сети, работы с сетевыми протоколами, обработки сетевых ошибок, ожидания результатов и т.д., программное обеспечение RPC выполняет все эти задачи автоматически.

Кроме того, механизм RPC в Windows работает с любыми транспортами, которые имеются в системе.

Создавая приложение RPC, программист решает, какие процедуры будут выполняться локально, а какие - удаленно.

Функционирует приложение RPC следующим образом:

В процессе своей работы приложение вызывает как локальные процедуры, так и процедуры, отсутствующие на локальной машине.

Для обработки последнего случая приложение связывается с локальной DLL, которая содержит интерфейсные процедуры **(stub procedures)** для всех удаленных процедур.

В простой программе интерфейсные процедуры статически связываются с приложением, но в компоненте большего размера они включаются в отдельные DLL. В DCOM обычно применяется последний метод. Интерфейсная процедура имеет то же имя и тот же интерфейс, что и удаленная процедура, но вместо выполнения соответствующей операции она просто преобразует переданные ей параметры для передачи по сети — такой процесс называется **маршалингом (marshaling**).

Маршалинг заключается в упорядочении параметров и их упаковке в определенном формате.

Далее интерфейсная процедура вызывает процедуры библиотеки RPC периода выполнения, и они находят компьютер, на котором расположены удаленные процедуры, определяют используемые этим компьютером механизмы транспорта и посылают запрос при помощи локального программного обеспечения сетевого транспорта.

Когда удаленный сервер получает запрос RPC, он выполняет обратное преобразование параметров **(unmarshaling)**, реконструирует исходный вызов процедуры и вызывает ее.

Закончив обработку, сервер выполняет обратную последовательность действий для возврата результатов вызывающей программе.

Кроме интерфейса, основанного на описанном здесь синхронном вызове процедур, RPC в Windows также поддерживает асинхронный RPC (**asynchronous RPC**).

Он позволяет приложению RPC вызывать функцию и, не дожидаясь ее выполнения, продолжать свою работу. На это время приложение может перейти к выполнению другого кода. Когда от сервера придет ответ, библиотека RPC периода выполнения освободит объект-событие, сопоставленный клиентом с асинхронным вызовом. Чтобы узнать о завершении функции, клиент использует стандартные Win32-функции типа **WaitForSingleObject.**

Помимо библиотеки периода выполнения в Microsoft RPC входит компилятор MIDL (Microsoft Interface Definition Language). Этот компилятор упрощает создание приложений RPC. Программист пишет набор обычных прототипов функций (предполагается, что он использует язык C или C++), описывающих удаленные процедуры, а затем помещает их в какой-либо файл. Далее он добавляет к этим прототипам нужную дополнительную информацию, например, уникальный для сети идентификатор пакета процедур, номер версии и атрибуты, указывающие, являются ли параметры входными, выходными или и теми, и другими одновременно. В конечном счете программист получает файл на языке IDL (Interface Definition Language).

Подготовленный IDL-файл транслируется компилятором MIDL, который создает интерфейсные процедуры для клиентской и серверной сторон, а также заголовочные файлы, включаемые в приложение. Когда клиентское приложение связывается с файлом интерфейсных процедур, компоновщик разрешает все ссылки на удаленные процедуры. Аналогичным образом удаленные процедуры устанавливаются на серверной машине. Программист, который намерен вызывать существующее приложение RPC, должен написать только клиентскую часть программы и скомпоновать ее с локальной библиотекой RPC периода выполнения.

Библиотека RPC периода выполнения использует для взаимодействия с транспортным протоколом универсальный интерфейс провайдеров транспорта RPC **(RPC transport provider interface).**

Этот интерфейс служит тонкой прослойкой между механизмом RPC и транспортом, которая увязывает операции RPC с функциями, предоставляемыми транспортом.

RPC в Windows реализует DLL-модули провайдеров транспорта для именованных каналов, NetBIOS и TPC/IP

Поддержку других транспортов можно ввести, написав дополнительные DLL. Аналогичным образом RPC поддерживает работу с различными средствами сетевой защиты.

Большинство сетевых служб Windows является приложениями RPC, а это значит, что они могут вызываться как локальными процессами, так и процессами на удаленных машинах.

Таким образом, удаленный клиентский компьютер может обращаться к службам сервера для просмотра списка общих ресурсов, открытия файлов, записи данных в очереди печати или добавления пользователей на этом сервере, либо он может вызывать Messenger Service (Службу сообщений) для посылки сообщений (конечно, при наличии соответствующих прав доступа).

Сервер может регистрировать свое имя по адресу, который будет доступен клиенту при поиске. Эта возможность, называемая публикацией имени сервера, реализована в RPC и интегрирована с Active Directory. Если Active Directory не установлена, служба локатора имен возвращается к широковещательной рассылке с использованием NetBIOS. Это позволяет взаимодействовать с системами под управлением Windows NT 4 и дает возможность RPC функционировать на автономных серверах и рабочих станциях.

Защита в RPC

RPC интегрирован с компонентами поддержки защиты (security support providers, SSP), что позволяет клиентам и серверам RPC использовать аутентификацию и шифрование при коммуникационной связи.

Когда серверу RPC требуется защищенное соединение, он должен зарегистрироваться в SSP под именем участника безопасности (principal name), специфичным для SSP. Во время привязки к серверу клиент регистрирует свои удостоверения защиты, указывая имя участника безопасности, принадлежащее серверу. В процессе привязки клиент также указывает нужный уровень аутентификации. Различные уровни аутентификации обеспечивают подключение к серверу только авторизованных клиентов, проверку каждого сообщения, получаемого сервером, (на предмет того послано ли оно авторизованным клиентом), контроль за целостностью RPC-сообщений и даже шифрование данных RPC-

сообщений. Чем выше уровень аутентификации, тем больше требуется обработки.

SSP берет на себя все, что связано с аутентификацией и шифрованием при связи через сеть, не только для RPC, но и для Winsock.

В Windows имеется несколько встроенных SSP, в том числе Kerberos SSP, реализующий аутентификацию Kerberos v5, и SChannel (Secure Channel), реализующий Secure Sockets Layer (SSL), протокол TLS (Transport Layer Security) и PCT (Private Communication Technology).

Если SSP не указан, программное обеспечение RPC использует встроенную защиту именованных каналов.