GeekBrains

Компьютерные сети

Транспортный уровень. Часть 1. Сокеты

Задачи транспортного уровня. Идея сокетов

Введение

OSI/ISO	TCP/IP (DOD)
7. Прикладной уровень	4. Уровень приложений
6. Уровень представления	
5. Сеансовый уровень	
4. Транспортный уровень	3. Транспортный уровень
3. Сетевой уровень	2. Сетевой уровень
2. Канальный уровень	1. Уровень сетевых интерфейсов
1. Физический уровень	

Мы изучили сетевой уровень, которые связывает хосты между собой, и переходим к изучению транспортного уровня. Транспортный уровень называется одинаково и в модели OSI/ISO, и в стеке TCP/IP. Но задачи у этих двух уровней не совсем идентичные. В модели OSI/ISO транспортный уровень обеспечивает надежную связь «точка-точка», то есть от хоста к хосту, таким образом, чтобы маршрутизаторы и промежуточные узлы были прозрачными. В модели TCP/IP транспортный уровень фактически идентифицирует приложения, для чего используются номера портов. Это характерно для TCP/IP, но не обязано выполняется в других стеках. Так, в стеке IPX/SPX идентификация приложения (идентификатор сокета) выполняется на сетевом уровне, в адресе IPX, и это нисколько не противоречит модели OSI/ISO.

На прошлых занятиях мы рассмотрели, как происходит физическая адресация и для чего она нужна, как происходит адресация на сетевом уровне. Как вы помните, на канальном уровне мы можем идентифицировать сетевые интерфейсы в пределах сети, на сетевом уровне мы можем идентифицировать хосты, даже находящиеся в других сетях (так как сетевой адрес содержит и часть, идентифицирующую сеть, и часть, идентифицирующую хост). Но нам необходимо определить, не только с какого компьютера на какой передавать информацию, но и какому приложению отдать полученные данные. На одном компьютере могут быть запущены несколько серверов: HTTP-сервер, почтовый сервер, удаленный доступ по SSH.

Теперь мы подходим к рассмотрению к межпроцессному взаимодействию.

Канальный уровень — это взаимодействие между сетевыми интерфейсами, сетевой уровень — взаимодействие между хостами. Транспортный же уровень — взаимодействие между процессами (неважно, на одной или на разных машинах они запущены).

Транспортный уровень принимает данные прикладных протоколов, которые реализуются непосредственно прикладными программами, при необходимости разбивает поток данных на фрагменты (сегменты), снабжает каждый сегмент или дейтаграмму (если данные на прикладном уровне меньше MTU и используется транспортный протокол без гарантированной доставки), добавляет заголовок транспортного уровня, в котором указаны порт отправителя и порт получателя, — специальные идентификаторы, необходимые для идентификации приложения на хосте отправителя и приложения на хосте получателя, между которыми происходит обмен сообщениями. Заголовки также содержат контрольную сумму, чтобы можно было контролировать целостность данных при движении между сетями (контрольная сумма канального уровня, как вы помните, позволяет только проверить, что данные не были повреждены именно на конкретном участке пути при передаче через физическую среду, но не были ли данные повреждены ранее, она не покажет). Кроме того, в зависимости от протокола, в транспортном заголовке могут содержаться и другие данные, которые мы сегодня рассмотрим.

Отметим, что данный подход характерен только для стека TCP/IP. Согласно модели OSI/ISO, транспортный уровень связывает отправителя и получателя, обеспечивая гарантированную или негарантированную доставку, а задача идентификации приложений не привязана строго к транспортному уровню. Стоит отметить, что в стеке IPX/SPX идентификация приложений осуществлялась на сетевом уровне, и адреса отправителя и получателя помимо адреса сети и адреса хоста содержали также идентификатор сокета (для отправителя и получателя соответственно). Таким образом, IPX-пакеты, или, как еще принято говорить, IPX-дейтаграммы, являлись аналогом не столько IP-пакетов, сколько UDP-дейтаграмм, вложенных в IP-пакеты.

Типы транспортных протоколов

Обычно различают транспортные протоколы с установкой соединения и протоколы без установки соединения. В стеке TCP/IP, как правило, под первым в большинстве случаев подразумевается протокол TCP (Transmission Control Protocol), под вторым — UDP (User Datagram Protocol). В других стеках протоколов могут быть и иные протоколы. Так, в стеке IPX/SPX изначально для гарантированной доставки применялся транспортный протокол SPX (Sequenced Packet eXchange), решающий те же задачи, что и TCP в стеке TCP/IP. Также говорят, что TCP обеспечивает гарантированную доставку сообщений, а UDP — негарантированную.

Протоколы с установкой соединения используются, когда требуется надежное соединение, когда прикладная программа не решает вопросы сборки пакетов, проверки последовательности их приема и когда последовательность приема данных и идентичность полученных данных отправленным важна. Т. е. текстовый документ или двоичный файл будет отправляться в большинстве случаев через протокол с установкой соединения. Но есть и исключения: протокол TFTP (Trivial File Transfer Protocol), используемый для загрузки образа операционной системы с другой машины на бездисковые станции, и альтернативный протокол для передачи гипертекста от Google — QUIC («быстрый») самостоятельно решают задачи установки соединения и гарантированной доставки на вышестоящих

уровнях модели OSI/ISO (сеансовом, представления, прикладном), работая поверх транспортного протокола UDP. Протоколы с установкой соединения также используются, когда требуется организовать двусторонний канал обмена, как правило, текстовыми сообщениями или текстовыми сообщениями и файлами, то есть осуществить работу в режиме чата, когда двое или несколько участников могут переписываться в обе стороны (например, XMPP для Jabber), и, как несложно догадаться, сюда же относятся реализации виртуальных терминалов (не что иное, как «чат» с удаленной машиной) — такие протоколы, как Telnet и SSH, позволяющие получить доступ к консоли или оболочке удаленной машины.

Протоколы без установки соединения используются, когда требуется доставка небольших порций данных и когда потеря или порядок переданных сообщений не критичен, когда скорость передачи важнее гарантированной доставки всех сообщений. К ним относится UDP, который используется для онлайн-трансляций и VoIP. Если несколько сообщений выпадут, это приведет лишь к искажению голоса, если же мы будем ожидать гарантированной доставки, может возникнуть задержка, которая практически неизбежна при использовании для этих целей TCP. В то же время на качество передачи будет влиять и канал связи: в частности, низкая скорость соединения может потребовать применения соответствующих кодеков на прикладном уровне, которые уменьшают размер передаваемой информации благодаря снижению качества голоса (т. е. голос будет звучать словно «роботизированный», как из рации). На эту тему можно почитать по ключевому слову «вокодеры». UDP также может использоваться в реализации онлайн-игр.

Стоит отметить, что некоторые протоколы могут одновременно использовать и TCP, и UDP, но обычно для разных задач или как дополнительную опцию. Наиболее характерный пример — протокол DNS (Domain Name Service), использующий для запросов и ответов UDP, а для передачи файлов доменных зон между первичным и вторичными серверами — протокол TCP.

Итак, резюмируем. В стеке TCP/IP транспортный уровень решает следующие сетевые задачи:

- сегментирование данных, полученных от протоколов прикладного уровня, на фрагменты (TCP-сегменты для протокола TCP) для передачи по сети;
- нумерация и упорядочивание сегментов;
- буферизация дейтаграмм/сегментов;
- сопоставление и адресация процессов (приложение) и сетевых запросов (создание сокетов);
- управление интенсивностью передачи.

Основные протоколы транспортного уровня — TCP и UDP. Также следует знать, что есть перспективные альтернативные протоколы транспортного уровня: SCTP и DCCP.

Идентификация процессов, порты, сокеты

Каждое приложение, получая доступ к сетевым услугам, использует номер порта, чтобы идентифицировать себя как удаленное приложение (если приложение обращается к серверу) либо

чтобы другие приложения могли обратиться к данному серверу удаленно (если приложение само является сервером). Клиентские приложения (равно как и серверные приложения, которые подключаются к другим серверам как клиенты — так, например, работает прокси-сервер) используют динамические порты, назначаемые операционной системой. Сетевой порт — это число, двухбайтовый идентификатор, который идентифицирует на данном хосте приложение (как сервер, так и клиент). Не путайте его с портом маршрутизатора или портом на материнской плате. На транспортном уровне порт — понятие исключительно логическое, номер, по которому мы идентифицируем приложение на хосте-отправителе (которому нужно вернуть ответ) и приложение на хосте-получателе (которому следует передать сообщение).

В стеке TCP/IP используется два набора портов: один с гарантированной доставкой (TCP-порты, STREAM-сокеты), второй — без гарантированной доставки (UDP-порты, DGRAM-сокеты). Это два разных пространства идентификаторов, возможна ситуация, когда одно приложение использует TCP-порт, а другое UDP-порт с тем же номером. Эти два приложения не будут мешать друг другу. С другой стороны, одно приложение для реализации услуги может использовать и TCP-, и UDP-порт (например, когда нужна передача больших файлов или дополнительная надежность — TCP, а когда нужно отправлять короткие сообщения — UDP). Так, DNS использует порт 53/UDP для коротких DNS-запросов и ответов, а 53/TCP — для обмена между первичным и вторичными серверами объемными файлами зон.

Номера портов, как правило, общеизвестны и зарезервированы для тех или иных сервисов. При этом не уточняется, какой используется протокол — TCP или UDP. На практике используется либо тот, либо другой, либо оба, но в зависимости от ситуации.

Два байта позволяют идентифицировать до 65 545 портов (то есть на каждом хосте теоретически имеется 65 545 портов для передачи данных с гарантированной доставкой, т. е. условно TCP-портов, и 65 545 портов для передачи данных с негарантированной доставкой, т. е. UDP-портов).

Порты с номерами 1–1023, как правило, зарезервированы для стандартных служб. В Linux прослушивание портов до 1023-го требует прав суперпользователя (root). Также имеется диапазон локальных портов, динамически выделяемых для клиентских приложений. Например, в Linux это могут быть порты от 32 768 по 60 999 (диапазон содержится в файле /proc/sys/net/ipv4/ip_local_reserved_ports).

```
cowboy@tom3:~$ cat /proc/sys/net/ipv4/ip_local_port_range
32768   60999
```

Назначение портов определяется Администрацией адресного пространства Интернет (IANA — Internet Assigned Numbers Authority, она же выдает группы IP-адресов и номера автономных систем RIR — региональным интернет-регистраторам). При этом программное обеспечение может использовать и незарегистрированные или даже зарегистрированные для других целей порты (частым примером является использование 80 или 443 порта, предназначенных для HTTP/HTTPS, службой SSH, чтобы

обойти запрет на доступ к порту 22 либо вообще ко всем портам, кроме предназначенных для передачи гипертекста).

Некоторые общеизвестные номера портов:

- 20 — передача данных по протоколу FTP;
- 21 — передача команд по протоколу FTP;
- 22 — SSH (и заодно SFTP, являющийся надстройкой над SSH);
- 23 — Telnet;
- 25 — SMTP;
- 53 — DNS;
- 67 — BOOTP, DHCP (сервер);
- 68 — BOOTP, DHCP (клиент);
- 69 — TFTP;
- 70 — Gopher — предшественник HTTP;
- 80, 8080 — HTTP;
- 123 — NTP, SNTP;
- 110 — POP3;
- 161 — SNMP;
- 143 — IMAP4;
- 179 — BGP (Border Gateway Protocol);
- 443 — HTTPS (HTTP, инкапсулированный в TLS);
- 520 — RIP;
- 989 — FTPS (FTP, инкапсулированный в TLS) — данные;
- 990 — FTPS (FTP, инкапсулированный в TLS) — управление;
- 1935 — RTMP (Real Time Messaging Protocol) — используется в вебинарах (например в Clickmeeting; это пример неофициального использования порта);
- 3306 — MySQL;
- 5060 — SIP.

Также в исходящих дейтаграммах UDP-соединения в качестве исходящего порта может быть указан 0, если не предполагается получение сообщений в ответ.

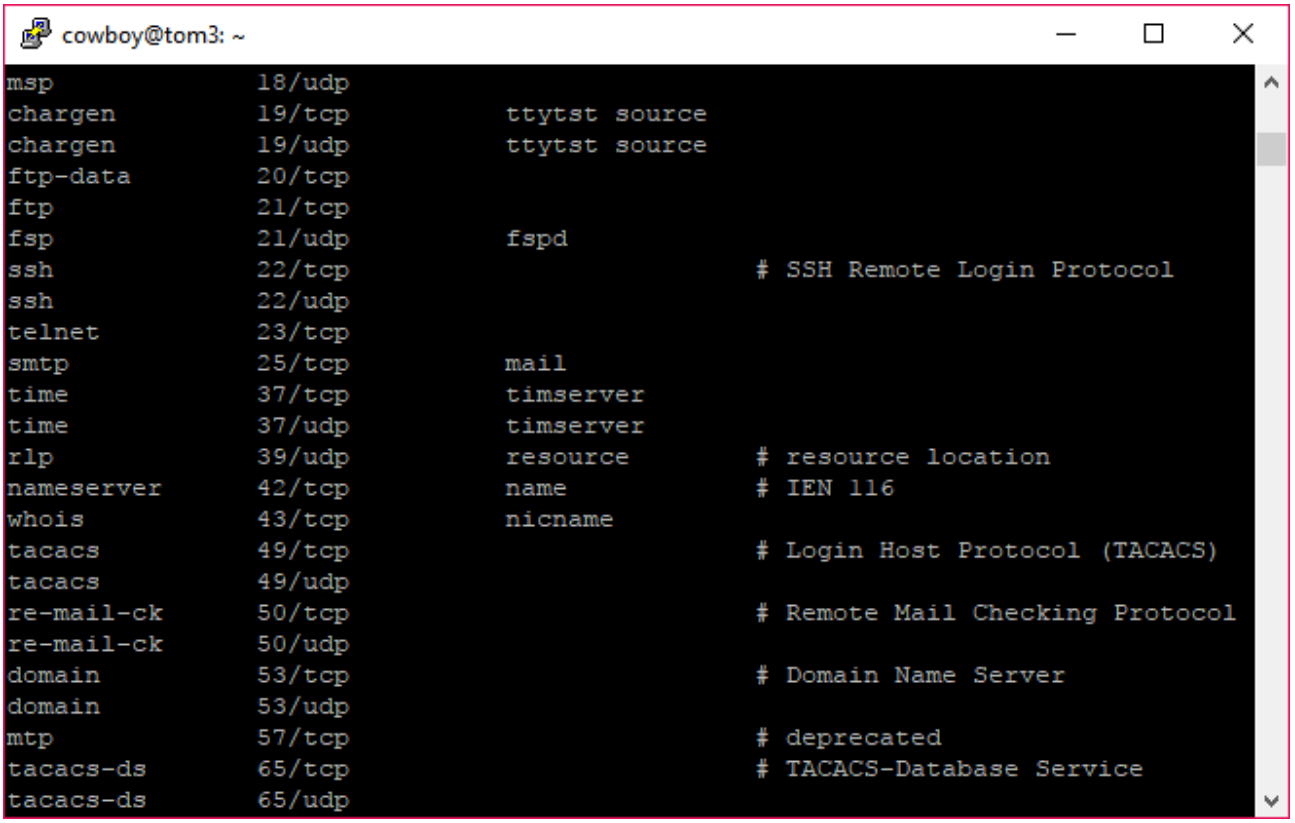
Механизм портов в TCP/IP таков, что сервер должен открыть слушающий сокет (указав используемый стек, IPv4, IPX, IPv6 и тип соединения, STREAM или DGRAM), связать его с IP-адресом (может использоваться IP-адрес хоста или 0.0.0.0, если будут прослушиваться все адреса) и номером порта (в зависимости от типа соединения будет использоваться пространство TCP или UDP-номеров). Далее сервер ожидает входящих сообщений или соединений. Клиентское приложение также должно открыть сокет, но вместо того, чтобы прослушивать входящие соединения, клиентский сокет сам подключается к удаленной машине (указав свой IP-адрес, IP-адрес сервера и порт сервера), а подключившись, может ожидать и входящих сообщений после подключения и до закрытия сокета — в

принципе, соединения равноправны. Локальный порт клиентскому приложению назначается автоматически операционной системой, более того, приложение номер этого порта даже и не знает.

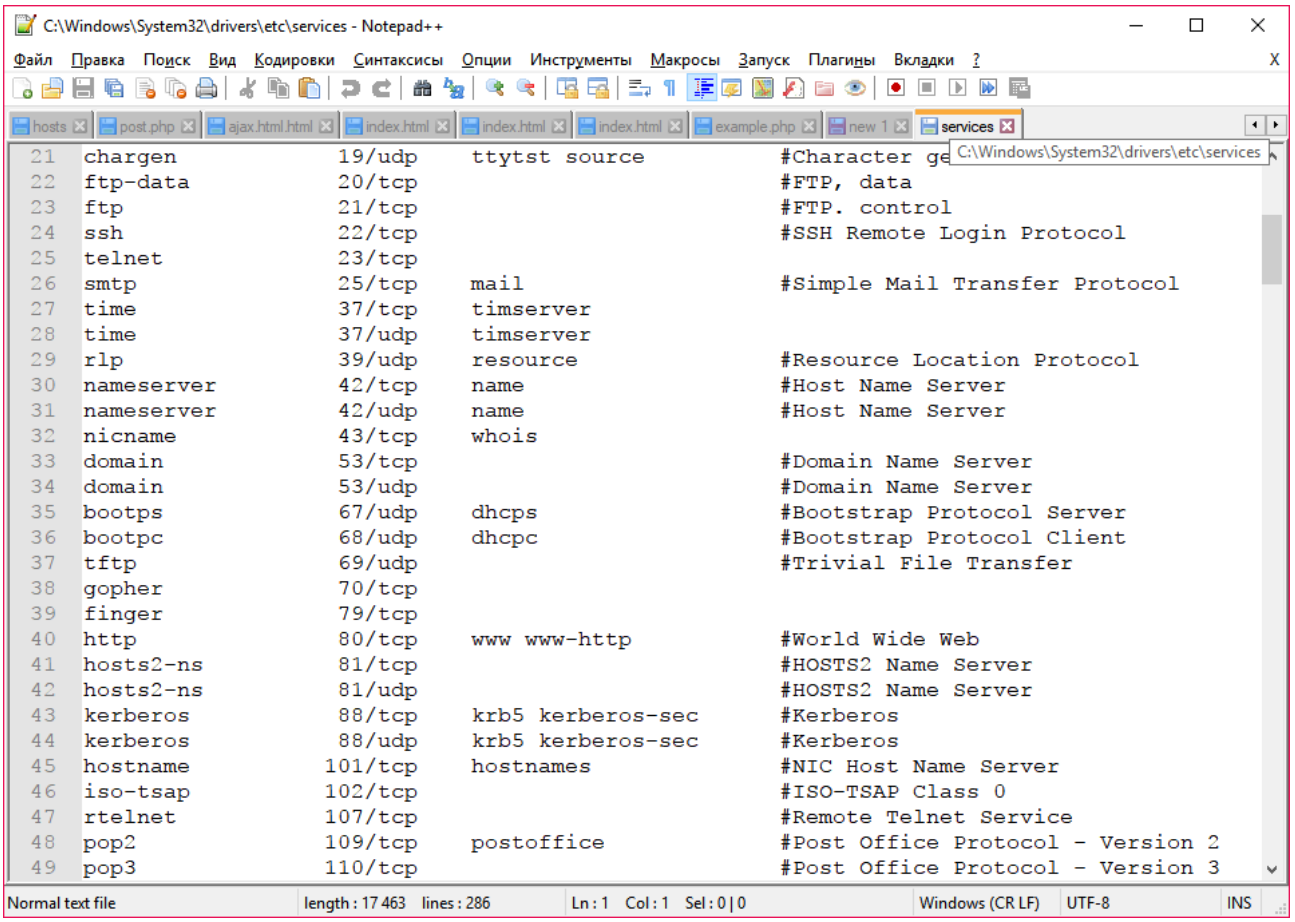
Таким образом, формально соединение является дуплексным. Интересно, что в предшественнике TCP/IP, протоколе NCP (Network Contorl Program, использовался в ARPANET), соединения являлись симплексными и использовали по два симплексных порта — один для отправки сообщения, второй для приема. Еще интереснее, что и сейчас в стеке TCP/IP иногда используются пары портов — как в FTP, хотя там один порт нужен для данных, другой — для управления соединением. Но есть один случай, когда для обмена сообщениями используются два по-настоящему разных порта, — протокол DHCP: сервер использует порт 67/UDP, а клиент — 68/UDP, впрочем, они все равно работают в дуплексном режиме (клиент отправляет со своего 68 порта на 67 порт сервера, а сервер — со своего 67 порта на 68 порт клиента). Аналогичная картинка сохранилась и в DHCPv6 (546 порт — клиентский, 547 — серверный).

В Linux перечень портов и сервисов, для которых они зарезервированы, можно посмотреть в файле /etc/services:

```
$ cat /etc/services
```



Аналогичный файл есть и в Windows (C:\Windows\System32\drivers\etc\services).



Порты с номерами выше 49 152 называются динамическими и используются клиентским программным обеспечением.

Говорят, что для TCP и UDP используется независимая нумерация портов, т. е. одновременно, не мешая друг другу, на одной и той же машине могут быть открыты два порта с одним и тем же номером, TCP и UDP. Впрочем, это не совсем верно, так как кроме TCP и UDP есть другие протоколы.

Сокет (программный интерфейс)

Технология сокетов Беркли — фундаментальная для стека протоколов TCP/IP. Впервые они появились в BSD UNIX в 1982 году, и их основные идеи используются до сих пор.

Интерфейс сокета Беркли используется для взаимодействия между компьютерами в сети или процессами, запущенными на компьютере. Сокеты — это стандарт интерфейсов для транспортных подсистем. Различные варианты сокетов могут быть реализованы в разных ОС и языках программирования.

Сокет в Unix — интерфейс, доступ к которому похож на работу с файлом — в соответствии с концепцией unix-way, одним из положений которой является то, что в Unix всё является файлом (устройства, структуры процессора, механизмы межпроцессного взаимодействия и т. д.). Аналогично и

операции сокетов изначально были подобны операциям работы с файлами: функции **read()** и **write()** — те же, что и для файлов, аналогичное закрытие с помощью **close()**.

Для начала приложение должно открыть сокет, используя функцию **socket()**. Функция возвращает дескриптор сокета (аналогия с файловым дескриптором, пришедшая из unix-way). При этом указывается, какой стек протоколов будет использоваться (IPv4, IPv6, Unix-сокеты, использующие для межпрограммного взаимодействия механизм файловых сокетов), тип используемого протокола (STREAM, DGRAM, RAW) и протокол.

Тип используемого сокета:

- STREAM — для передачи с установкой соединения; используется в TCP и SCTP, а также для UNIX-сокетов;
- DGRAM — для передачи дейтаграмм, то есть без установки соединения; используется в UDP и DCCP;
- RAW — сырой сокет, т. е. непосредственно работающий с IP-пакетами;
- другие типы, в том числе позволяющие непосредственно захватывать кадры канального уровня.

Протокол:

- TCP;
- UDP;
- SCTP;
- DCCP;
- RAW (для сырых сокетов).

Работа с сокетом аналогична работе с файлом, но фактически запись в сокет приводит к передаче файла по сети (сразу или после накопления определенного количества байт в буфере), а чтение происходит из буфера полученных данных. То есть все действия сводятся к чтению или записи, сама передача данных прозрачна для разработчика.

Практика показала, что в чистом виде сокеты — нечто большее, чем файлы, потому набор операций пополнили сокет-специфичные (**send()**, **recv()** для отправки и получения).

Операция SOCKET создает новый сокет и записывает его в таблицу транспортной подсистемы. Параметры вызова задают тип используемого формата адресации, тип применяемого сервиса (например, надежный поток байтов) и протокол.

Например, при обращении к серверу geekbrains.ru на HTTP-порт сокет будет выглядеть так: 5.61.239.21:80, а ответ будет поступать на IP-адрес запросившей машины и случайный динамический порт клиента (для клиентских приложений используются динамические порты, а для серверных, как правило, фиксированные).

Базовые операции сокетов для TCP

- SOCKET (СОКЕТ) — создание нового сокета;
- BIND (СВЯЗАТЬ) — привязать локальный IP-адрес и порт;
- LISTEN (ОЖИДАТЬ) — слушать входящие соединения, указав размер очереди;
- ACCEPT (ПРИНЯТЬ) — подтвердить установление входящего соединения;
- CONNECT (СОЕДИНИТЬ) — инициировать процесс установления соединения с удаленной машиной;
- SEND (ПОСЛАТЬ) — передать информацию по установленному соединению;
- RECEIVE (ПОЛУЧИТЬ) — принять информацию по установленному соединению;
- CLOSE (ЗАКРЫТЬ) — закрыть сеанс связи и отправить сообщение о завершение соединения.

bind() используется сервером, чтобы связать с сокетом прослушиваемые IP-адрес и порт.

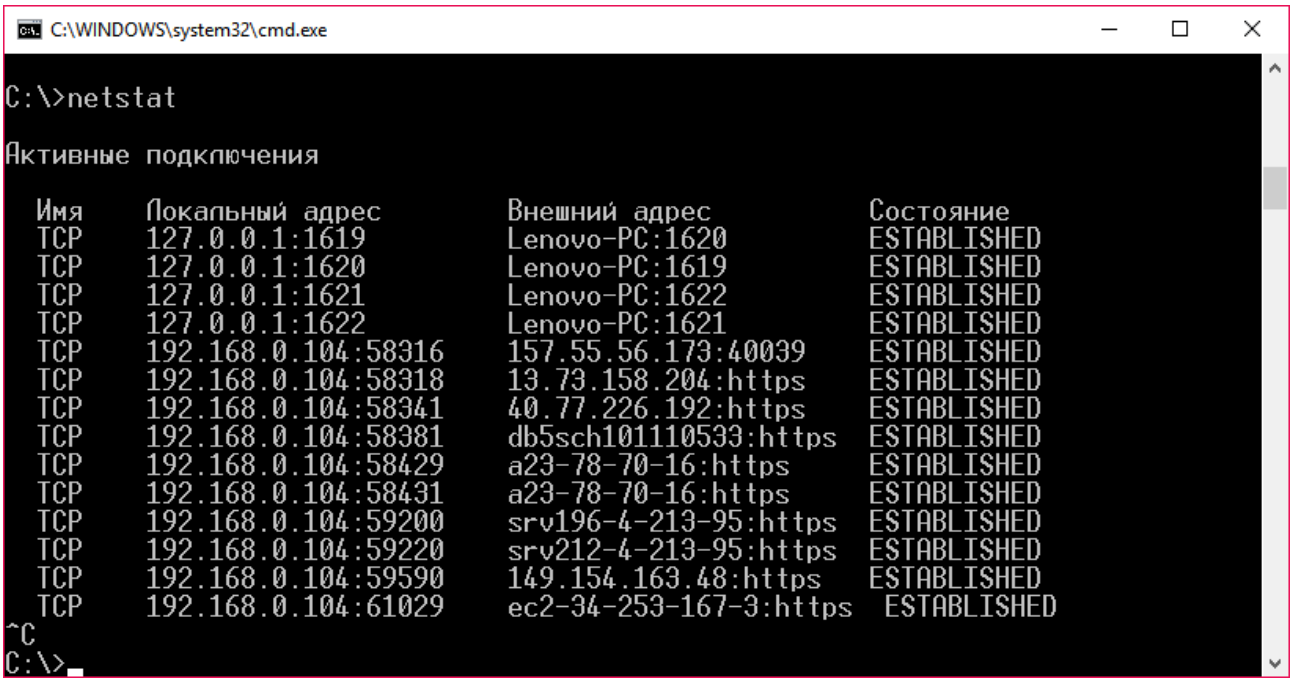
connect() служит для того, чтобы подключиться к серверу, для чего в аргументах функции указывается (в составе структуры), с какого IP-адреса, на какой IP-адрес и к какому порту мы будем подключаться.

Для перевода сокета в прослушивающее состояние (актуально для сервера) используется **listen()**.

Для закрытия сокета используется **close()**.

Также есть механизм для разрешения доменных имен **gethostbyname()**. Происходит обращение к файлу /etc/hosts к локальной DNS-службе, которая уже ищет по распределенной базе IP-адрес, соответствующий указанному доменному имени. Хоть DNS и протокол прикладного уровня, в данном контексте он совершенно особенный, так как используется независимо от того, создаем ли мы сокет с установкой соединения, без установки соединения или даже сырой.

Посмотреть соединения (как в Windows, так и в Linux) можно с помощью команды **netstat**.



В Windows ключ **-a** также позволяет отобразить не только установленные соединения, но и прослушиваемые (listen) порты. Похожее поведение в Linux. Дополнительно в Linux можно отобразить только прослушиваемые подключения с помощью ключа **-l**.

Самостоятельно посмотрите все возможности программы **netstat** с помощью ключа **--help**.

Место протоколов TCP и UDP в модели OSI

Протоколы TCP и UDP традиционно относятся к транспортному уровню. При этом задачи, решаемые TCP и UDP, не обязательно могут относиться к транспортному уровню модели OSI/ISO. Так, протокол UDP транспортного уровня стека TCP/IP является во многом аналогом сетевого протокола IPX-стека IPX/SPX. Не менее важно, что протокол TCP, относясь к транспортному уровню и в модели OSI/ISO, и в модели TCP/IP, частично решает задачи сеансового уровня модели OSI. В частности, это задачи установки и разрыва соединения (это отмечается в учебном пособии «Анализ трафика мультисервисных сетей», см. п. 1 в доп. материалах). С другой стороны, иногда вместо того, чтобы использовать TCP, создают протоколы с собственным механизмом управления сеансом, реализуя сеансовый уровень поверх протокола UDP. Так поступили разработчики протокола QUIC (альтернатива HTTP, решающая задачи сжатия, шифрования и управления сеансом, то есть прикладного, представления и сеансового уровня, работающая поверх протокола UDP. Как несложно догадаться, используются порты 80/UDP и 443/UDP).

Практическое задание

Используйте командную строку вашей операционной системы.

В Windows запустите Win-X Powershell (администратор) или Win-R и введите **cmd**.

В GNU Linux или Mac OS используйте terminal (bash).

Запустите **netstat --help** либо **netstat --help|more** чтобы узнать перечень ключей.

В UNIX-подобных системах (GNU/Linux или Mac OS) можно использовать также **netstat --help|less** или **man netstat**.

Используйте разные варианты ключей, чтобы определить, какие у вас имеются открытые порты (TCP и UDP), какие имеются прослушивающие TCP-сокеты, установленные TCP-соединения (как исходящие, так и входящие). Выберите 10 произвольных строк и проанализируйте, какой протокол прослушивает этот сокет или установленное соединение (входящее либо исходящее). Если порт нестандартный или не зарезервирован за каким-либо протоколом, попробуйте узнать имя процесса или идентификатор процесса (чтобы узнать его в списке процессов или диспетчере задач). Узнайте, что это за сервис, и зачем ему нужны соединения. Можно искать в Интернете информацию о сервисе.

Данная задача может быть полезной на практике при поиске вредоносного ПО. Если программа запущена из домашней директории или /tmp, если она соединяется с неизвестным вам адресом, и это не используемый вами сервис, возможно, это троян или вирус.

Отчет должен содержать подробное описание 10 произвольных строк:

- что это за сокет;
- какой протокол, в каком состоянии соединение;
- какой сервис или программа его использует;
- зачем.

Отчет должен быть подготовлен в формате .docx или .pdf и сдан в качестве домашнего задания. Отчет может содержать скриншоты, но не должен состоять из одного скриншота вывода **netstat** без каких-либо пояснений.

Дополнительное усложненное задание: выбрать из приведенных примеров эхо-сервера (либо написать самостоятельно на выбранном вами языке, либо найти в Интернете), запустить на машине (можно виртуальной, или VDS, если используете).

Доступны примеры эхо-сервера:

- echo7.c – пример на Си;
- phpecho.php – пример на PHP;
- pyecho.py и pyecho2.php – примеры на Python.

Также доступны примеры на Java от нашего преподавателя **Алексея Степченко**: простой пример эхо-сервера и клиента в архиве samples.zip, более сложный — в simple_server.zip.

Используя Telnet либо PuTTY (в Windows), либо написав собственный клиент, установить одно, затем два, затем три соединения с сервером, отследить с помощью **netstat** поведение и точно также добавить в отчет.

Облегченная версия дополнительного усложненного задания: вместо эхо-сервера использовать установленный на виртуальной машине (Ubuntu) либо VDS OpenSSH-server и соединяться с ним с помощью SSH-клиента или PuTTY. Задокументировать, что происходит с сокетами при одном, двум, трех одновременных подключениях.

Дополнительные материалы

1. https://en.wikipedia.org/wiki/Network_Control_Program.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы.

2. https://en.wikipedia.org/wiki/Network_Control_Program.