Unit 5 Socket

Linux provides a flexible platform for communication between a wide range of applications and devices. Its robust networking component makes Linux a popular solution for running web servers and developing distributed systems and applications.

The Linux networking stack includes many network interfaces, protocols, and other components. Linux sockets are an important network data-transfer mechanism used to build networked applications, implement protocols and provide network services.

What are Linux Sockets?

Sockets are Linux file descriptors that serve as the communication end-points for processes running on that device. Each Linux socket consists of the device's IP address and a selected **port number**.

A socket connection is a bidirectional communication pipe that allows two processes to exchange information within a network.

What are Linux Sockets Used for

The typical sockets use case is the client-server network interaction model. In this model, the server process socket listens and waits for clients' requests. The clients exchange information with the server using **TCP/IP**, **UDP/IP** network protocols, and application-level protocols such as IMAP, POP3, SMTP, or FTP.

Although sockets primarily connect processes on a computer network, they also enable communication between processes on the same device. The same-machine connections use the IPC (Inter-process communication) sockets, also known as **Unix domain sockets**.

The IPC sockets are helpful because they eliminate the need for loopback network interfaces when applications such as Redis or MySQL use REST API to communicate with the database on the same machine.

Types of Linux Sockets

There are four main types of Linux sockets.

- **Stream-oriented sockets** provide a reliable, connection-based communication channel.
- **Datagram-oriented sockets** offer a connectionless UDP-based channel.
- **Raw sockets** allow direct access to the underlying networking protocols.
- **Sequenced packet sockets** guarantee that the sent packets will arrive in the original order.

The sections below provide details on each socket type.
Stream-Oriented Sockets

Stream-oriented sockets are mainly used in TCP/IP communication. Since TCP is a connection-oriented protocol, the stream sockets establish a **persistent connection** between two communicating processes.

The data packets sent over stream sockets arrive reliably and in the correct order, making this socket type suitable for web and email servers.

You can list the currently listening TCP sockets on Linux with the ss command: **ss –tln**

Datagram-Oriented Sockets

Datagram-oriented sockets support the connectionless UDP (User Datagram Protocol). Unlike stream sockets, datagram sockets do not establish a permanent connection between two interacting processes. Instead, **each packet is an independent datagram**, a self-contained message whose arrival or integrity is not guaranteed.

Applications usually employ datagram-oriented UDP sockets when they require low overhead or real-time communication. For example, online gaming and VoIP use UDP because they prioritize the speed of information exchange over the integrity of the information.

The following **ss** command shows the active datagram-oriented UDP sockets on a Linux system:

Ss –uln

Raw Sockets

Raw sockets allow processes to access the underlying communication protocols directly. This property enables processes to avoid transport-layer formatting associated with a protocol and exchange the data at the lowest level.

Network applications that need a high level of control over communication, such as **ping** and **traceroute**, require raw sockets to function correctly. However, since raw sockets provide easy access to the link layer, their extensive use can be a security concern.

Sequenced Sockets

Sequenced sockets allow processes to manage incoming packets at the network layer before the packets move to the transport layer. They offer a middle-ground solution between stream and datagram-oriented sockets.

- Like stream-oriented sockets, sequenced sockets are connection-oriented.
- On the other hand, they offer datagram-like delineated packet boundaries.

How Sockets Work in Linux

Each Linux socket uses a specific domain and type. The domain determines the protocol family the socket will use, such as **IPv4** or **IPv6**. The socket type specifies whether the socket will support reliable two-way communication (e.g., TCP) or one-way communication with best-effort delivery (e.g., UDP).

Sockets perform all the communication functions through the socket API. Using API calls, users can:

- Establish and manage connections with other systems.
- Obtain information about relevant network resources.
- Transfer data to and from the machine.
- Perform system functions.
- Stop the socket connections.

For example, to use a stream-oriented TCP socket in Linux, a user needs to start by specifying the type with the **socket()** system call. The next API calls differ depending on the system's role in the network.

On the **server side**:

- **bind()** - Binds a socket to a network address and port.
- **listen()** - Tells the server to wait for incoming connections to the specified network location.
- **accept()** - Receives the client connections.
- **read()** and **write()** - Communicate with the remote endpoint once the server establishes the communication and creates a new socket for the client.

On the **client side**:

- **connect()** - Connects with the server process. Takes the address of the remote server socket as an argument.
- **send()** and **recv()** - Send and receive data, respectively.
- **close()** - Ends the connection between the client and the server.
- Datagram communication, on the other hand, does not require establishing a connection. Therefore, the server and the clients use the same system calls to exchange information between the sockets.
- The **socket()** call initializes the socket. However, since there is no need to establish a permanent connection, the datagram socket does not utilize the **bind()**, **listen()**, and **accept()** calls like the stream-oriented type. Instead, the **send()** and **recv()** calls conduct packet exchange directly.

Socket attributes

The Sockets Detail attributes refer to characteristics associated with socket details, including user ID, local and foreign addresses, socket states, and socket protocols.

Foreign Address The address of the remote end of the socket. Like “netstat” * indicates that the address is unassigned/unavailable. The following values are valid: alphanumeric text strings with a maximum length of 16 characters.

Foreign Port The number of the foreign port. The following values are valid: integers. For example, the following value is valid: Value_Exceeds_Maximum=9223372036854775807.

Local Address The address of the local end of the socket, presented as a dotted ip address. The following values are valid: alphanumeric text strings with a maximum length of 16 characters.

Local Port The local port number. The following values are valid: integers. For example, the following value is valid: Value_Exceeds_Maximum=9223372036854775807.

Local Service Name The local port number translated to service name from /etc/services. The following values are valid: alphanumeric text strings with a maximum length of 64 characters.

Receive Queue (Bytes) The count of bytes not copied by the user program connected to this socket. The following values are valid: integers. For example, the following value is valid: Value_Exceeds_Maximum=9223372036854775807.

Send Queue (Bytes) The count of bytes not acknowledged by the remote host. The following values are valid: integers. For example, the following value is valid: Value_Exceeds_Maximum=9223372036854775807.

Socket Inode The inode used by the socket. The following values are valid: integers. For example, the following value is valid: Value_Exceeds_Maximum=9223372036854775807.

Socket Owner Name The user name associated with the user ID that owns or started the socket connection. The following values are valid: text strings with a maximum length of 64 bytes.

Socket Protocol Indicates the sockets using this protocol.

WebServers in Linux

1. Nginx Web server

The **Nginx web server** is renowned for its high performance and stability. It functions as a web server, reverse proxy, and load balancer, making it a versatile option for various use cases. Nginx is highly efficient in handling many concurrent connections, making it suitable for **high-traffic websites** and applications.

Unique features of NGINX:
Works as a mail proxy server.
Has a binary upgrade option for live server changes.
Supports WebSocket, allowing real-time data streaming.
Offers HTTP/HTTPS/2 support.

2. Apache HTTP Server

The **Apache HTTP Server** is the most popular web server software currently used, known for its flexibility and extensibility. It has a strong presence in the web server market, powering many websites worldwide. With a large community of developers and extensive documentation, Apache HTTP Server is an excellent choice for many web applications.

Features of Apache:
Install apache web server for a range of authentication modules.
Offers IP address-based geolocation.
URL rewriting for advanced redirect rules.
Compatible with common scripting languages.

3. Lighttpd Webserver

Lighttpd web server is a lightweight solution optimized for speed-critical environments. With a small memory footprint and low resource usage, it is ideal for applications where performance is crucial. Its modular architecture and support for FastCGI, SCGI, and CGI make it a flexible option for serving dynamic content.

Lighttpd features:
Allows secure downloads with SSI and CGI.
Uses the Event MPM model for more effective requests.
Compliant with IPv6.
Supports FLAC and MP3 streaming.

4. OpenLiteSpeed

OpenLiteSpeed web server is a high-performance, lightweight web server with built-in caching capabilities. The web server is designed to handle thousands of concurrent connections with minimal resource usage, making it an excellent choice for busy websites.
OpenLiteSpeed offers a user-friendly web administration interface, simplifying server management tasks.

Unique features:
Offers anti-DDoS features for better security.
Supports Ruby, Python, and **Node.js applications**.
Has GZIP compression for faster data transfer.
Built-in page speed optimization module.

5. H2O Web Server
H2O Web Server is a modern, optimized web server supporting cutting-edge technologies such as HTTP/2 and QUIC. It is designed to provide low latency and high throughput, making it an attractive option for performance-sensitive applications. H2O offers features like server push and connection coalescing, enabling efficient content delivery.

Features include:
Built-in support for MRuby to extend the server.
Proxy buffering to improve back-end performance.
Offers access log sampling.
Supports global rate throttling.

Databases in Linux

As the volume and complexity of data continue to grow exponentially, the demand for efficient and reliable database management systems has become paramount. Since Linux has emerged as the go-to platform for powering mission-critical applications, web services, and data-intensive projects, the importance of a Linux database has grown proportionally.

At the moment, you can choose from top Linux databases, including PostgreSQL and MySQL.

PostgreSQL stands out as a powerful and feature-rich open-source database management system. Widely known for its versatility and reliability, PostgreSQL offers an extensive feature set catering to diverse data management needs.

As an open-source solution, it ensures cost-effectiveness and allows users to access, modify, and customize the source code to fulfill project requirements.

PostgreSQL supports various data types, rich indexing options, and advanced querying capabilities resulting in efficient data organization and manipulation. Moreover, PostgreSQL's extensibility allows users to create and use custom functions, operators, and data types. As a result, developers prefer it as the application DBMA because they can easily tailor the database operations to their applications.

Advantages of PostgreSQL
Extensive Feature Set
PostgreSQL offers comprehensive features, including support for advanced data types, full-text search, JSON, and spatial data. This versatility makes it well-suited for handling diverse and complex data structures.

Highly Reliable
With a strong focus on data integrity and crash recovery mechanisms, PostgreSQL ensures that data remains consistent despite system failures. This is one critical reason behind its popularity as the DBMS for mission-critical applications.

Scalability and Performance
PostgreSQL's ability to scale vertically and horizontally, coupled with its advanced indexing and optimization techniques, allows it to handle large-scale, high-performance workloads efficiently.

Extensibility
The database's modular architecture and support for user-defined functions and data types enable developers to extend and customize PostgreSQL to meet specific business requirements.

ACID Compliance
PostgreSQL adheres to ACID (Atomicity, Consistency, Isolation, Durability) principles, ensuring transactions are reliable, predictable, and maintain data integrity.

Community Support
PostgreSQL has an active and dedicated community of developers and users who continuously contribute to the core platform improvement and improve documentation and product support.

PostgreSQL Use Cases
Here are some application areas where PostgreSQL offers huge dividends:

Web Applications
PostgreSQL is commonly used as the backend database for web applications, where it handles user profiles, content management systems, and eCommerce platforms.

Geospatial Applications
With support for geospatial data and a dedicated PostGIS extension, PostgreSQL is a popular choice for mapping, location-based services, and geographic information systems (GIS).

Data Warehousing
PostgreSQL's ability to handle complex data structures and large volumes of data makes it suitable for data warehousing and Big Data applications.

Financial Systems
PostgreSQL's ACID compliance and reliability make it a favored choice for financial applications, where data integrity and transactional consistency are critical.

Scientific Research
PostgreSQL's support for advanced data types and extensibility allows researchers to effectively manage and analyze complex scientific data.

Does PostgreSQL Comply With SQL Standards?
PostgreSQL's widespread popularity stems from its strict adherence to SQL standards. Developers can seamlessly transition to PostgreSQL using their existing SQL knowledge, ensuring smooth migration with minimal compatibility issues. This strict compliance makes PostgreSQL a familiar platform developers can quickly adopt without extensive query rewrites.

Developers can easily port applications based on PostgreSQL to other SQL-compliant database platforms without delays caused by structural rewrites and changes to application architecture.
This dedication to industry-standard SQL practices underscores its commitment to data integrity and consistency, making it a preferred choice for critical applications.

As an open-source solution, PostgreSQL's SQL standards compliance benefits the broader developer community through collaborative improvements and ensures it remains a top-tier database system for various industries.

About Dynamic Host Configuration Protocol (DHCP)

The Dynamic Host Configuration Protocol (DHCP) is a network service that enables host computers to be automatically assigned settings from a server as opposed to manually configuring each network host. Computers configured to be DHCP clients have no control over the settings they receive from the DHCP server, and the configuration is transparent to the computer's user.

The most common settings provided by a DHCP server to DHCP clients include:
IP address and netmask
IP address of the default-gateway to use
IP addresses of the DNS servers to use
However, a DHCP server can also supply configuration properties such as:
Hostname
Domain name
Time server
Print server

The advantage of using DHCP is that any changes to the network, such as a change in the DNS server address, only need to be changed at the DHCP server, and all network hosts will be reconfigured the next time their DHCP clients poll the DHCP server. As an added advantage, it is also easier to integrate new computers into the network, as there is no need to check for the availability of an IP address. Conflicts in IP address allocation are also reduced.

DHCP configuration
A DHCP server can provide configuration settings using the following methods:

Manual allocation (MAC address)
This method uses DHCP to identify the unique hardware address of each network card connected to the network, and then supplies a static configuration each time the DHCP client makes a request to the DHCP server using that network device. This ensures that a particular address is assigned automatically to that network card, based on its MAC address.

Dynamic allocation (address pool)
In this method, the DHCP server assigns an IP address from a pool of addresses (sometimes also called a range or scope) for a period of time (known as a lease) configured on the server, or until the client informs the server that it doesn't need the address anymore. This way, the clients receive their configuration properties dynamically and on a "first come, first served" basis. When a DHCP client is no longer on the network for a specified period, the configuration is expired and released back to the address pool for use by other DHCP clients. After the lease period expires, the client must renegotiate the lease with the server to maintain use of the same address.

Automatic allocation
Using this method, the DHCP automatically assigns an IP address permanently to a device, selecting it from a pool of available addresses. Usually, DHCP is used to assign a temporary address to a client, but a DHCP server can allow an infinite lease time.
The last two methods can be considered "automatic" because in each case the DHCP server assigns an address with no extra intervention needed. The only difference between them is in how long the IP address is leased; in other words, whether a client's address varies over time.

Available servers
Ubuntu makes two DHCP servers available:
isc-dhcp-server:
This server installs dhcpd, the dynamic host configuration protocol daemon. Although Ubuntu still supports isc-dhcp-server, this software is no longer supported by its vendor.
Find out how to install and configure isc-dhcp-server.
isc-kea:
Kea was created by ISC to replace isc-dhcp-server – It is supported in Ubuntu releases from 23.04 onwards.

DNS in Linux

The Domain Name System (DNS) is used to resolve (translate) hostnames to internet protocol (IP) addresses and vice versa. A DNS server, also known as a nameserver, maps IP addresses to hostnames or domain names .

Domain Name Service (DNS) is an Internet service that maps IP addresses and fully qualified domain names (FQDN) to one another. In this way, DNS alleviates the need to remember IP addresses. Computers that run DNS are called *name servers*. Ubuntu ships with BIND (Berkley Internet Naming Daemon), the most common program used for maintaining a name server on Linux.

What Is DNS?

DNS is a global directory of computing resources. It can locate servers, PCs, phones, IoT devices, routers, switches, and essentially anything with an IP address. DNS also holds information linking computers to security keys, services such as instant messaging, GPS coordinates, and much more.

DNS design is *hierarchical* and *distributed*. The diagram below illustrates both points. It shows how DNS divides all resources into domains, and delegates different levels of name servers to provide information within each domain.

At the top of the DNS tree are the root servers, usually referred to by a single dot ("."). The root servers are actually comprised of 13 different sets of servers, each operated by a different organization. They act as a starting point for all other parts of the hierarchy.

Below the root servers are top-level domains (TLDs). Common TLDs include .com, .edu, and .org, but there are currently around 1,500 TLDs.
Further down in the hierarchy are second- and third-level domains, and so on, referring to their distance from the root. The domain example.com is an example of a second-level domain, while engineering.example.com is a third-level domain.
Regardless of level, all DNS name servers provide information about items within their domains. For example, linode.com name servers provide information about hostnames, addresses, mail servers, and name servers within that domain.

How Does DNS Work?

DNS clients use recursion, the concept of repeating a process multiple times using information gleaned from the previous step, to learn how to locate resources.

For example, when visiting www.linode.com, your system first conducts a series of recursive DNS exchanges to learn the server's IP address before it can connect with the Linode web server.

First, your client checks to see if it has an IP address for the Linode webserver in its local cache. If not, it asks your local DNS server.

If the local DNS server doesn't have an IP address in its cache, it then asks one of the root servers how to reach www.linode.com. The root server replies, essentially saying "I don't know, but I see you're asking about something in the .com domain. Here are some IP addresses for .com name servers".
Your local DNS server then repeats its query, this time to a .com name server. The .com name server replies, "I don't know, but I can tell you the IP addresses of name servers for linode.com".

The local DNS server then asks, for the third time, what IP addresses correspond to the hostname www.linode.com. This time, the linode.com name servers are authorized to provide a direct answer to your question.

A Linode name server responds with IP addresses for the Linode web server. Your local DNS server passes along that information to your client, and caches it for future use.

Finally, your system has an IP address for the Linode web server. It can now set up a TCP connection and make an HTTP request.

Before you could retrieve the web page, your client and the local DNS server had multiple conversations with three outside sets of DNS servers, plus your local DNS server, just to get an IP address.

Believe it or not, that's a relatively simple example. A popular commercial site, such as a news site, has advertising and other embedded third-party content. This could require having these sets of DNS conversations several dozen times, or more, just to load a single page.

While all this occurs in milliseconds, it underscores how critical DNS is to a functional Internet. Everything begins with a series of DNS queries.

Types of DNS Servers

There are several types of DNS servers: *authoritative* (both primary and secondary), *forwarding*, and *recursive*. All three types may reside on a single server, or may be assigned to separate dedicated servers for each type. Regardless of location, these servers work cooperatively to serve DNS information.

An authoritative server provides definitive responses for a given zone, which is a set of computing resources under common administrative control. While non-authoritative DNS servers may also store and forward answers about a zone's contents, only an authoritative server can provide definitive answers.

A zone may be a domain such as linode.com, but there isn't always a 1:1 relationship between domains and zones. Authoritative servers may delegate authority to subdomains. For example, the .com authoritative servers delegate responsibility for linode.com to Linode's name servers. In this case, linode.com is part of the .com domain, but .com and linode.com represent different zones.

Each zone should have at least two authoritative name servers, for both redundancy and load-balancing of user queries. This guide covers setup of a primary name server.

Both primary and secondary name servers are authoritative, and appear nearly identical to the outside world. However, secondary name servers automatically receive all zone updates from the primary name server. After initial setup, zone configuration changes only need to be made once, on the primary name server. It then automatically pushes out all updates to any secondary name servers.

A forwarding server is one that makes external DNS requests on behalf of clients or other DNS servers. Your local DNS server, while not authoritative for most zones, is most likely either a forwarding server itself, or configured to send requests to one. When you ask for a DNS resource outside your zone, a forwarding server finds that resource.

A recursive resolver makes DNS queries, either for itself or on behalf of other clients, and optionally caches the replies. As with any type of caching, it's best to store information as close as possible to your client. While your local client may perform recursion, your local DNS server almost certainly does.

By far the most common DNS server software is Bind 9, the open source package maintained by the Internet Systems Consortium. Bind can function as any type of DNS server. It also supports many DNS extensions that go well beyond it's original design goals.