Socket

Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix **file descriptors**. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else.

To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Sockets were first introduced in 2.1BSD and subsequently refined into their current form with 4.2BSD. The sockets feature is now available with most current UNIX system releases.

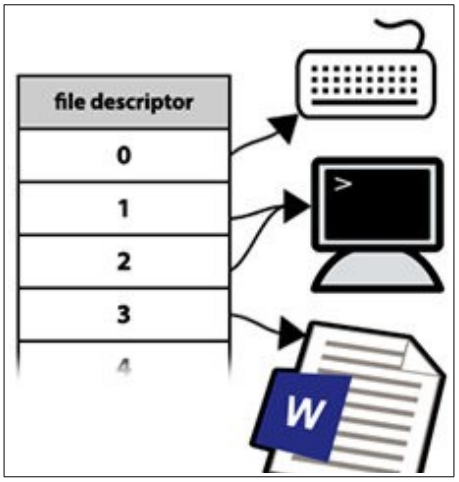
File descriptor

A **file descriptor** is a number that uniquely identifies an open file in a computer's operating system. It describes a data resource, and how that resource may be accessed.

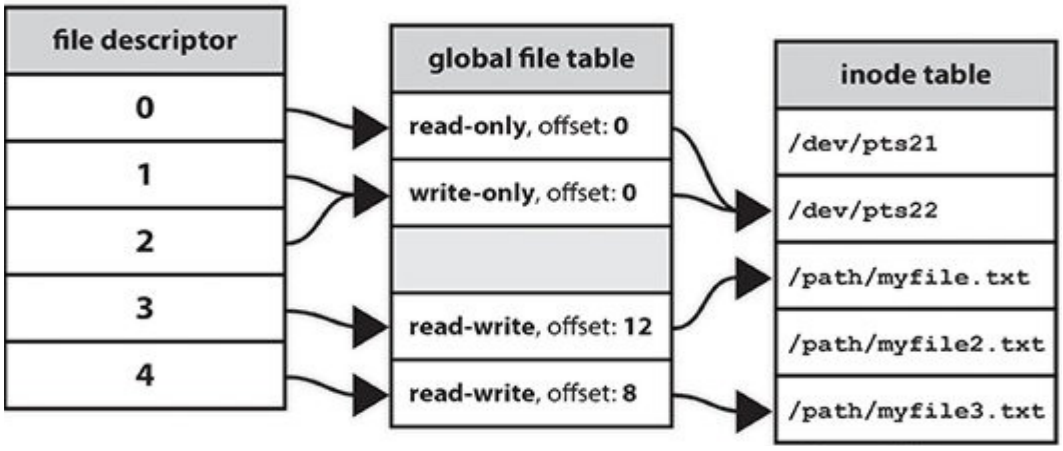
When a program asks to open a file or another data resource, like a network socket the kernel of the operating system grants access, makes an entry in the **global file table**, and provides the software with the location of that entry.

The descriptor is identified by a unique non-negative integer, such as **0**, **12**, or **567**. At least one file descriptor exists for every open file on the system.

File descriptors were first used in Unix, and are used by modern operating systems including Linux, macOS X, and BSD



When a process makes a successful request to open a file, the kernel returns a file descriptor which points to an entry in the kernel's **global file table**. The file table entry contains information such as the **inode** of the file, **byte offset**, and the **access restrictions** for that data stream (read-only, write-only, etc.).



List all Open Files with lsof Command

n the below example, it will show long listing of open files some of them are extracted for better understanding which displays the columns like **Command**, **PID**, **USER**, **FD**, **TYPE** etc

```
[ali@fedora29 ~]$ lsof -u ali
COMMAND  PID   USER   FD   TYPE  DEVICE  SIZE/OFF     NODE   NAME
bash            6765   ali      cwd    DIR   253,2      4096        10616833   /home/ali
bash            6765   ali      rtd    DIR   253,0      4096               2  /
bash            6765   ali      txt    REG   253,0     1190216     2359589  /usr/bin/bash
bash            6765   ali      mem  REG   253,0   217749968   2359350 /usr/lib/locale/locale-archive
bash            6765   ali      mem  REG   253,0     8406312    2626254 /var/lib/sss/mc/passwd
bash           6765    ali      mem  REG   253,0       47648    2381517 /usr/lib64/libnss_sss.so.2
bash           6765    ali      mem REG   253,0      2786472     2373176 /usr/lib64/libc-2.28.so
bash          6765    ali      mem REG   253,0      29320       2376250 /usr/lib64/libdl-2.28.so
bash          6765  ali        mem REG   253,0      205480      2374029 /usr/lib64/libtinfo.so.6.1
man           6829   ali        1u   CHR   136,0      0t0                3 /dev/pts/0
```

Sections and it's values are self-explanatory. However, we'll review **FD & TYPE** columns more precisely.

FD – stands for File descriptor and may seen some of the values as:

1. **cwd** current working directory
2. **rtd** root directory
3. **txt** program text (code and data)
4. **mem** memory-mapped file

Also in **FD** column numbers like **1u** is actual file descriptor and followed by **u,r,w** of it's mode as:

Find Processes running on Specific Port

To find out all the running process of specific port, just use the following command with option **-i**. The below example will list all running process of port 443

```
[hayder@fedora29 ~]$ lsof -i TCP:443
COMMAND    PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
firefox   5118 hayder   94u  IPv4 282104      0t0  TCP fedora29.hayder:35098->ec2-54-149-0-100.us-west-2.compute.amazonaws.com:https (ESTABLISHED)
firefox   5118 hayder  100u  IPv4 326715      0t0  TCP fedora29.hayder:52610->sof02s33-in-f7.1e100.net:https (ESTABLISHED)
firefox   5118 hayder  162u  IPv4 326758      0t0  TCP fedora29.hayder:47472->sof02s27-in-f14.1e100.net:https (ESTABLISHED)
firefox   5118 hayder  200u  IPv4 254216      0t0  TCP fedora29.hayder:36580->wl-in-f189.1e100.net:https (ESTABLISHED)
firefox   5118 hayder  243u  IPv4 250623      0t0  TCP fedora29.hayder:40572->sof02s27-in-f5.1e100.net:https (ESTABLISHED)
firefox   5118 hayder  248u  IPv4 251676      0t0  TCP fedora29.hayder:44822->wn-in-f189.1e100.net:https (ESTABLISHED)
firefox   5118 hayder  257u  IPv4 251686      0t0  TCP fedora29.hayder:36250->sof02s22-in-f195.1e100.net:https (ESTABLISHED)
chrome-gn 5425 hayder   78u  IPv4  93629      0t0  TCP fedora29.hayder:36558->sof02s28-in-f10.1e100.net:https (CLOSE_WAIT)
```

Where is Socket Used?

A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

Socket Types

There are four types of sockets available to the users. The first two are most commonly used and the last two are rarely used.

Processes are presumed to communicate only between sockets of the same type but there is no restriction that prevents communication between sockets of different types.

- **Stream Sockets** − Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.
- **Datagram Sockets** − Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).
- **Raw Sockets** − These provide users access to the underlying communication protocols, which support socket abstractions. These sockets are normally datagram oriented, though their exact characteristics are dependent on the interface provided by the protocol. Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.
- **Sequenced Packet Sockets** − They are similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to

be used with all outgoing data, and allows the user to receive the headers on incoming packets.

Network Addresses

The IP host address, or more commonly just IP address, is used to identify hosts connected to the Internet. IP stands for Internet Protocol and refers to the Internet Layer of the overall network architecture of the Internet.

An IP address is a 32-bit quantity interpreted as four 8-bit numbers or octets. Each IP address uniquely identifies the participating user network, the host on the network, and the class of the user network.

An IP address is usually written in a dotted-decimal notation of the form N1.N2.N3.N4, where each Ni is a decimal number between 0 and 255 decimal (00 through FF hexadecimal).

Address Classes

IP addresses are managed and created by the *Internet Assigned Numbers Authority* (IANA). There are five different address classes. You can determine which class an IP address is in by examining the first four bits of the IP address.

- **Class A** addresses begin with **0xxx**, or **1 to 126** decimal.
- **Class B** addresses begin with **10xx**, or **128 to 191** decimal.
- **Class C** addresses begin with **110x**, or **192 to 223** decimal.
- **Class D** addresses begin with **1110**, or **224 to 239** decimal.
- **Class E** addresses begin with **1111**, or **240 to 254** decimal.

Addresses beginning with **01111111**, or **127** decimal, are reserved for **loopback** and for internal testing on a local machine [You can test this: you should always be able to ping **127.0.0.1**, which points to yourself ; Class D addresses are reserved for **multicasting**;

Class E addresses are reserved for future use. They should not be used for host addresses.

Client Process

This is the process, which typically makes a request for information. After getting the response, this process may terminate or may do some other processing.

Example, Internet Browser works as a client application, which sends a request to the Web Server to get one HTML webpage.

Server Process

This is the process which takes a request from the clients. After getting a request from the client, this process will perform the required processing, gather the requested information, and send it to the requestor client. Once done, it becomes ready to serve another client. Server processes are always alert and ready to serve incoming requests.

Example − Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.

Note that the client needs to know the address of the server, but the server does not need to know the address or even the existence of the client prior to the connection being established. Once a connection is established, both sides can send and receive information.

2-tier and 3-tier architectures

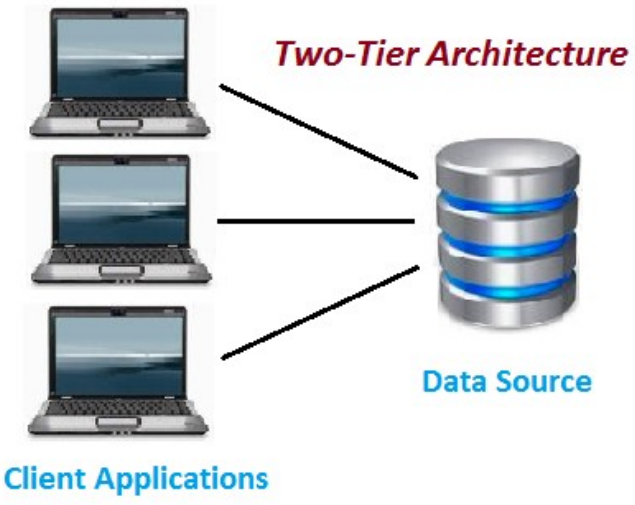
There are two types of client-server architectures −

- **2-tier architecture** − In this architecture, the client directly interacts with the server. This type of architecture may have some security holes and performance problems. Internet Explorer and Web Server work on two-tier architecture. Here security problems are resolved using Secure Socket Layer (SSL).

- **3-tier architectures** − In this architecture, one more software sits in between the client and the server. This middle software is called ‘middleware’. Middleware are used to perform all the security checks and load balancing in case of heavy load. A middleware takes all requests from the client and after performing the required authentication, it passes that request to the server. Then the server does the required processing and sends the response back to the middleware and finally the middleware passes this response back to the client. If you want to implement a 3-tier architecture, then you can keep any middleware like Web Logic or WebSphere software in between your Web Server and Web Browser.

Two-Tier Architecture:

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.



The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.

The Two-tier architecture is divided into two parts:

1) Client Application (Client Tier)
2) Database (Data Tier)

Advantages:

1. Easy to maintain and modification is bit easy
2. Communication is faster

Disadvantages:

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective

Three-Tier Architecture:

Three-tier architecture typically comprise a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows:

1) Client layer
2) Business layer
3) Data layer

1) Client layer:

It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

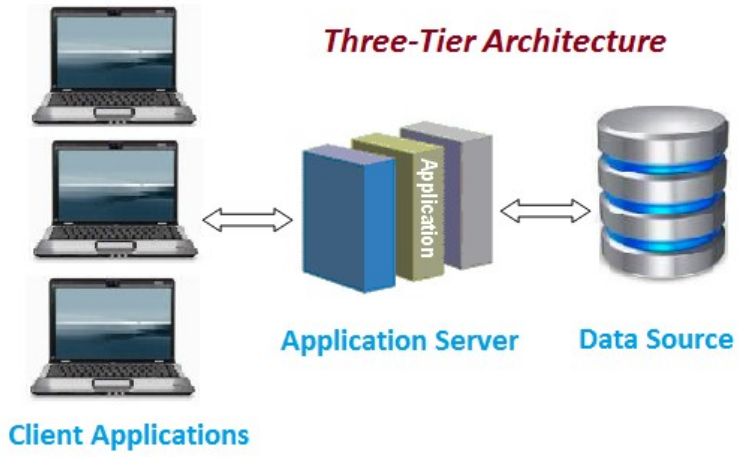
2) Business layer:

In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is

also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.



Advantages

1. High performance, lightweight persistent objects
2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration

5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won't affect other modules
9. In three tier architecture application performance is good.

Disadvantages

1. Increase Complexity/Effort

Socket - Ports and Services

When a client process wants to a connect a server, the client must have a way of identifying the server that it wants to connect. If the client knows the 32-bit Internet address of the host on which the server resides, it can contact that host. But how does the client identify the particular server process running on that host?

To resolve the problem of identifying a particular server process running on a host, both TCP and UDP have defined a group of well-known ports.

For our purpose, a port will be defined as an integer number between 1024 and 65535. This is because all port numbers smaller than 1024 are considered *well-known* -- for **example, telnet uses port 23, http uses 80, ftp uses 21, and so on.**

The port assignments to network services can be found in the file **/etc/services.** If you are writing your own server then care must be taken to assign a port to your server. You should make sure that this port should not be assigned to any other server.

Normally it is a practice to assign any port number more than 5000. But there are many organizations who have written servers having port numbers more than 5000. For example, Yahoo Messenger runs on 5050, SIP Server runs on 5060, etc.

Example Ports and Services

Here is a small list of services and associated ports. You can find the most updated list of internet ports and associated service at IANA - TCP/IP Port Assignments

Service	Port Number	Service Description
echo	7	UDP/TCP sends back what it receives.
discard	9	UDP/TCP throws away input.
daytime	13	UDP/TCP returns ASCII time.
chargen	19	UDP/TCP returns characters.
ftp	21	TCP file transfer.
telnet	23	TCP remote login.
smtp	25	TCP email.
daytime	37	UDP/TCP returns binary time.
tftp	69	UDP trivial file transfer.
finger	79	TCP info on users.
http	80	TCP World Wide Web.
login	513	TCP remote login.
who	513	UDP different info on users.
Xserver	6000	TCP X windows (N.B. >1023).