LINUX - SOCKET - DOCUMENTATION

Preface: useful macros

Here are some useful macros from my go-to socket demo files I wrote for myself, which make it clear how to get UDP vs TCP packets.

From these files:

```
socket__geeksforgeeks_udp_server_GS_edit_GREAT.c
socket__geeksforgeeks_udp_client_GS_edit_GREAT.c
```

...you can find these macros:

```
// See: https://linux.die.net/man/7/ip
// AF = "Address Family"
// INET = "Internet"
// AF_INET = IPv4 internet protocols
// AF_INET6 = IPv6 internet protocols; see: https://linux.die.net/man/2/socket
// DGRAM = "Datagram" (UDP)
//
// IPv4
#define SOCKET_TYPE_TCP_IPV4              AF_INET, SOCK_STREAM, 0
#define SOCKET_TYPE_UDP_IPV4              AF_INET, SOCK_DGRAM, 0
#define SOCKET_TYPE_RAW_IPV4(protocol)    AF_INET, SOCK_RAW, (protocol)
// IPv6
#define SOCKET_TYPE_TCP_IPV6              AF_INET6, SOCK_STREAM, 0
#define SOCKET_TYPE_UDP_IPV6              AF_INET6, SOCK_DGRAM, 0
#define SOCKET_TYPE_RAW_IPV6(protocol)    AF_INET6, SOCK_RAW, (protocol)
```

Usage examples:

```
int socket_tcp = socket(SOCKET_TYPE_TCP_IPV4);
int socket_udp = socket(SOCKET_TYPE_UDP_IPV4);
// See also: https://www.binarytides.com/raw-sockets-c-code-linux/
int socket_raw = socket(SOCKET_TYPE_RAW_IPV4(IPPROTO_RAW));
```

Now back to the question:

> What is SOCK_DGRAM and SOCK_STREAM?

Brief Summary

UDP --(is the protocol utilized by)--> AF_INET, SOCK_DGRAM or AF_INET6, SOCK_DGRAM
TCP --(is the protocol utilized by)--> AF_INET, SOCK_STREAM or AF_INET6, SOCK_STREAM

Examples: from https://linux.die.net/man/7/ip (or as shown in your terminal man pages by running man 7 ip):

> tcp_socket = `socket(AF_INET, SOCK_STREAM, 0);`
>
> udp_socket = `socket(AF_INET, SOCK_DGRAM, 0);`
>
> raw_socket = `socket(AF_INET, SOCK_RAW, protocol);`

Long Summary

Reference the int socket(AddressFamily, Type, Protocol) socket creation function documentation here and here (can also be viewed by running man 2 socket). It allows you to specify these 3 parameters:

1. Address Family
2. Socket Type
3. Protocol

For many if not most use-cases, however, the most-useful options for these parameters are frequently:

1. Address Family: `AF_INET` (for IPv4 addresses) or `AF_INET6` (for IPv6 addresses).
2. Socket Type: `SOCK_DGRAM` or `SOCK_STREAM`.
3. Protocol: just use 0 to allow it to use default protocols, as specified from the documentation link above (emphasis added):

   > Protocol: Specifies a particular protocol to be used with the socket.

   > **Specifying the Protocol parameter of `0` causes the socket subroutine to default to the typical protocol for the requested type of returned socket.**

4. `SOCK_DGRAM`: if you create your socket with `AF_INET` as

   ```
   int s = socket(AF_INET, SOCK_DGRAM, 0)
   ```

   or with `AF_INET6` as

   ```
   int s = socket(AF_INET6, SOCK_DGRAM, 0)
   ```

   ...the socket utilizes the UDP protocol by default when the (`AF_INET` or `AF_INET6`) address family and `SOCK_DGRAM` Socket Types are selected.
   1. In the **UNIX Address Family domain** (`AF_UNIX`): when communicating between processes running on the same operating system via the AF_UNIX Address Family, this is similar to an inter-process message queue.

   2. In the **Internet Address Family domain** (`AF_INET` and `AF_INET6`): when communicating between a local process and a process running on a remote host via the `AF_INET` Address Family, this is "implemented on the User Datagram Protocol/Internet Protocol (UDP/IP) protocol."

5. `SOCK_STREAM`: if you create your socket with `AF_INET` as

   ```
   int s = `socket(AF_INET, SOCK_STREAM, 0)`
   ```

   or with `AF_INET6` as

   ```
   int s = `socket(AF_INET6, SOCK_STREAM, 0)`
   ```

   ...the socket utilizes the TCP protocol by default when the (AF_INET or AF_INET6) address family and SOCK_STREAM Socket Types are selected.

   1. In the UNIX Address Family domain (AF_UNIX): when communicating between processes running on the same operating system via the AF_UNIX Address Family, this type of socket "works like a pipe" IPC (Inter-process Communication) mechanism.

   2. In the Internet Address Family domain (AF_INET and AF_INET6): when communicating between a local process and a process running on a remote host via the AF_INET Address Family, this is "implemented on the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol."

Details

In the explanation below, wherever I (or they, in the quoted sections) use `AF_INET` (for IPv4 addresses), keep in mind you can also use `AF_INET6` (for IPv6 addresses) if you like.

In socket-based communication, including for sending both UDP/IP and TCP/IP Ethernet data packets back and forth between two running processes on the same computer, or between two separate computers, you must specify both the Address Family (these constants begin with AF_) and Socket Type (these constans begin with SOCK_).

The best documentation I have found on sockets, hands-down, is from IBM.com, such as here:

1. `int socket(AddressFamily, Type, Protocol)` function: https://www.ibm.com/docs/en/aix/7.1?topic=s-socket-subroutine
2. Address Families: https://www.ibm.com/docs/en/aix/7.1?topic=domains-address-families and here
3. Socket Types: https://www.ibm.com/docs/en/aix/7.1?topic=protocols-socket-types

For additional information on "Sockets", click the links in the left-hand navigation pane after clicking one of the links above.

Other excellent documentation can also be found on linux.die.net, such as the ip(7) page here.

Address Family (`AF_`) Domains

From the "Address Families" link above, first, we learn about the various **socket Address Families (AF) domains**, which are a prerequisite to understanding the socket types. Here is that information (emphasis added, and my notes added in square brackets []):

> A socket subroutine that takes an address family (AF) as a parameter can use **AF_UNIX** (UNIX), **AF_INET** (Internet), **AF_NS** (Xerox Network Systems), or **AF_NDD** (Network Device Drivers of the operating sytem) protocol. These address families are part of the following communication domains:

> **UNIX**: Provides socket communication between processes running on the same operating system when an address family of **AF_UNIX** is specified. A socket name in the UNIX domain is a string of ASCII characters whose maximum length depends on the machine in use.

> **Internet**: Provides socket communication between a local process and a process running on a remote host when an address family of **AF_INET** is specified. The Internet domain requires that Transmission Control Protocol/Internet Protocol (**TCP/IP**) be installed on your system. A socket name in the Internet domain is an Internet address, made up of a 32-bit IP address and a 16-bit port address .

> **NDD**: Provides socket communication between a local process and a process running on a remote host when an address family of AF_NDD is specified. The NDD domain enables applications to run directly on top of physical networks. This is in contrast to the Internet domain, in which applications run on top of transport protocols such as TCP, or User Datagram Protocol (UDP). A socket name in the NDD domain consists of operating system NDD name and a second part that is protocol dependent.

> *Communication domain*s are described by a domain data structure that is loadable. Communication protocols within a domain are described by a structure that is defined within the system for each protocol implementation configured. When a request is made to create a socket, the system uses the name of the communication domain to search linearly the list of configured domains. If the domain is found, the domain's table of supported protocols is consulted for a protocol appropriate for the type of socket being created or for a specific protocol request. (A wildcard entry may exist for a raw domain.) Should multiple protocol entries satisfy the request, the first is selected.*

Socket Types (`SOCK_`)

From the "Socket Types" link above, we learn about the various "underlying communication protocols" (emphasis added, and my notes added in square brackets []):

> Sockets are classified according to communication properties. Processes usually communicate between sockets of the same type. However, if the underlying communication protocols support the communication, sockets of different types can communicate.
>
> Each socket has an associated type, which describes the semantics of communications using that socket. The socket type determines the socket communication properties such as reliability, ordering, and prevention of duplication of messages. The basic set of socket types is defined in the sys/socket.h file:

```
/*Standard socket types */ \
#define  SOCK_STREAM            1 /*virtual circuit*/ \
#define  SOCK_DGRAM             2 /*datagram*/ \
#define  SOCK_RAW               3 /*raw socket*/ \
#define  SOCK_RDM               4 /*reliably-delivered message*/ \
#define  SOCK_CONN_DGRAM        5 /*connection datagram*/ \
```

> Other socket types can be defined.
> The operating system supports the following basic set of sockets:
>
> `SOCK_DGRAM`: Provides datagrams, which are connectionless messages of a fixed maximum length. This type of socket is generally used for short messages, such as a name server or time server, because the order and reliability of message delivery is not guaranteed.
>
> **In the *UNIX domain* , the `SOCK_DGRAM` socket type is similar to a message queue. In the Internet domain , the `SOCK_DGRAM` socket type is implemented on the User Datagram Protocol/Internet Protocol (UDP/IP) protocol.**
>
> A datagram socket supports the bidirectional flow of data, which is not sequenced, reliable, or unduplicated. A process receiving messages on a datagram socket may find messages duplicated or in an order different than the order sent. Record boundaries in data, however, are preserved. Datagram sockets closely model the facilities found in many contemporary packet-switched networks.

> `SOCK_STREAM`: Provides sequenced, two-way byte streams with a transmission mechanism for stream data. This socket type transmits data on a reliable basis, in order, and with out-of-band capabilities.
>
> In the UNIX domain , the `SOCK_STREAM` socket type works like a pipe. In the Internet domain , the SOCK_STREAM socket type is implemented on the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol.
>
> A stream socket provides for the bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries. Aside from the bidirectionality of data flow, a pair of connected stream sockets provides an interface nearly identical to pipes.
>
> `SOCK_RAW`: Provides access to internal network protocols and interfaces. This type of socket is available only to users with root-user authority, or to non-root users who have the `CAP_NUMA_ATTACH` capability. (For non-root raw socket access, the chuser command assigns the `CAP_NUMA_ATTACH` capability, along with `CAP_PROPAGATE`. For further information, refer to the `chuser` command.)
>
> Raw sockets allow an application to have direct access to lower-level communication protocols. Raw sockets are intended for advanced users who want to take advantage of some protocol feature that is not directly accessible through a normal interface, or who want to build new protocols on top of existing low-level protocols.
>
> Raw sockets are normally datagram-oriented, though their exact characteristics are dependent on the interface provided by the protocol.
>
> `SOCK_SEQPACKET`: Provides sequenced, reliable, and unduplicated flow of information.
>
> `SOCK_CONN_DGRAM`: Provides connection-oriented datagram service. This type of socket supports the bidirectional flow of data, which is sequenced and unduplicated, but is not reliable. Because this is a connection-oriented service, the socket must be connected prior to data transfer. Currently, only the Asynchronous Transfer Mode (ATM) protocol in the Network Device Driver (NDD) domain supports this socket type.

How do they work?

> The `SOCK_DGRAM` and `SOCK_RAW` socket types allow an application program to send datagrams to correspondents named in send subroutines. The Protocol parameter is important when using the `SOCK_RAW` socket type to communicate with low-level protocols or hardware interfaces. The application program must specify the address family in which the communication takes place.

This is the **general sequence of function calls required to communicate using `SOCK_STREAM (TCP` protocol) socket types:**

> The SOCK_STREAM socket types are full-duplex byte streams. A stream socket must be connected before any data can be sent or received on it. When using a stream socket for data transfer, an application program needs to perform the following sequence:
>
> 1. Create a connection to another socket with the connect subroutine.
> 2. Use the read and write subroutines or the send and recv subroutines to transfer data.
> 3. Use the close subroutine to finish the session.
>
> An application program can use the `send` and `recv` subroutines to manage out-of-band data.
>
> Possible errors returned or set in the `errno` variable when using `SOCK_STREAM`:

> `SOCK_STREAM` communication protocols are designed to prevent the loss or duplication of data. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable period of time, the connection is broken. When this occurs, the socket subroutine indicates an error with a return value of `-1` and the `errno` global variable is set to `ETIMEDOUT`. If a process sends on a broken stream, a `SIGPIPE` signal is raised. Processes that cannot handle the signal terminate. When out-of-band data arrives on a socket, a `SIGURG` signal is sent to the process group.
>
> The process group associated with a socket can be read or set by either the `SIOCGPGRP` or `SIOCSPGRP` `ioctl` operation. To receive a signal on any data, use both the `SIOCSPGRP` and `FIOASYNC` `ioctl` operations. These operations are defined in the sys/ioctl.h file.

That about covers it. I hope to write some basic demos in my eRCaGuy_hello_world repo in the c dir soon.

Main References:

1. [my answer] What does the number in parentheses shown after Unix command names in manpages mean?
2. https://linux.die.net/man/7/ip
3. https://linux.die.net/man/2/socket
4. https://linux.die.net/man/7/tcp
5. Titre lien externe
6. `int socket(AddressFamily, Type, Protocol)` function: https://www.ibm.com/docs/en/aix/7.1?topic=s-socket-subroutine
7. Address Families: https://www.ibm.com/docs/en/aix/7.1?topic=domains-address-families and here
8. Socket Types: https://www.ibm.com/docs/en/aix/7.1?topic=protocols-socket-types

Related:

1. What is SOCK_DGRAM and SOCK_STREAM?
2. when is IPPROTO_UDP required?
3. IPPROTO_IP vs IPPROTO_TCP/IPPROTO_UDP