Important Points regarding Socket Programming

What exactly *is* a socket?

Sockets allow communication between two different processes on the same or different machines. In simpler words, sockets are simply a mechanism using which two computers can easily talk to each other.

At the programming level, a socket simply looks like a file descriptor, and we can perform read and write operations over this file descriptor.

A file descriptor is just and integer associated with an open file that identifies a file, and has the ability to perform I/O operations on the file (like `read()`, `write()`, etc.).

These read/write operations are carried over the network connecting the two computers. It creates an endpoint on both ends (client and server) to send and receive information over the network.

Stream VS Datagram sockets

Since there are two popular ways in which we can send packets over a network (TCP and UDP), it makes sense that there are two different types of sockets - one for each mechanism.

Stream Sockets -

Stream sockets are those which correspond to the TCP/IP (connection-oriented) protocol. They ensure that the data sent is received error free, and in the same order in which it was sent. Thus, this is a reliable method of data transfer.

In order to ensure the order, the stream sockets first need to establish a TCP connection with each other. Only then can they begin packet transfer. They are used in file transfer applications where the loss of data is strictly not permitted.

Datagram Sockets -

Datagram sockets are those which correspond to the UDP (connectionless) protocol. The packets transferred through this mechanism may or may not reach the receiver. Thus, this is an unreliable method of data transfer. But if they reach the receiver, they will be error free.

These sockets do not need to explicitly establish a connection with each other. They begin packet transfer right away. They are used for streaming audio/video data like in video conferencing wherein the loss of some packets is still tolerable.

Endianness-

We need to be careful regarding the endianness of the data we are sending through the sockets. There are two types of byte orders that different processors (hosts) follow.

Little Endian Byte Order:

Here, the least significant byte (LSB) is placed at the byte with the lowest address. Very common, used mostly in Intel/AMD or other x86 compatible processors.

Big Endian Byte Order:

Here, the most significant byte (MSB) is placed at the byte with the lowest address. Much rare, used in Motorola processors.

```
**NOTE:**  Intel, AMD and other x86 and x86-64 (AMD64) processors use the little-endian format to store bytes. However, for socket programming, the network byte order is big endian.
```

This means that if our system is `little endian` (which it usually is), we need to convert our endianness from `little endian` to `big endian` to transfer packets to and from the network.

Thus, in order to send packets over the network, we need to use the apis `htons` (host-to-network-short) to convert from `little endian` (host byte order) to `big endian` (network byte order).

Similarly, while receiving packets from the network, we need to use the apis `ntohs` (network-to-host-short) to convert from `big endian` (network byte order) to `little endian` (host byte order).

References-

1. https://youtu.be/bqj4dWG7v3c
2. https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm
3. https://www.geeksforgeeks.org/socket-programming-cc/