TCP/IP Sockets – 1

Monday, February 14, 2022 3:26 PM

Open file – it can be a network connection, a FIFO, a pipe, a terminal and so on.

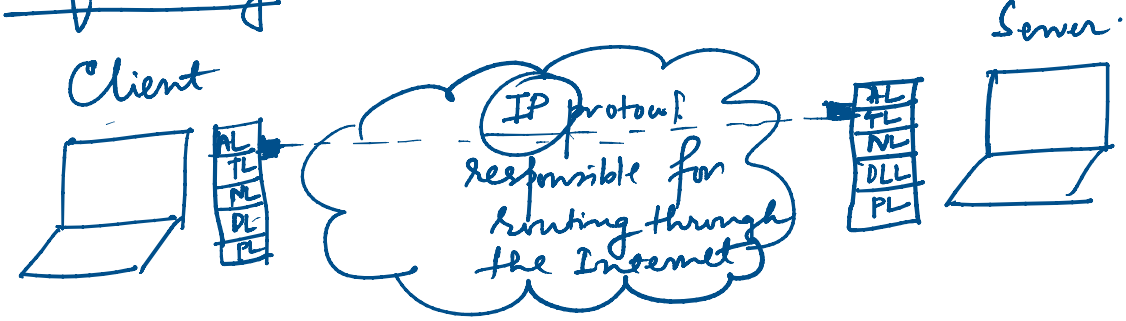
Socket – an integer associated with an open file.

UDP segment header – Src port, Dst port

TCP segment header – Src IP address, Src port, Dst IP address, Dst port.

(1) Stream sockets – uses TCP to ensure that the data arrives sequentially and error-free and IP for Internet routing.

→ SOCK_STREAM.



(2) Datagram sockets: uses UDP for data delivery and IP for Internet routing.

– Unreliable: the datagram which is sent may arrive out-of-order

– Connectionless: don't have to maintain an open connection as with stream sockets – just build a segment,

- Connectionless: as with stream sockets — just build a segment, put an IP header (datagram/packet) on it with the destination information and send it out — no connection is needed.

Byte order.

The order in which bytes are stored in the memory can be different and is an attributes of the processor.

Two common ordering systems:

(1) Little-Endian: Low-order byte is stored at a lower address. This is also called Intel order since Intel's x86 family of CPUs popularized this ordering. Intel, AMD, VAX etc. are little endian systems.

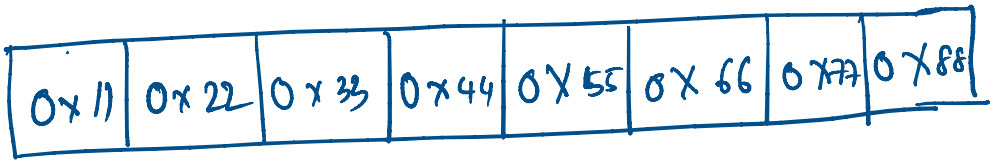
0	1	2	3	4	5	6	7
Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7

Address Byte. || Little Endian System.

(2) Big-Endian: High-order byte is stored at a lower address. This is also called Motorola order since Motorola's Power PC architecture used this ordering. Motorola 68K and IBM Mainframes are big-endian systems.

Address	0	1	2	3	4	5	6	7
Bytes	Byte7	Byte6	Byte5	Byte4	Byte3	Byte2	Byte1	Byte0

Memory Content:



64-bit value on Little Endian:

0X8877665544332211

least significant byte ("octet") first

↳ Host Byte Ordering (locally)

64-bit value on Big Endian:

0X1122334455667788

most significant byte ("octet") first

↳ Network Byte Ordering (Over the network)

To allow machines with different byte order conventions communicate with each other, the Internet protocols specify a canonical byte order convention in big-endian format for data transmitted over the network. This is known as Network Byte Order.

The ordering used on a computer is called Host Byte Order.

* A host system may be little-endian but while sending data into the n/w, it must covert data into big-endian format. Likewise, a little-endian machine must first convert network data into little-endian format before processing it.

struct sockaddr → feeds data into a socket

(struct sockaddr)

- sa_family – Value is AF_INET
  - ↳ it an address family that is used to designate the type of addresses that your socket can communicate with (e.g. IPv4 addresses)

- sa_data : contains the destination IP address and port number for the socket.

```
Struct in_addr{
    unsigned long s_addr; // 32-bit long or 4 bytes in Network Byte Order.
};
```

sin_addr and sin_port → Network Byte Order – get encapsulated in the packet at the IP and Transport layers, respectively.

sin_family → Host Byte Order
- ↳ only used by the local kernel to determine what type of address the structure (sockaddr_in) contains and is not sent out on the network.

Sendto()

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen)
```

→ Returns -1 if there is an error.

int sendto(int sockfd, flags, const struct sockaddr *to, int tolen)

→ sends the no. of bytes actually sent

- sockfd: socket descriptor you want to send data to.
- msg: pointer to the data you want to send.
- len: length of the data in bytes.
- flags: set to 0.
- to: is a pointer to "struct sockaddr" that contains a destination IP address and port.
- tolen: simply set to sizeof(struct sockaddr)

recvfrom()

int recvfrom(int sockfd, void *buff, int len, unsigned int flags, struct sockaddr *from, int *fromlen)

→ Returns -1 if there is an error.

→ returns the no. of bytes received.

- sockfd — socket description to read data from.
- buff - buffer is to read the information into.
- len - maximum length of the buffer.
- flags — can be set to 0.
- from - pointer to the local "struct sockaddr" that will be filled with the IP address and port of the originating machine (source)
- fromlen: pointer to a local 'int' that should be initialized to sizeof(struct sockaddr).

connect() — Connect to a remote host

connect() — Connect to a remote host

```
int connect (int sockfd, struct sockaddr *serv_addr, int addrlen);
```

Client side

- `sockfd` — socket descriptor returned by `socket()`
- `serv_addr` — pointer to "struct sockaddr" containing destination IP address and port no.
- `addrlen` — set to `sizeof(struct sockaddr)`.
- returns −1 if there is an error.

— No need to call `bind()` — we don't care our local port numbers — the kernel which choose a local port number on behalf of us and the site we connect will automatically get this information from us.