Sockets

Network Layers

LAYER	WHAT DOES IT DO?
Transport	Deals with processes on machine
Network	Uniqueness in the scale of the internet
Data Link	Locally inside the LAN
Physical	Electrons

Network API

The OS provides API for network applications. Desirable characteristics of network APIs include:

- Simple to use.
- Flexible
    - Independent of app
    - allows usage of all functionality of network
- Standardized

API for networks are called sockets.

Division of Labour

Network:

- Deliver data packet to destination host
- based on destination IP address

OS:

- Deliver data to sockets based on port number

Application:

- read/write to/from sockets
- interpret the data

Sockets

Procs send/receive messages to/from their sockets. Using the socket for communication relies on transport infrastructure. Analogous to a mailbox. They support stream and datagram packets. They are similiar to the UNIX file I/O API i.e. provides a file descriptor.

Stream Sockets

- connection oriented
- two-way communication
- reliable, in-order delivery guaranteed
- Transmission Control Protocol(TCP)

Identified by `SOCK_STREAM` in the sockets API.

Datagram sockets

- connectionless, each packet is independent.
- best-effort
- no order guaranteed
- User Datagram Protocol(UDP)

Identified by `SOCK_DGRAM` in the sockets API.

Socket Identification

Communication Protocol:

- `SOCK_DGRAM` or `SOCK_STREAM`

Receiving host:

- Unique destination address: IP address(32-bit quantity)

Receiving socket:

- Unique Destination port(16-bit quantity)

Port Numbers

Popular applications(servers) have well-known ports, like 80 for web. Clients then choose unused ephemeral(temporary) ports(b/w 1024 and 65535).

Communication

Clients and servers communicate by sending streams of data over connections. These are p2p, full-duplex and reliable. The socket address is (IP addr:Port number). Each connection is identified by a pair of sockets.

Clients

Clients use the IP address of the server socket to identify the host. Well-known ports in the server socket identifies the service being provided. It is the responsibility of the Kernel to direct the connection to the ports on the machine.

Servers

These are long-running(daemon) processes. They run till the machine is turned off, or spawned in response to a connection to a port(uses `inetd` to listen on ports).

Byte Ordering

Different machines have different byte ordering schemes. It can be either little-endian(high order to low order bytes) or big-endian(low order to high order bytes). This is a problem because we don't know which order to send messages in. So we need to create a new byte-ordering for sending messages. This is a standard format decided upon. The socket API gives us some functions to do this:

```
uint16_t htons(uint16_t host16bitvalue);
uint32_t htonl(uint32_t host16bitvalue);
uint16_t ntohs(uint16_t net16bitvalue);
uint32_t ntohl(uint32_t net16bitvalue);
```

Socket Creation and Setup

- `socket()` to create a socket file descriptor(protocol mentioned here). `AF_INET` specifies IPv4, `AF_INET6` for IPv6 etc. `SOCK_STREAM` and `SOCK_DGRAM` for TCP/UDP. If error -1, else file descriptor of socket.
- `bind()` to bind to an address. Choose a port number from 1024 to 49151. 49152 through 65535 are random ports, avoid using them. You give `bind()` and IP address and port number.
- `listen()` to set it in passive mode and bound on length of unaccepted connections queue.
- `accept()` returns a file descriptor of the client socket connected to. It also gets the address of the client.

Establishing Connection

Client executes `connect()`. Allows it to connect to the server address.

Sending and receiving data

As sockets are file descriptors, they can be written to/read from using `read()` and `write()` system calls. `send()` and `recv()` can also be used, and they can be controlled with flags. For UDP sockets use `sendto()` and `recvfrom()` where you mention the destination address.

Tearing down sockets

- `close()` closes sockets. This closes communication in both direction. After closing, the socket is not valid for reading or writing.
- `shutdown()` forces termination of communication in one or both direction. `SHUT_RD` stops reading, `SHUT_WR` writing and `SHUT_RDWR` both.

TCP teardown doesn't happen until all copies having access to the socket have closed the socket. Close is the only way to actually destroy a socket.
