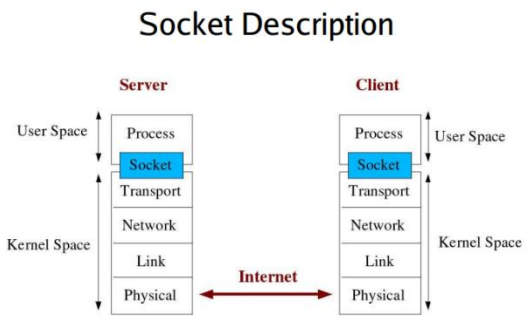
Endianness

- The order in which computer memory stores a sequence of bytes
- Big-Endian: Most important bit is stored in lowest address
- Little-Endian: Most important bit is stored in highest address
- TCP/IP uses big-endian
- If two computers with different architectures wants to communicate, they should convert the data to same endian structure
- Byte swapping, there are many software to change endianness (ex. OSByteOrder.h for macOS)

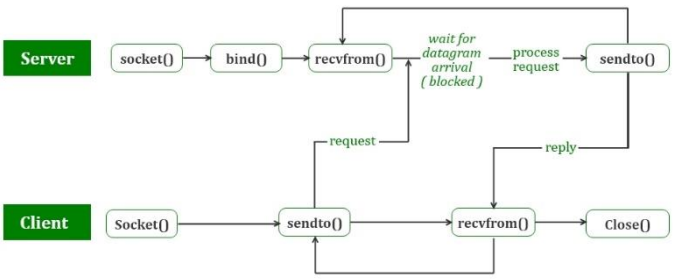
Socket Programming

- Socket: An interface between an application process and transport layer
- Application processes communicate (send/receive data) themselves via socket
- Internet sockets are two types: Stream socket for TCP and Datagram socket for UDP
- Byte ordering:
- Network byte order → High order byte – lowest address, generally big endian



- Host byte order (byte ordering of the computer architecture), generally little endian because of Intel domination
- TCP/IP expects Network byte order

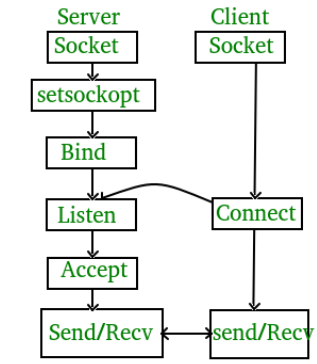
1. UDP:
   - User datagram protocol
   - UDP is a simple transport-layer protocol
   - UDP is connectionless which means there is no established connection between client and server.
   - It is unreliable since there is no guarantee that a UDP will reach the destination, that the order of the datagrams will be preserved across the network or that datagrams arrive only once

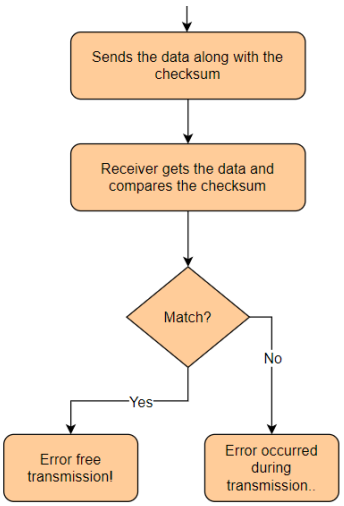


   - If a datagram is dropped in the network, it is not automatically retransmitted

- UDP datagrams have a fixed length and the length is passed to the application with the data

2. TCP:
   - Transmission control protocol
   - TCP is a connection-oriented service, since it establishes connection before transmitting data
   - TCP is reliable because it ensures that the data is sent correctly and completely.
   - First, TCP sends the data to server and requires an acknowledgement in return. If acknowledgement is not received, it transmits the data again.
     Also, TCP assigns a sequence number to each octet transmitted. These numbers are used for ordering correctly the transmitted segments and eliminating duplicates.
   - TCP is a bitstream protocol, which means there is no length limit





Checksum is used for checking the transmitted data is altered in the transmission and it aims to prevent data corruption

TCP functions

Client functions:

- **socket** (int family, int type, int protocol):
  creates the socket with specified protocol like (TCP based on IPv4, TCP based on IPv6, UDP)
- **connect** (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen):
  establish a connection between a TCP client and a TCP server, returns 0 for successful connection, -1 otherwise
- **read(), write():**
  send and receive data with these functions
- **close** (int sockfd):
  close a socket and terminate a TCP socket

Server functions:

- **bind**(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen):
  assigns a local protocol address to a socket. Address is combination of an IPv4 or IPv6 address (32-bit or 128-bit) address along with a 16 bit TCP port number
- **listen**(int sockfd, int backlog):
  unconnected socket → passive socket (start waiting for a connection), backlog is for max number of connections

- **accept**(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen):
  retrieve a connect request and convert that into a request
- **send() and receive():**
  similar to read and write functions but some options can be specified

UDP functions

- recvfrom():
  passing socket address it reads the specified number of bytes

- sendto():
  similar to write function with some additional options

Multiple Threading

Multiple threading is a method being used to exploit instruction level parallelism. Since speeding up the single threaded programs is becoming harder, todays computers try to execute multiple programs parallelly or concurrently. Multiple threading aims to improve throughput the tasks and performance gain.

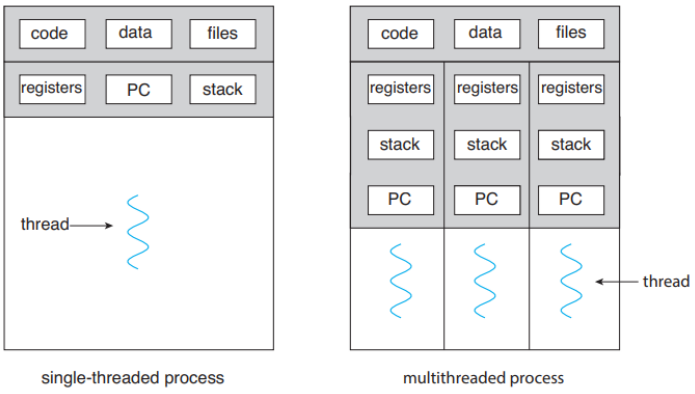


Figure 4.1 Single-threaded and multithreaded processes.

Using idle resources which can not be used by a single thread

Some problems:

- Multiple threads may try to use the same cache at the same time, therefore the expected speed up may be decreased.
- Multiple threads can lead to lots of cache or TLB misses, so the performance gain can be changed.
- Fork – exec problems