Socket Programming

Anand Baswade

anand@iitbhilai.ac.in

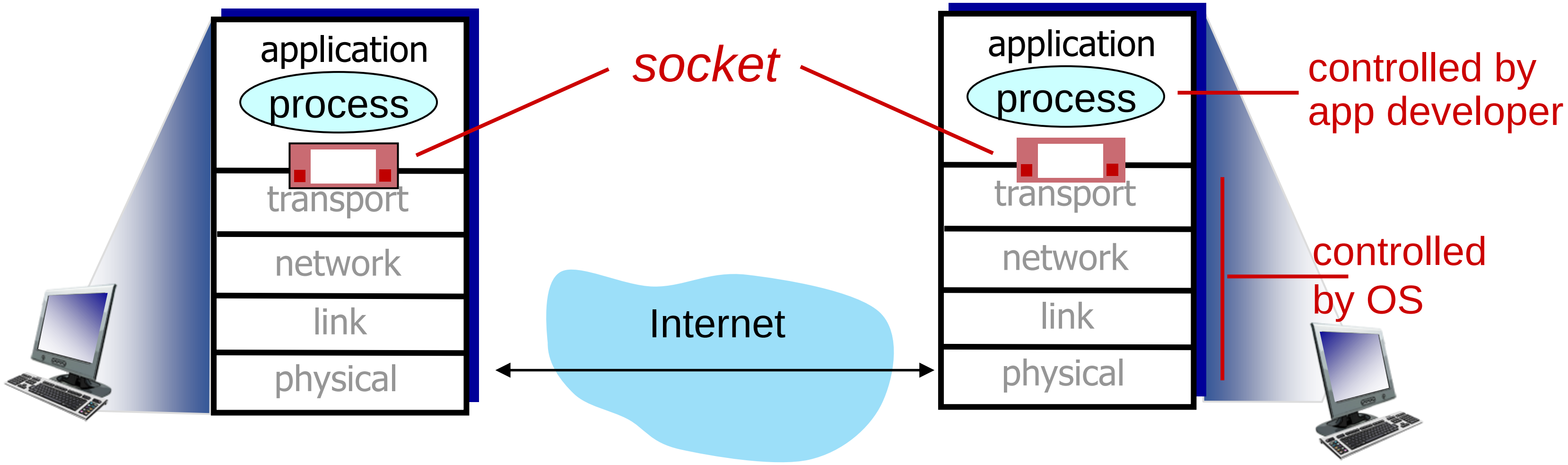
Sources/References

- https://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html
- http://www.linuxhowtos.org/C_C++/socket.htm
- Tutorial on Socket Programming http://www.cs.northwestern.edu/~agupta/cs340/sockets/Tutorial_Socket.ppt]
- Computer Networks: Top Down Approach by Ross and Kuros

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol

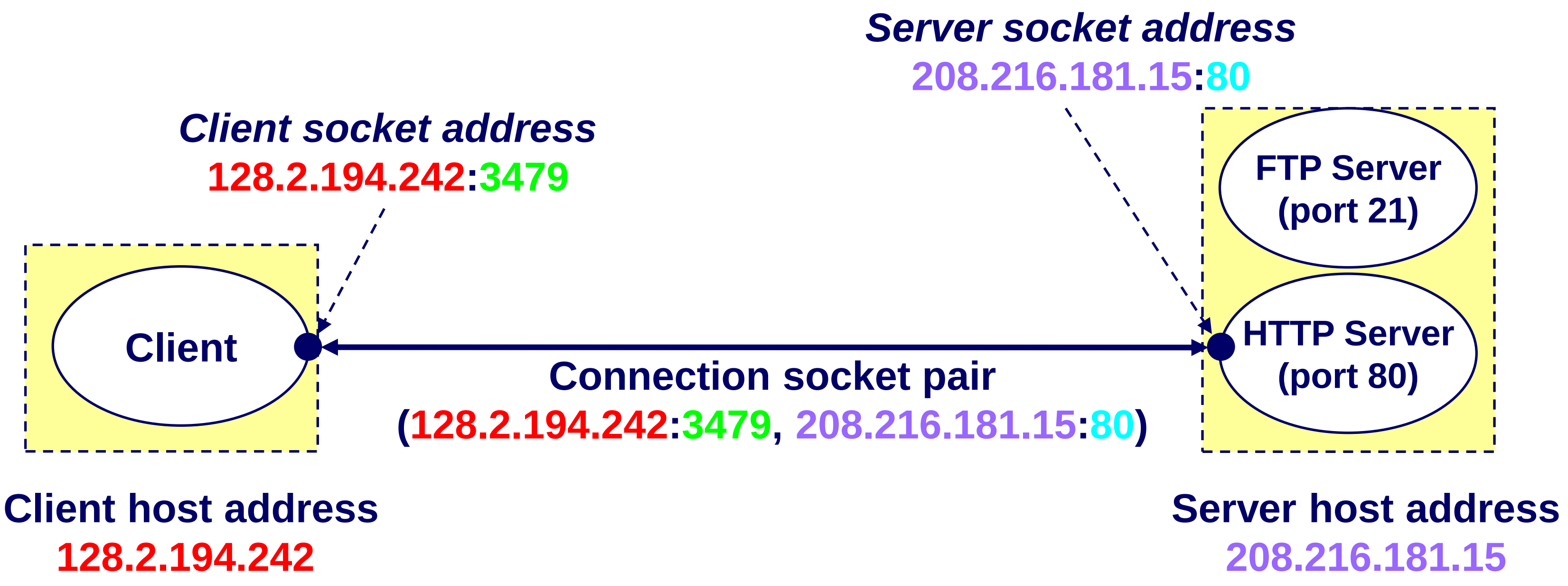


Sockets

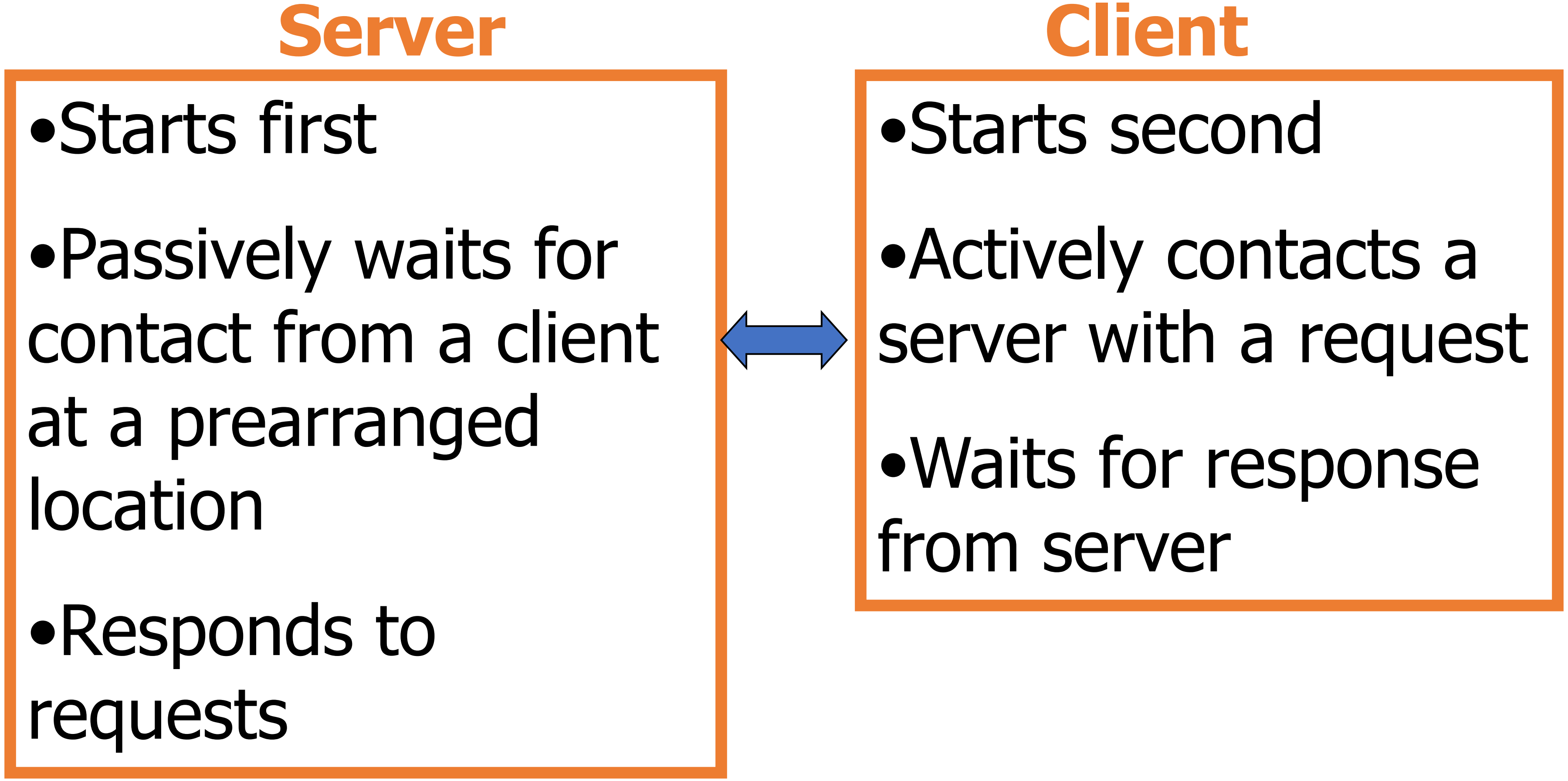
- How to use sockets
  - Setup socket
    - Where is the remote machine (IP address, hostname)
    - What service gets the data (port)
  - Send and Receive
    - Designed just like any other I/O in unix
    - send -- write
    - recv -- read
  - Close the socket

Identify the Destination

- Addressing
  - IP address
  - hostname (resolve to IP address via DNS)
- Multiplexing
  - port



Client/Server Model



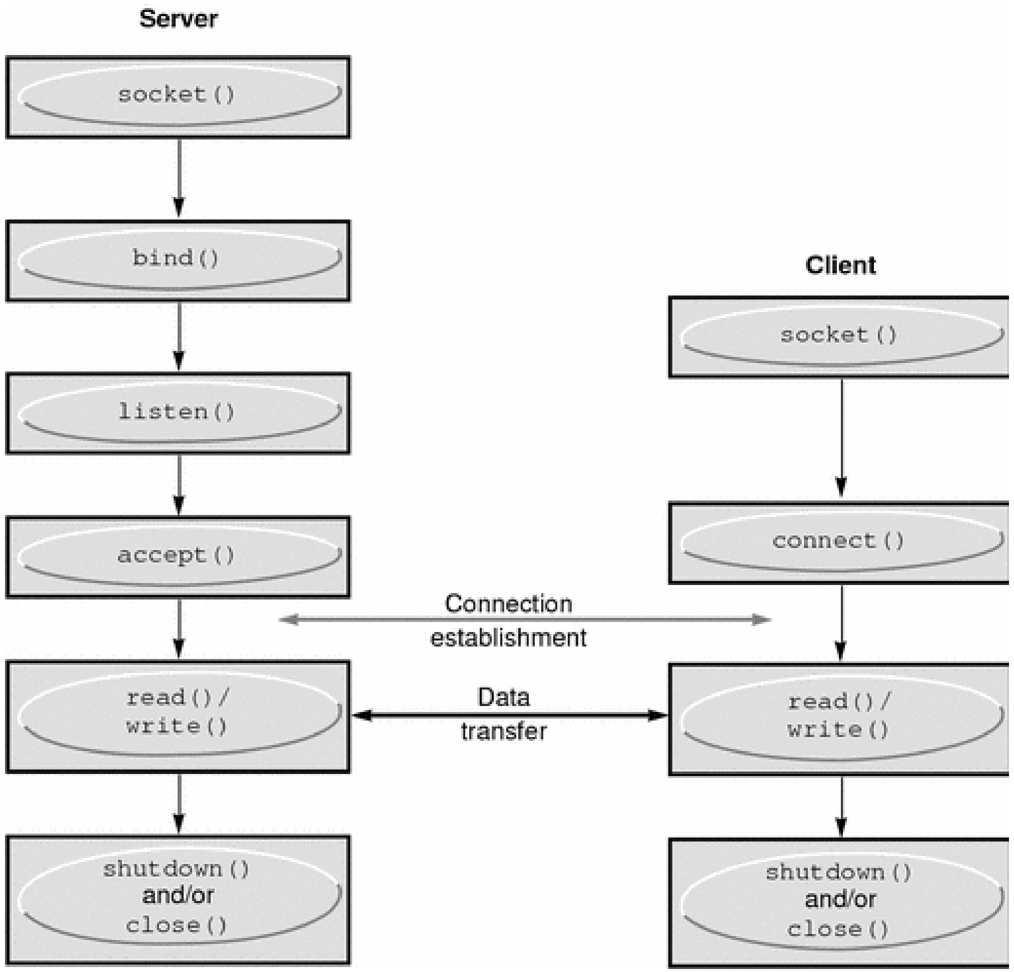
Server

- Starts first
- Passively waits for contact from a client at a prearranged location
- Responds to requests

Client

- Starts second
- Actively contacts a server with a request
- Waits for response from server

Socket Programming- System calls



Steps for establishing a socket on the *server* side

1. Create a socket with the socket() system call

2. Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.

3. Listen for connections with the listen() system call

4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5. Send and receive data

Steps for establishing a socket on the *client* side

1. Create a socket with the socket() system call

2. Connect the socket to the address of the server using the connect() system call

3. Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

Source: www.cs.northwestern.edu Tutorial on Socket Programming

Socket Types

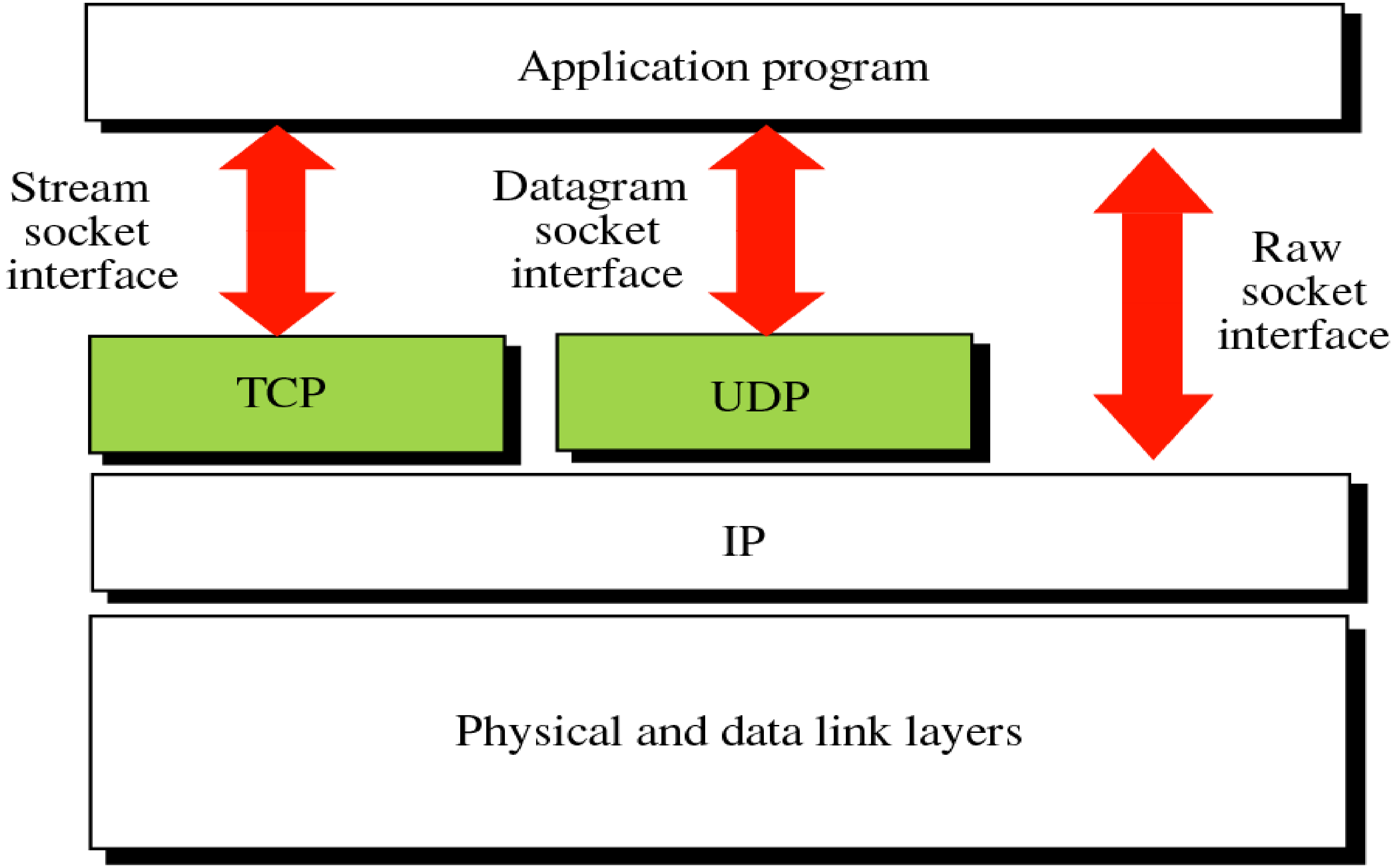
Two socket types for two transport services:

- UDP SOCKET
    - *Datagram Socket (SOCK_DGRAM):* unreliable datagram

- *TCP SOCKET*
    - *Stream Socket (SOCK_STREAM):* reliable, byte stream-oriented

RAW Socket: If you want to bypass the transport layer

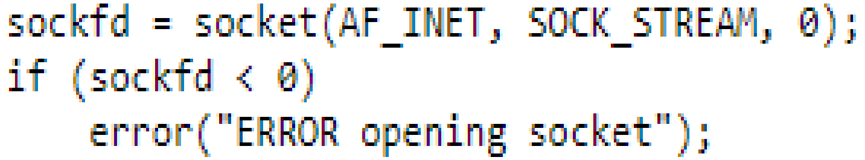
Socket Types



Source: www.cs.northwestern.edu Tutorial on Socket Programming

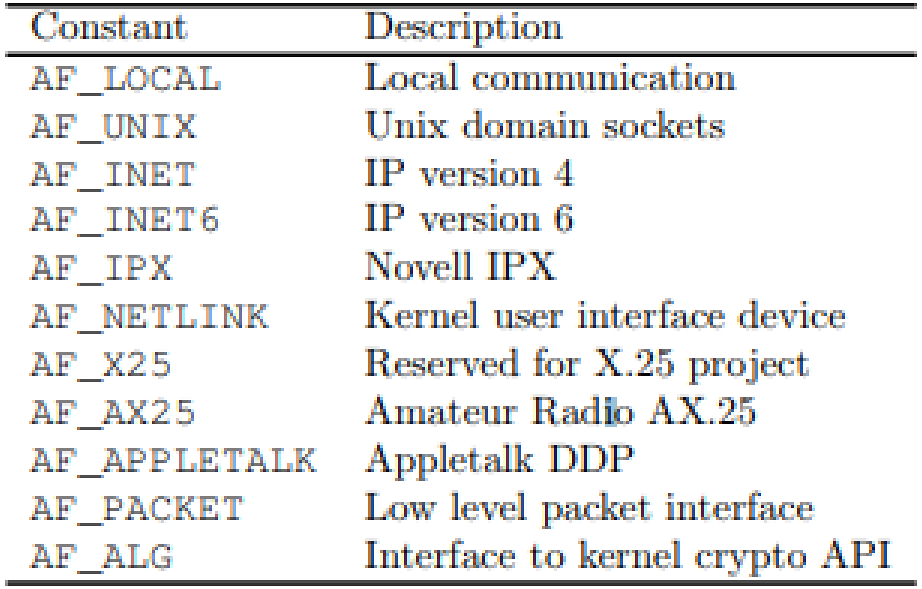
socket() -- Get the file descriptor

- int socket(int family, int type, int protocol);
  - domain should be set to AF_INET (e.g., IPV4)
    - AF_INET -- IPv4 (AF_INET6 for IPv6)
  - the type of service (e.g., STREAM or DGRAM)
    - SOCK_STREAM -- TCP
    - SOCK_DGRAM -- UDP
  - set protocol to 0 to have socket choose the correct protocol based on type
    - It always set to 0 except for unusual circumstances **(Explore),** OS will choose TCP for stream sockets and UDP for datagram sockets.)
  - socket() returns a socket descriptor for use in later system calls or -1 on error
- For example,
  - *int sockfd = socket(AF_INET, SOCK_STREAM, 0);*



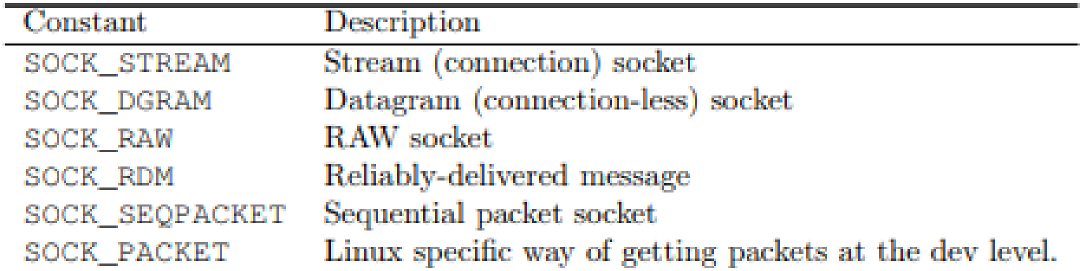
```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0)
    error("ERROR opening socket");
```

- **family** expects a constant value that describes the used address family. The following values are defined in <sys/socket.h>



Constant	Description
AF_LOCAL	Local communication
AF_UNIX	Unix domain sockets
AF_INET	IP version 4
AF_INET6	IP version 6
AF_IPX	Novell IPX
AF_NETLINK	Kernel user interface device
AF_X25	Reserved for X.25 project
AF_AX25	Amateur Radio AX.25
AF_APPLETALK	Appletalk DDP
AF_PACKET	Low level packet interface
AF_ALG	Interface to kernel crypto API

- **Type** defines the socket type.



Constant	Description
SOCK_STREAM	Stream (connection) socket
SOCK_DGRAM	Datagram (connection-less) socket
SOCK_RAW	RAW socket
SOCK_RDM	Reliably-delivered message
SOCK_SEQPACKET	Sequential packet socket
SOCK_PACKET	Linux specific way of getting packets at the dev level.

Socket Data Structures

- **Struct Sockaddr:** Holds socket address information for many types of sockets
  - struct sockaddr

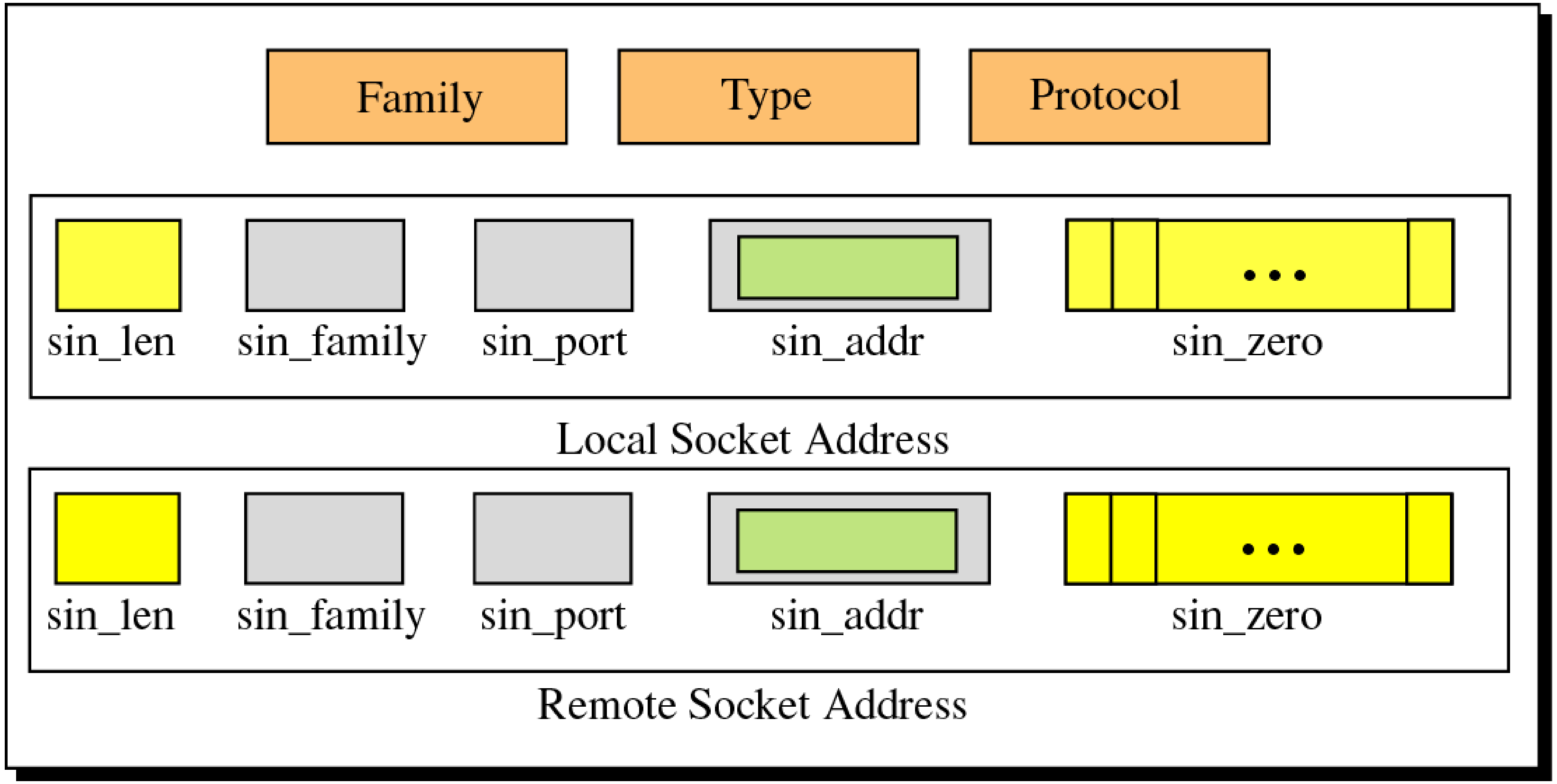
    ```
    { unsigned short sa_family;             //address family AF_xxx
       unsigned short sa_data[14];        //14 bytes holds IP and port number.
    }
    ```

- **struct sockaddr_in:** A parallel structure that makes it easy to reference elements of the socket address
  - struct sockaddr_in

    ```
    { short int sin_family;        // set to AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr sin_addr;      // Internet address
    unsigned char sin_zero[8];  //set to all zeros
    }
    ```

- struct in_addr { unsigned long s_addr; // that's a 32bit long, or 4 bytes };

Socket Structure



Socket

Source: www.cs.northwestern.edu Tutorial on Socket Programming

bind() – Bind to IP and Port Number

- Used to associate a socket with a port on the local machine
  - The port number is used by the kernel to match an incoming packet to a process.
- int bind(int sockfd, struct sockaddr *my_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - my_addr is pointer to struct sockaddr that contains information about your IP address and port
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
  - my_addr.sin_port = 4000; //choose an unused port at random
  - my_addr.sin_addr.s_addr = INADDR_ANY; //use my IP adr

connect() - Hello!

- Used by Connection oriented clients to connects to a remote host/server.
- int connect(int sockfd, struct sockaddr *serv_addr, int addrlen)
  - sockfd is the socket descriptor returned by socket()
  - serv_addr is pointer to struct sockaddr that **contains information on destination IP address and port**
  - addrlen is set to sizeof(struct sockaddr)
  - returns -1 on error
- No need to bind(), kernel will choose a port

Listen() – Wait for Incoming connections

- int listen(int sockfd, int backlog);
  - sockfd is the socket file descriptor returned by socket()
  - backlog is the number of connections allowed on the incoming queue
  - listen() returns -1 on error

  - Need to call bind() before you can listen()
    - socket()
    - bind()
    - listen()
    - accept()

accept() – Connection Est.

- int accept(int sockfd, void *addr, int *addrlen);
  - sockfd is the listening socket descriptor
  - information about incoming connection is stored in addr which is a pointer to a local struct sockaddr_in
  - addrlen is set to sizeof(struct sockaddr_in)
  - accept returns a new socket file descriptor to use for this accepted connection and -1 on error

send() and recv() – Let’s Talk

- int send(int sockfd, const void *msg, int len, int flags);
  - sockfd is the socket descriptor you want to send data to (returned by socket() or got from accept())
  - msg is a pointer to the data you want to send
  - len is the length of that data in bytes
  - set flags to 0 for now
  - send() returns the number of bytes actually sent or -1 on error

send() and recv() – Let’s Talk

- int recv(int sockfd, void *buf, int len, int flags);
  - sockfd is the socket descriptor to read from
  - buf is the buffer to read the information into
  - len is the maximum length of the buffer
  - set flags to 0 for now
  - recv() returns the number of bytes actually read into the buffer or -1 on error
  - If recv() returns 0, the remote side has closed connection on you

sendto() and recvfrom() – UDP/SOCK_DGRAM

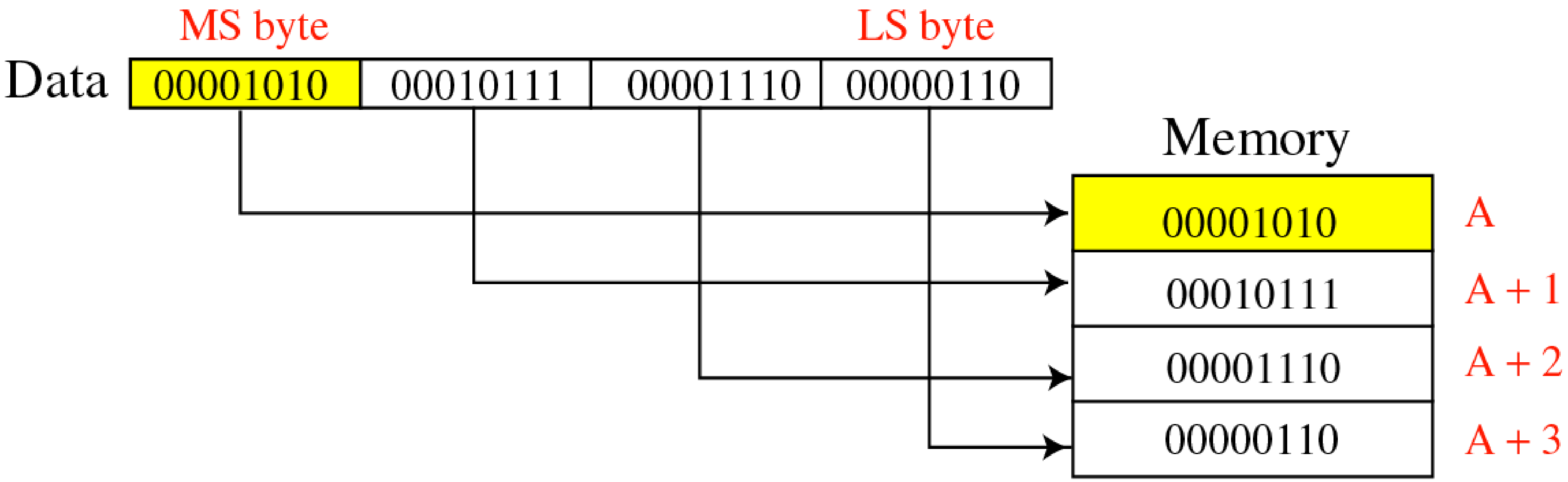
- int sendto(int sockfd, const void *msg, int len, int flags, const struct sockaddr *to, int tolen);
  - to is a pointer to a struct sockaddr which contains the destination IP and port
  - tolen is sizeof(struct sockaddr)
- int recvfrom(int sockfd, void *buf, int len, int flags, struct sockaddr *from, int *fromlen);
  - from is a pointer to a local struct sockaddr that will be filled with IP address and port of the originating machine
  - fromlen will contain length of address stored in from

close() - Bye Bye!

- int close(int sockfd);
  - Closes connection corresponding to the socket descriptor and frees the socket descriptor
  - Will prevent any more sends and recvs

Byte ordering (Little and Big Endian)

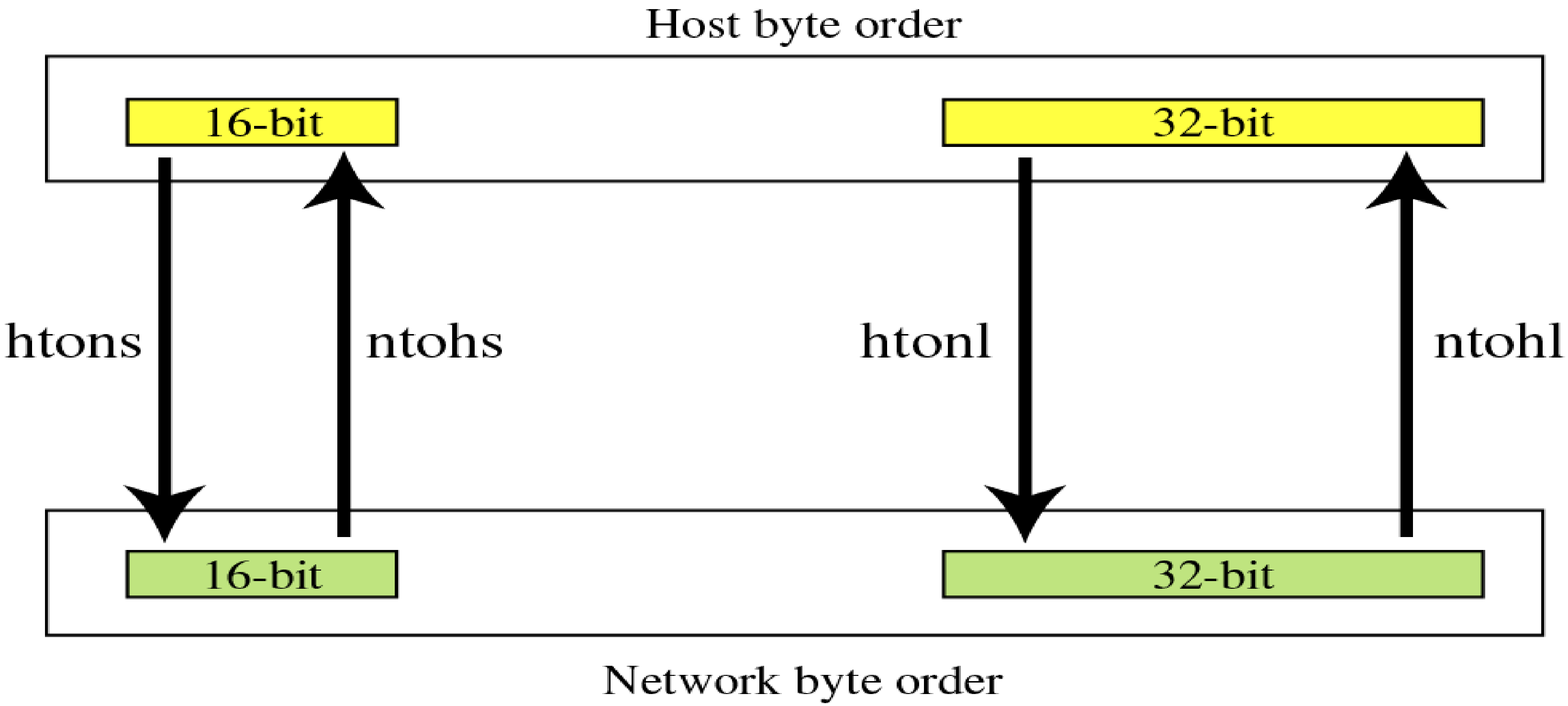
- Big Endian byte-order



The byte order for the TCP/IP protocol suite is big endian.

Source: www.cs.northwestern.edu Tutorial on Socket Programming

Byte-Order Transformation



u_short **htons** (u_short ***host_short***) ;

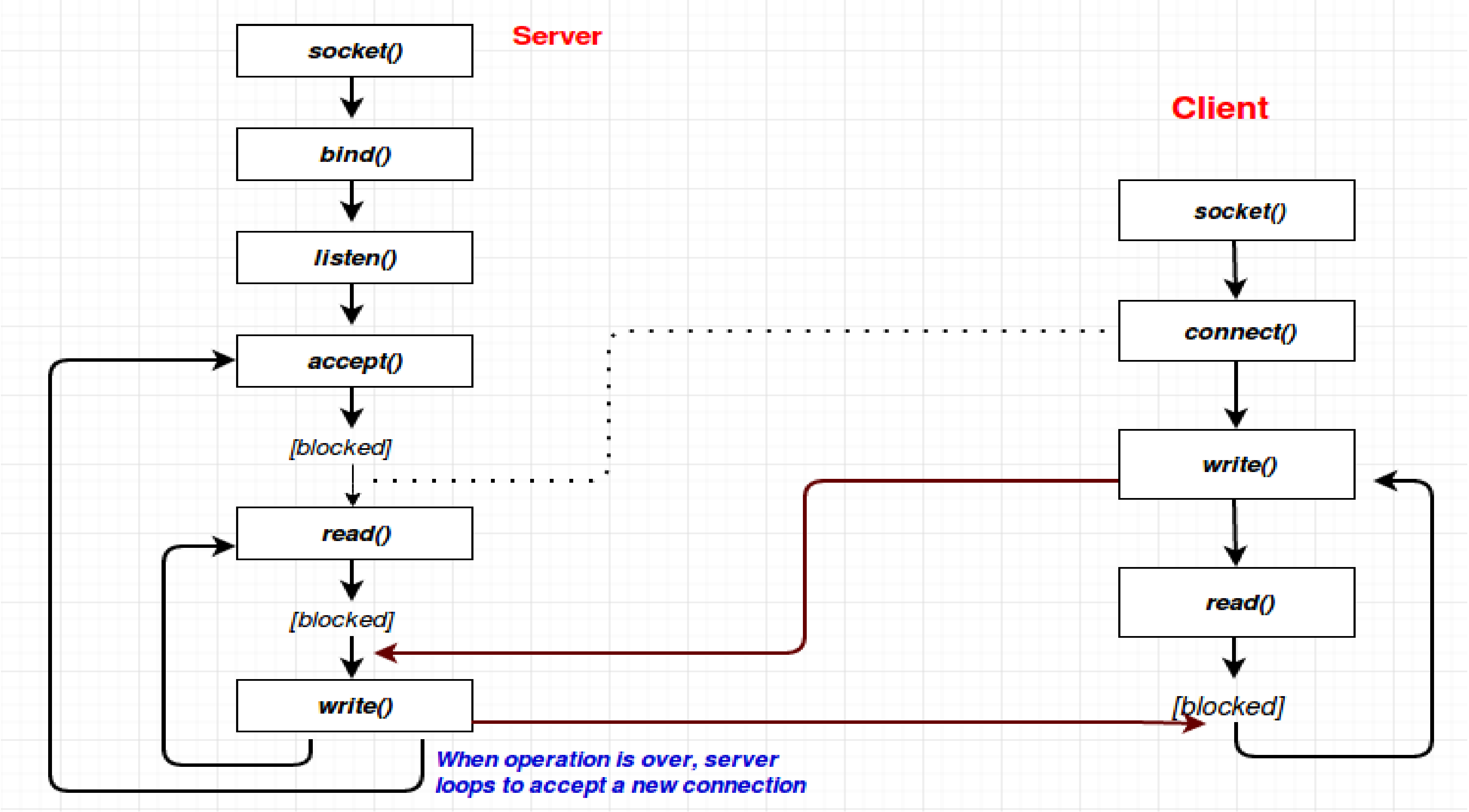
u_short **ntohs** (u_short ***network_short***) ;

u_long **htonl** (u_long ***host_long***) ;

u_long **ntohl** (u_long ***network_long***) ;

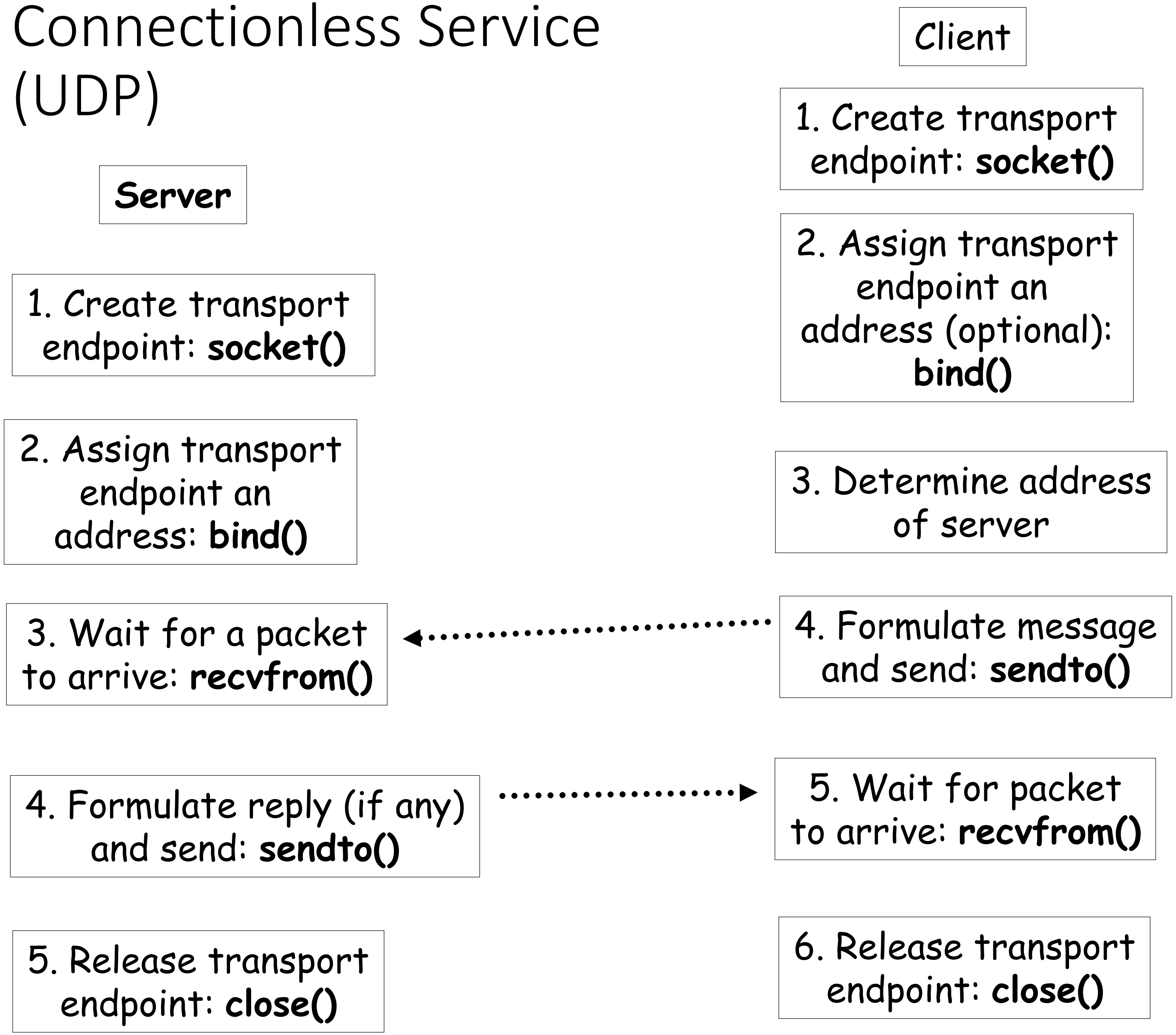
Source: www.cs.northwestern.edu Tutorial on Socket Programming

Socket Programming Flow



https://www.studytonight.com/network-programming-in-python/working-with-tcp-sockets

Connectionless Service (UDP)



Server

1. Create transport endpoint: **socket()**

2. Assign transport endpoint an address: **bind()**

3. Wait for a packet to arrive: **recvfrom()**

4. Formulate reply (if any) and send: **sendto()**

5. Release transport endpoint: **close()**

Client

1. Create transport endpoint: **socket()**

2. Assign transport endpoint an address (optional): **bind()**

3. Determine address of server

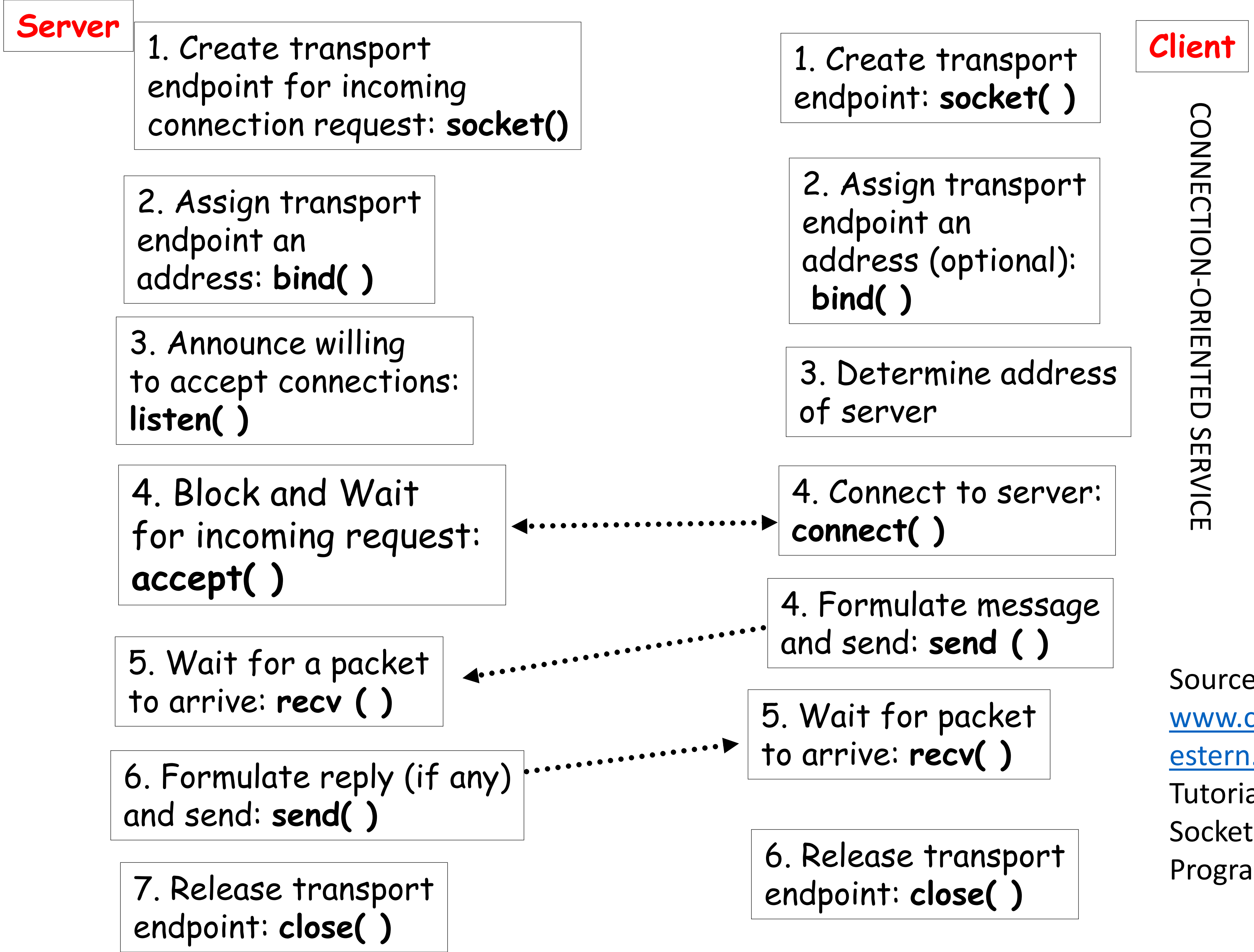
4. Formulate message and send: **sendto()**

5. Wait for packet to arrive: **recvfrom()**

6. Release transport endpoint: **close()**

Source: www.cs.northwestern.edu Tutorial on Socket Programming

CONNECTION-ORIENTED SERVICE



Server

1. Create transport endpoint for incoming connection request: **socket()**

2. Assign transport endpoint an address: **bind( )**

3. Announce willing to accept connections: **listen( )**

4. Block and Wait for incoming request: **accept( )**

5. Wait for a packet to arrive: **recv ( )**

6. Formulate reply (if any) and send: **send( )**

7. Release transport endpoint: **close( )**

Client

1. Create transport endpoint: **socket( )**

2. Assign transport endpoint an address (optional): **bind( )**

3. Determine address of server

4. Connect to server: **connect( )**

4. Formulate message and send: **send ( )**

5. Wait for packet to arrive: **recv( )**

6. Release transport endpoint: **close( )**

Source: www.cs.northwestern.edu Tutorial on Socket Programming

Demo - Example