Socket Programming Basics

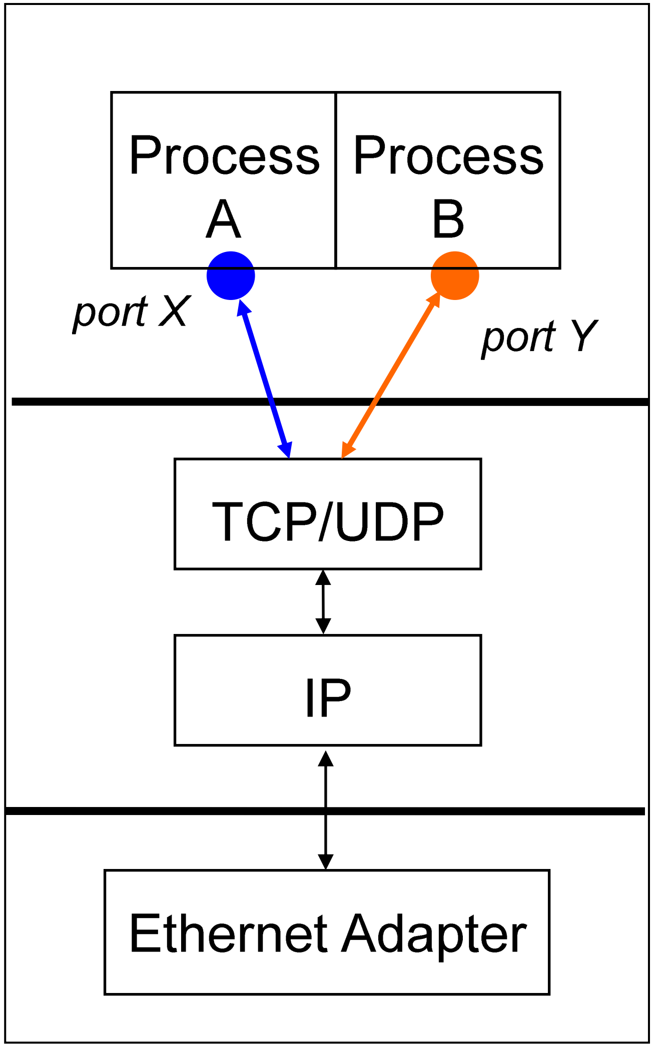
Need of Network Sockets

- To support message communication between two Networking Applications running on same (may user process communication as well) or different hosts

Properties of Network Socket

- Address of Computer / Node on Network?
  - **IP Address** (IPv4)
    - A 32 bit **unique** address on your network (Ex. 8.8.8.8)
    - Handled by L3 (IP/Internet Layer) of 5/7 layer protocol stack
- Address of Network Application on a Computer
  - **Port Number** [Logical]
    - A 16 bit unique identifier of Network Application
    - Handled by L4 (Transport Layer) of a 5/7 layer protocol stack
- Type of Communication
  - **TCP** (Connection Oriented / Reliable) / **UDP** (Connectionless / Best Effort)

Socket Identification



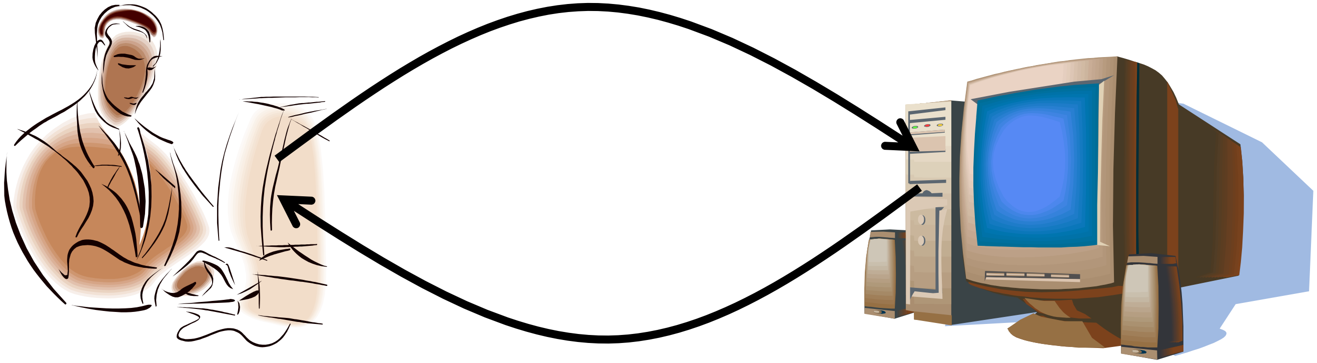
Socket Identification

- Can a machine has more than one IP address ?
  - **YES**, Each Network Interface Card (NIC) has a unique IP addresses (Ethernet and Wi-Fi NIC are assigned different IP address and both can be used at same time)
  - One can create multiple virtual Network Interfaces with each assigned different IP address associated with same NIC.
- Can one Socket (server) be made to associated with more than one IP
  - Either One or ALL
  - What if we want multiple ?
- Can two sockets be made to listen on same port
  - No, if the bind IP and Transport Layer protocol are same
  - Yes, if any of above is different

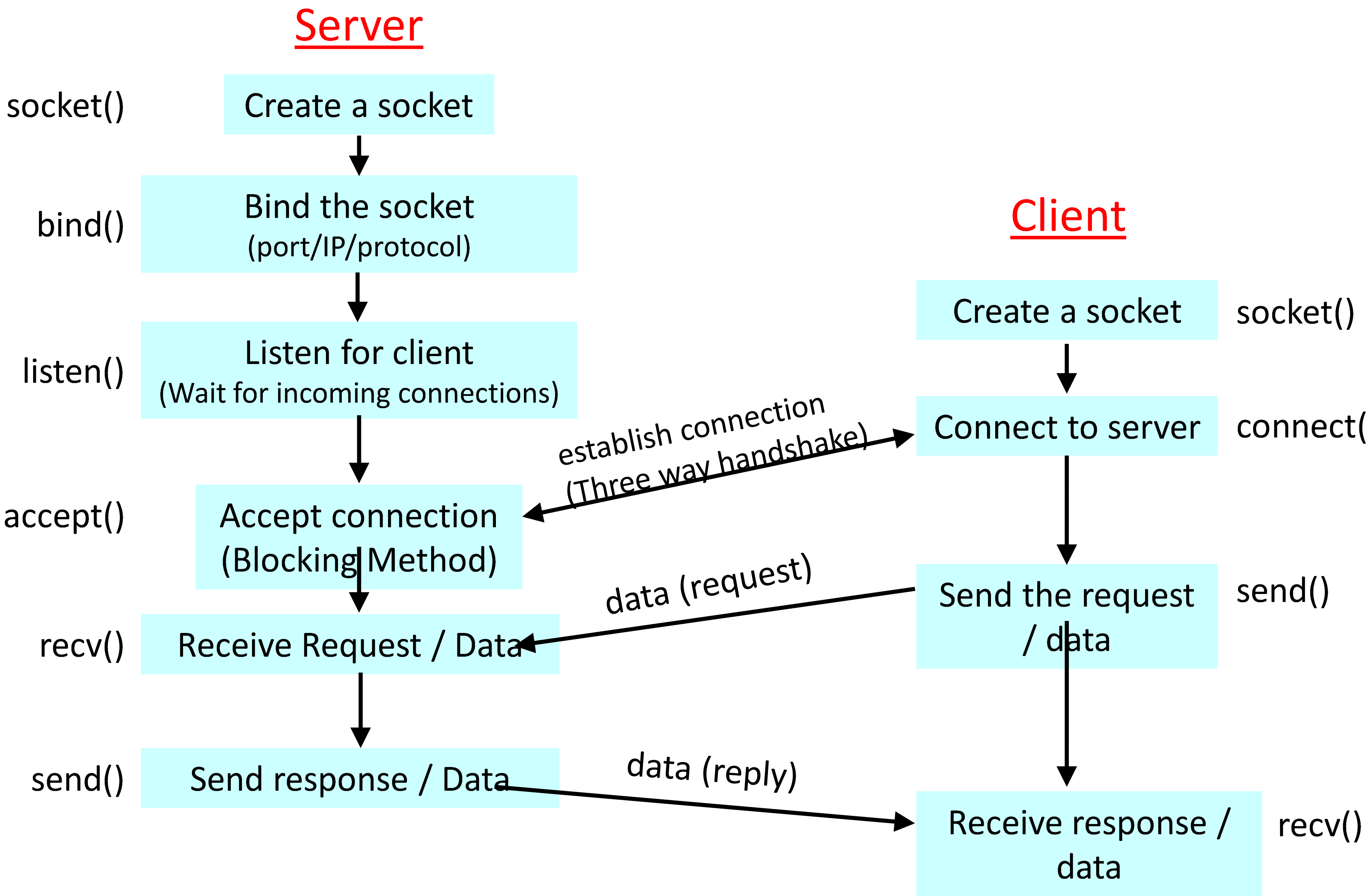
Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know server's address

- Server is "always on"
  - Handles services requests from many client hosts
  - E.g., Web server for the www.iitk.ac.in Web site
  - Doesn't initiate contact with the clients
  - Needs fixed, known address



Client-Server Communication
Stream Sockets (TCP): Connection-oriented



Server

socket() — Create a socket

bind() — Bind the socket (port/IP/protocol)

listen() — Listen for client (Wait for incoming connections)

accept() — Accept connection (Blocking Method)

recv() — Receive Request / Data

send() — Send response / Data

Client

Create a socket — socket()

Connect to server — connect()

Send the request / data — send()

Receive response / data — recv()

establish connection (Three way handshake)

data (request)

data (reply)

Implementation (Server)

- Create a Socket [Creating the Endpoint for Socket]
    - ***int sockid = socket(family, type, protocol);***

  – *Defined in sys/socket.h*

  – ***sockid****: socket descriptor, an integer (like a file-handle)*

  – ***family****: integer, communication domain, e.g.,*
    - *AF_INET, IPv4 protocols, Internet addresses (typically used)*
    - *AF_UNIX, Local communication, File addresses*

  – ***type****: communication type*
    - *SOCK_STREAM - reliable, 2-way, connection-based service* **[TCP]**
    - *SOCK_DGRAM - unreliable, connectionless, messages of maximum length* **[UDP]**

  – ***protocol****: specifies protocol*
    - *IPPROTO_TCP IPPROTO_UDP*
    - *usually set to 0 (i.e., use default protocol)*

  – *upon failure returns -1*

  – *Ex:* ***int sockid = socket(AF_INET, SOCK_STREAM, 0);***

Specifying Address

- Socket API defines a generic data type for addresses:

```
struct sockaddr {
unsigned short sa_family;          /* Address family (e.g. AF_INET) */
char sa_data[14];                  /* Family-specific address information */
}
```

- Particular form of the sockaddr used for TCP/IP addresses:

```
struct in_addr {
unsigned long s_addr               /* Internet address (32 bits) */
}
struct sockaddr_in {
unsigned short sin_family;          /* Internet protocol (AF_INET) */
unsigned short sin_port; /         * Address port (16 bits) */
struct in_addr sin_addr;           /* Internet address (32 bits) */
char sin_zero[8]; /                * Not used */
}
```

- sockaddr_in can be casted to a sockaddr

Assigning address to Socket

```
int status = bind(sockid, &addr, size);
```

- **sockid**: integer, socket descriptor
- **addr**: struct sockaddr, the (IP) address and port of the machine
  - for TCP/IP server, internet address is usually set to INADDR_ANY, i.e., chooses ALL incoming interface
  - or can be specified a particular IP
- **size**: the size (in bytes) of the addr structure
- **status**: 0 if successful bind, -1 otherwise
- Ex:

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_port = htons(5100);
addr.sin_addr.s_addr = htonl(INADDR_ANY);
bind(sockid, (struct sockaddr *) &addr, sizeof(addr10))
```

Start Listening for Connections

int status = listen(sockid, queueLimit);

- **Instructs TCP protocol implementation to listen for connections**
  - sockid: integer, socket descriptor
  - queuelen: integer, # of active participants that can “wait” for a connection while server is busy serving previously arrived client.
  - status: 0 if listening, -1 if error

Accepting connection request

```
int newsockid = accept(sockid, &clientAddr, &addrLen);
```

- newsockid: integer, the new socket (used for data-transfer)
- sockid: integer, the orig. socket (being listened on)
- clientAddr: struct sockaddr, address of the active participant
  - filled in upon return with the details of client information (which is available in the IP packet received by server)
  - addrLen: sizeof(clientAddr): value/result parameter
- **accept** method call is a blocking call. Program thread will keep waiting till it receives a connection request from client.

Data Communication

- Receiving Data

  n = read(newsockid, buffer, count);

  - newsockid : integer value returned by accept function call
  - buffer: buffer to store the read value
  - count: Number of bytes to be read
  - n - Number of blocks actually read (may be less than count)

- Sending Data

  n = write(newsockid, buffer, count);

  - newsockid : integer value returned by accept function call
  - buffer: data to be written
  - count: Number of bytes to be read
  - n - Number of blocks actually read (may be less than count)

- **fdopen :** can be used to make socket stream behave like FILE stream so that fread / fwrite and similar functions can be used.

Implementation (Client)

- Create the socket (end point for communication) same as it was created for server.

```
int sockid = socket(family, type, protocol);
```

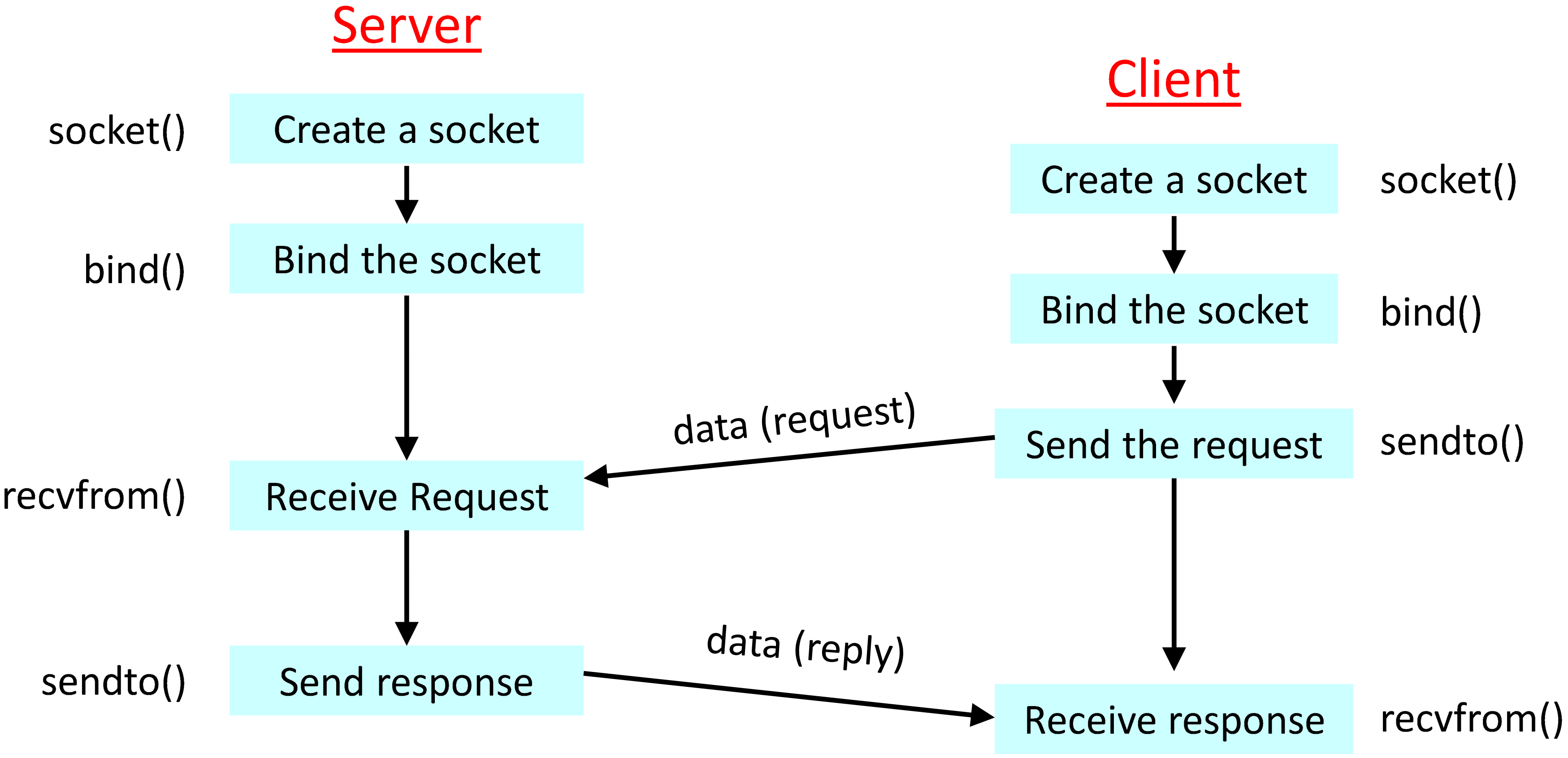
- Client executes a **connect** method call to send socket creation request to the server.

```
int status = connect(sockid, &serverAddr, addrlen);
```

  - sockid: integer, socket to be used in connection
  - serverAddr: struct sockaddr: address of the passive participant
  - addrlen: integer, sizeof(name)
  - status: 0 if successful connect, -1 otherwise

- On Successful connection sockid can be used to read and write data in the same way it was implemented for Server

Client-Server Communication
Datagram Sockets (UDP): Connectionless



Blocking vs. Non Blocking Socket

- Blocking Socket:
  - Clients will be served sequentially
- Non Blocking Socket:
  - Clients may be served in parallel
- Serving Clients in parallel with Blocking Socket
  - Create a child process or thread each time accept method is called. Child process will serve the client while parent will return to listening mode.

Concurrent Server - Example

```
while(1){
connfd = accept(listenfd, ...); /* blocking call */
  if ( (pid = fork()) == 0 ) {
    close(listenfd); /* child closes listening socket */
    /***process the request doing something using connfd ***/
    close(connfd);
    exit(0);  /* child terminates
   }
   close(connfd);  /*parent closes connected socket*/
}
```

- Questions ?