Intro To Socket Programming

By: Matthew Johnson

Sockets

- Endpoint of two-way communication between two programs
- UDP & TCP Sockets
  - UNIX
  - C
- Use the sockets API
  - sys/sockets.h

Client & Server

- Client
  - Needs resources from a server
  - Initiates a connection with a server
  - To create connection, client needs server's IP address and port number
- Server
  - Contains resources needed by clients
  - Waits for connections from clients on a specified port
  - Once a client initiates a connection, the server will have the client's IP address and port number

Socket Data Structures

- int socket(int domain, int type, int protocol)
  - socket() creates a socket to be used by the program
    - Returns socket descriptor of type int on success and -1 on failure
  - domain is the communication domain
    - For the labs we will use the constant PF_INET
  - type is the type of socket to be created
    - For lab 2 we will use SOCK_STREAM for TCP communication
    - For lab 3 we will use SOCK_DGRAM for UDP communication
  - protocol is the protocol used by the socket
    - For the labs we will use 0
  - Example: int sock = socket(PF_INET, SOCK_DGRAM,0)

Socket Data Structures Cont.

- Specifying Server Address for clients
- struct sockaddr_in addr
  - addr.sin_family
    - Communication domain
    - Set to PF_INET
  - addr.sin_port
    - Server port to send message to
    - Set to htons(portNum)
    - htons(int port) takes an integer and returns a network byte order short
  - addr.sin_addr.s_addr
    - Server IP address to send message to
    - Set to inet_addr(ipDest)
    - inet_addr(char *) takes a string and returns an IP address

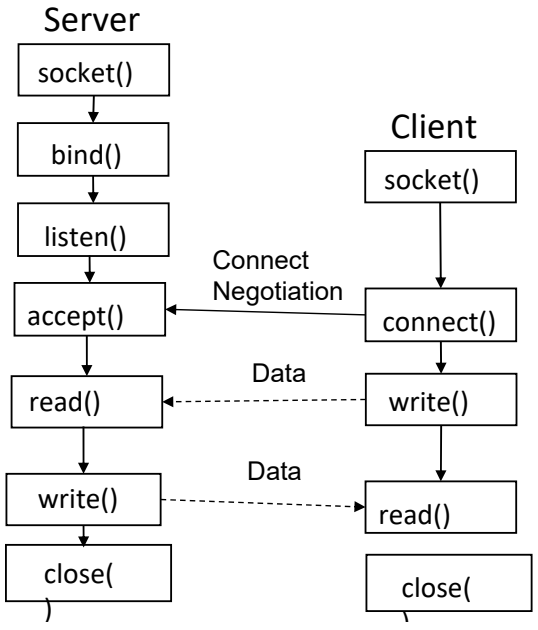
Socket Data Structures Cont.

- Specifying Client Address for servers
- struct sockaddr_in addr
  - addr.sin_family
    - Communication domain
    - For lab 3 PF_INET
  - addr.sin_port
    - Server port to receive messages into
    - Set to htons(portNum)
    - htons(int port) takes an integer and returns a network byte order short
  - addr.sin_addr.s_addr
    - Set to dummy IP address
    - Set to INADDR_ANY

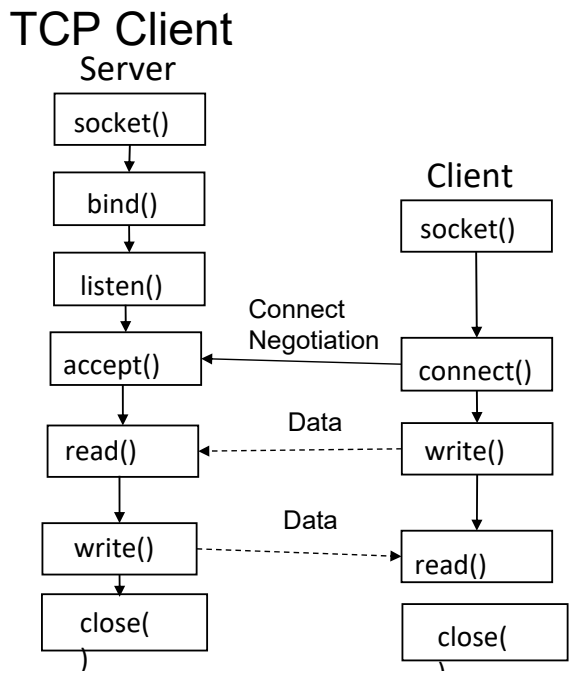
TCP Sockets

- Used in Lab 2
- Set up communication channel between server and client
- Server listens to a port for incoming communication requests
- Client connects to a server port to begin communication

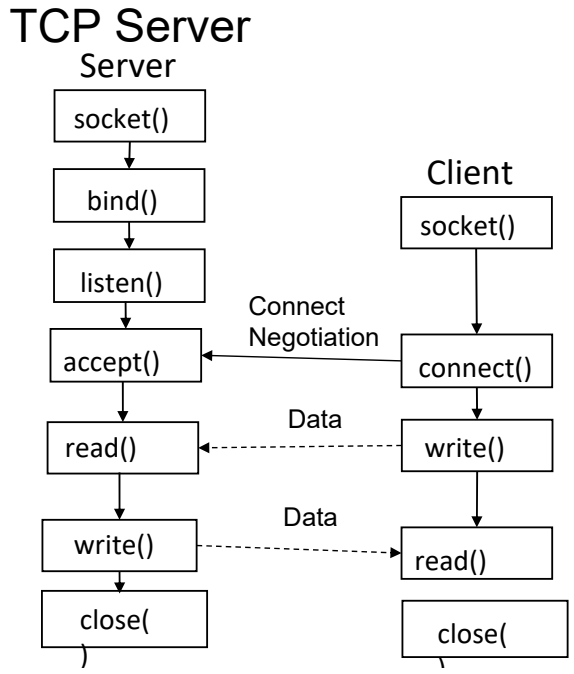
TCP Server



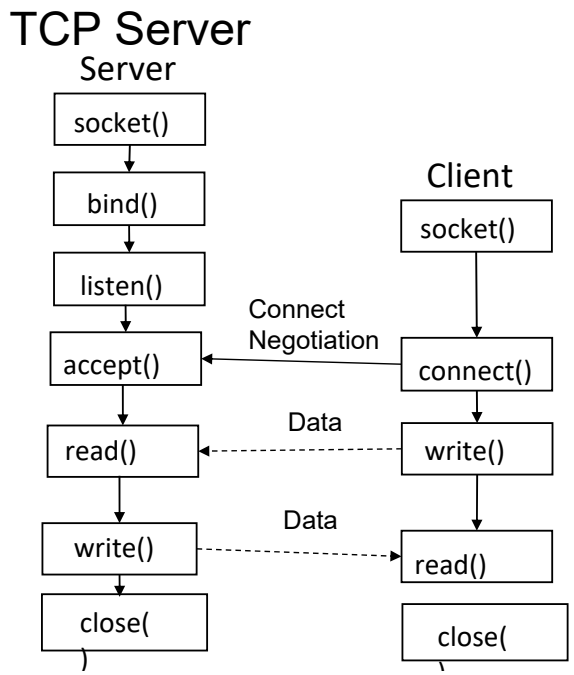
1. Server creates socket using socket()
    a. socket(int domain, int type, int protocol)



1.Client creates socket using socket()
   a. socket(int domain, int type, int protocol)

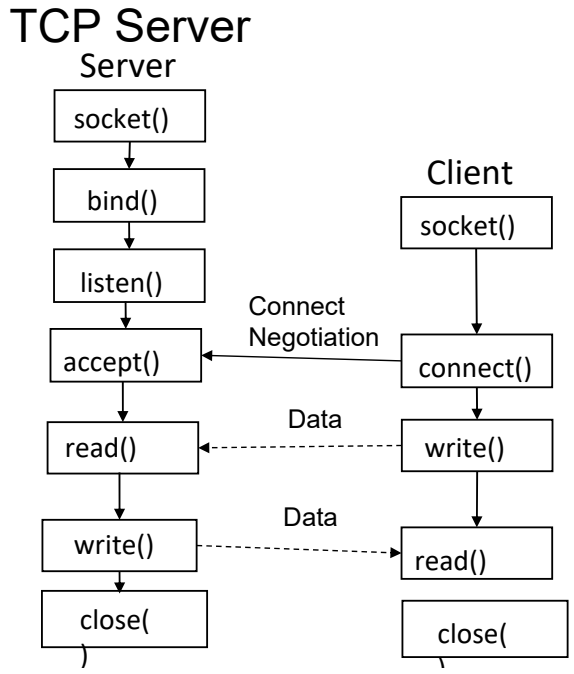


2.Server binds to port using bind()
   a. bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
   b. Allows server to bind to a port
   c. sockfd = socket created in step 1
   d. addr = struct containing server information
   e. addrlen = sizeof(serverAddr);
   f. returns 0 on success, -1 on failure



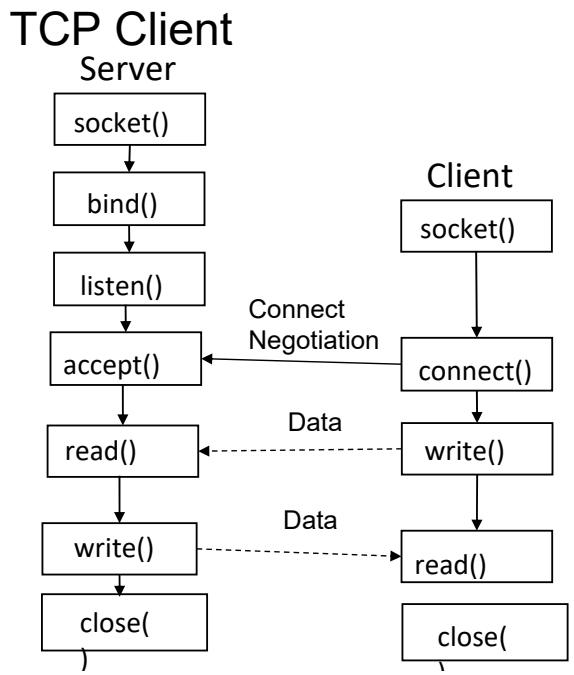
3.Server listens to port using listen()

a. listen(int sockfd, int backlog)
b. Allows server to listen to a port
c. sockfd = socket created in step 1
d. backlog = number of allowed pending connections
e. returns 0 on success, -1 on failure

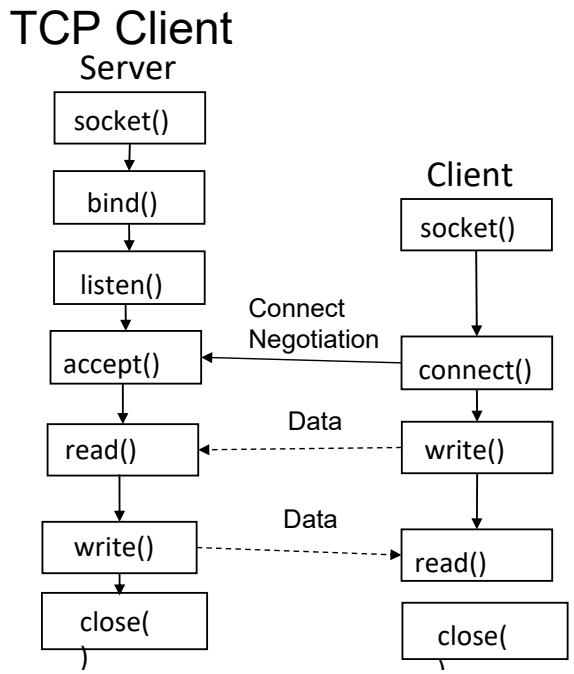


4.Server accepts an incoming connection using accept()

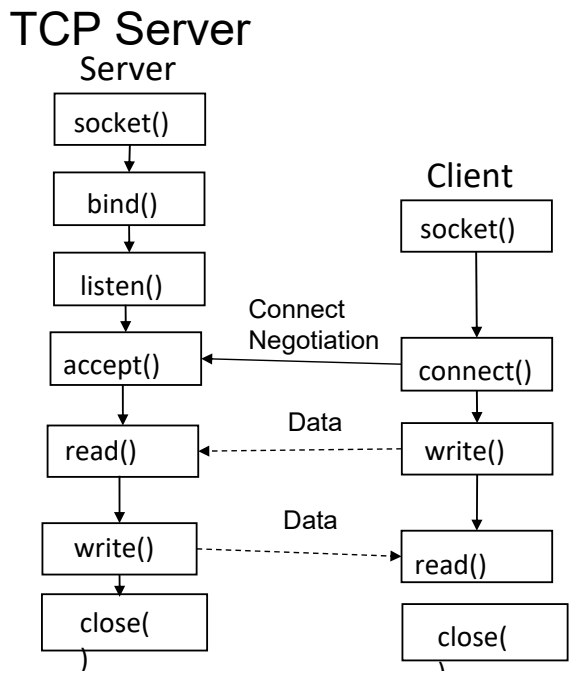
a. accept(int sockfd, struct sockaddr *cliaddr, int *addrlen)
b. Blocking while it waits for a connection
c. Accepts an incoming connection request
d. sockfd = socket created in step 1
e. cliaddr = pointer to sockaddr struct where client's address will be stored
f. addrlen = pointer to integer where length of client address will be stored
g. returns new socket on success, -1 on failure



2.Client attempts to connect to server port using connect()
   a. connect(int sockfd, struct sockaddr *seraddr, socklen_t addrlen);
   b. Attempts to make connection between client socket and server socket using TCP 3-way handshake
   c. sockfd = socket created in step 1
   d. seraddr = pointer to sockaddr struct where server's address and port are specified
   e. addrlen = length of server address struct
   f. returns 0 on success, -1 on failure

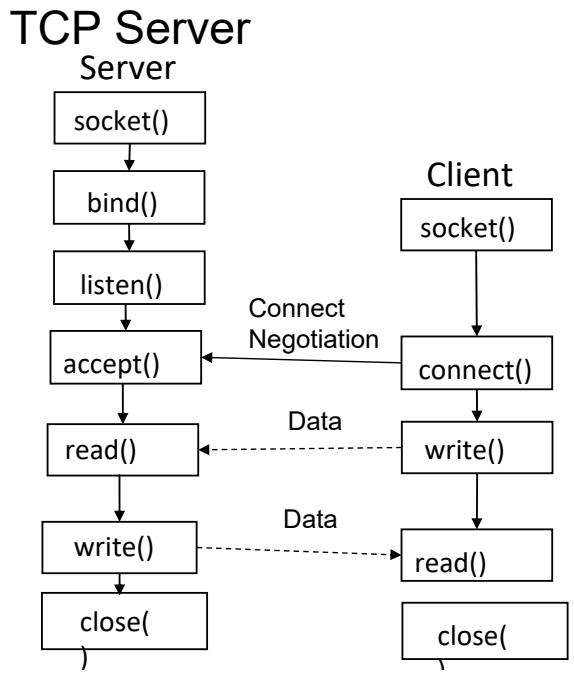


3.Client writes data over the socket using write()
   a. write(int fd, char *buf, int num)
   b. fd = socket created in step 1
   c. buf = char buffer that holds data to send
   d. num = number of bytes to send
   e. returns number of bytes sent on success, -1 on failure



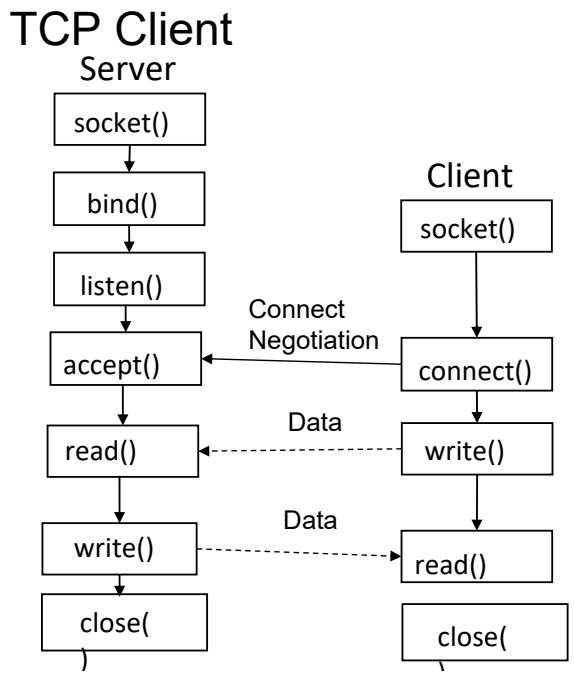
5.Server reads data sent by client using read()

a. read(int fd, char *buf, int max)
b. fd = socket created by accept() in step 4
c. buf = char buffer that will hold received data
d. max = maximum number of bytes that can be read
e. returns number of bytes read on success, -1 on failure



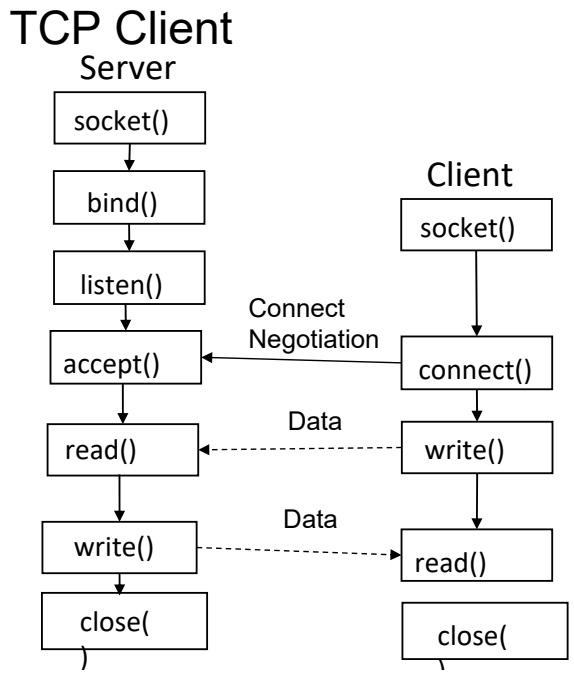
6.Server writes data over the socket using write()

a. write(int fd, char *buf, int num)
b. fd = socket created in accept() in step 4
c. buf = char buffer that holds data to send
d. num = number of bytes to send
e. returns number of bytes sent on success, -1 on failure



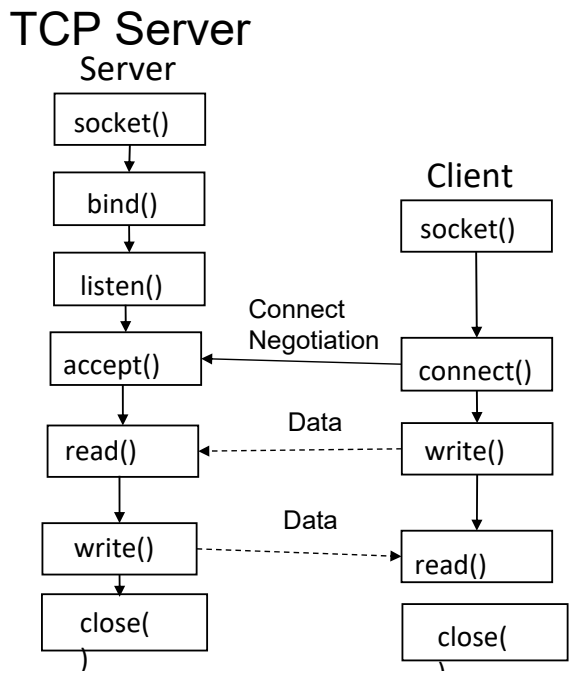
4.Client reads data sent by server using read()

a. read(int fd, char *buf, int max)
b. fd = socket created by step 1
c. buf = char buffer that will hold received data
d. max = maximum number of bytes that can be read
e. returns number of bytes read on success, -1 on failure



5.Client closes socket using close()

a. close(int fd)
b. Closes connection
c. fd = socket created by step 1
d. returns number of 0 on success, -1 on failure



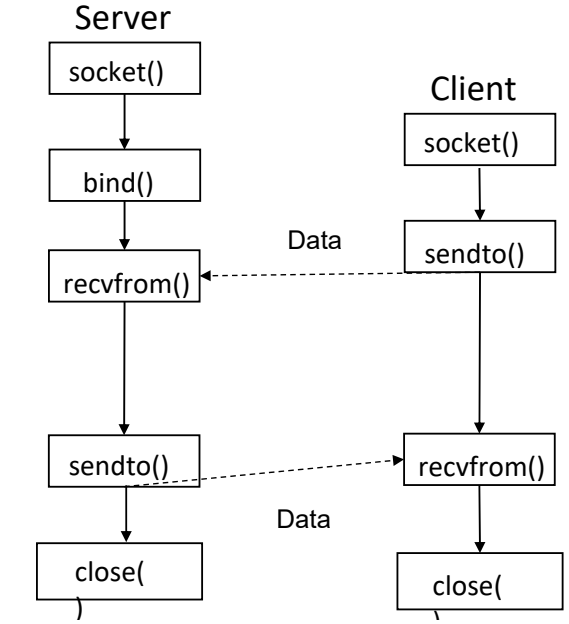
7.Server closes socket using close()

a. close(int fd)
b. Closes connection
c. fd = socket created by step 1
d. returns number of 0 on success, -1 on failure

UDP Sockets

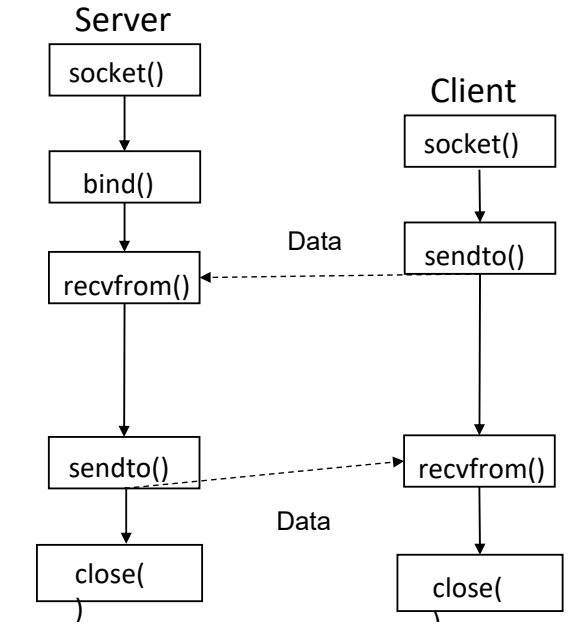
- Used in Lab 3
- Communication by datagram
- Server binds to socket and waits for incoming messages
- Client directly sends messages to a server's socket

UDP Server

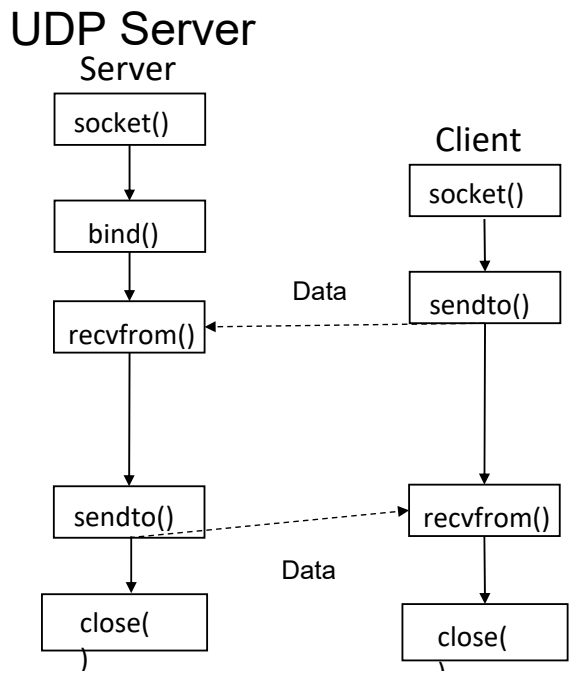


1. Server creates socket using socket()
   a. socket(int domain, int type, int protocol)

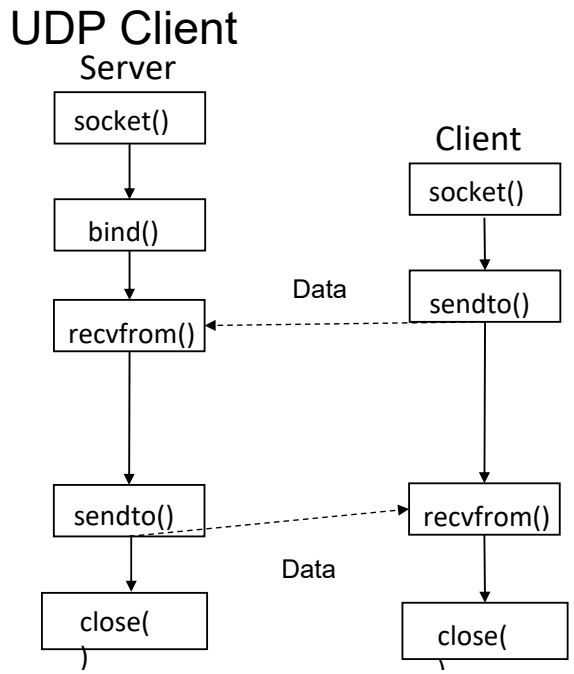
UDP Client



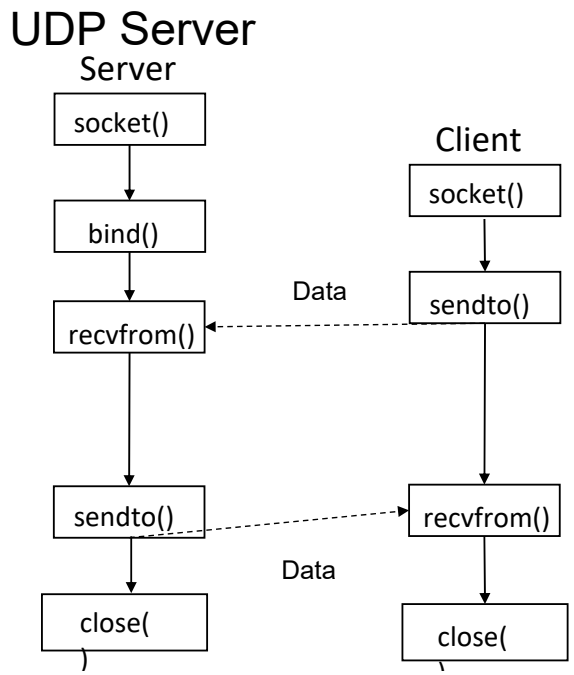
1. Client creates socket using socket()
   a. socket(int domain, int type, int protocol)



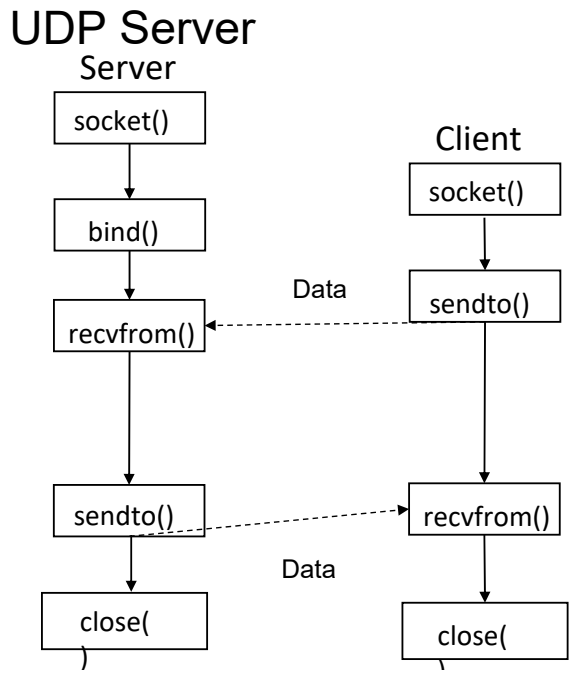
2. Server binds to port using bind()
    a. bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)
    b. Allows server to bind to a port
    c. sockfd = socket created in step 1
    d. addr = struct containing server information
    e. addrlen = sizeof(serverAddr);
    f. returns 0 on success, -1 on failure



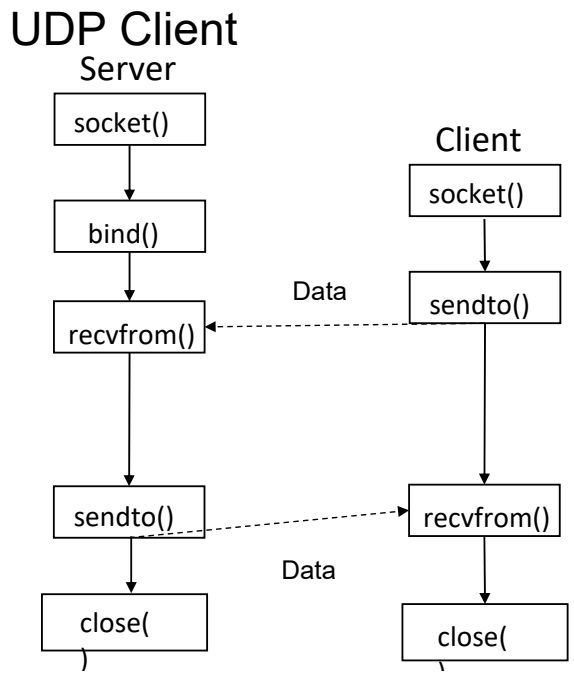
2. Client sends data to server
    a. sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)
    b. sockfd = socket created in step 1
    c. buf = char buffer to send data from data into
    d. len = bytes of buffer to send
    e. flags = options to receive data (0 for lab 3)
    f. dest_addr = struct of server address
    g. addrlen = sizeof(dest_addr)
    h. returns number of bytes sent on success, -1 on failure



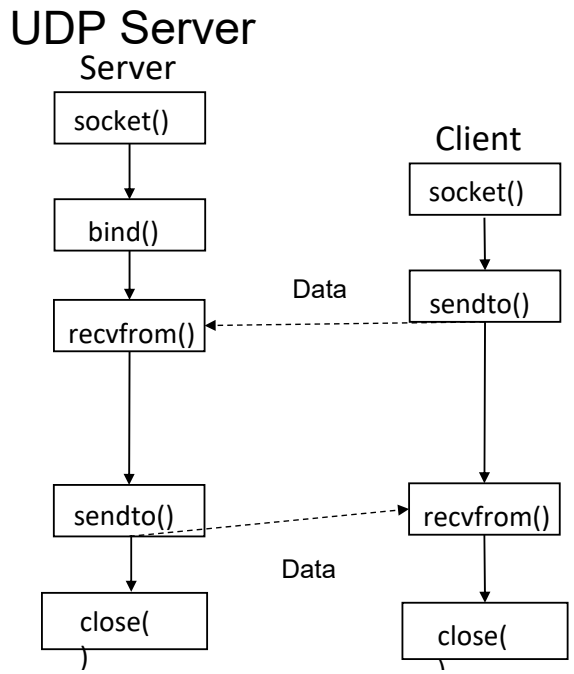
3. Server receives data from client(recvfrom is blocking)
    a. recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)
    b. sockfd = socket created in step 1
    c. buf = char buffer to receive data into
    d. len = size of char buffer
    e. flags = options to receive data (0 for lab 3)
    f. src_addr = struct to be populated with source address (for lab 3 use blank sockaddr_in struct)
    g. addrlen = sizeof(src_addr)
    h. returns number of bytes received on success, -1 on failure



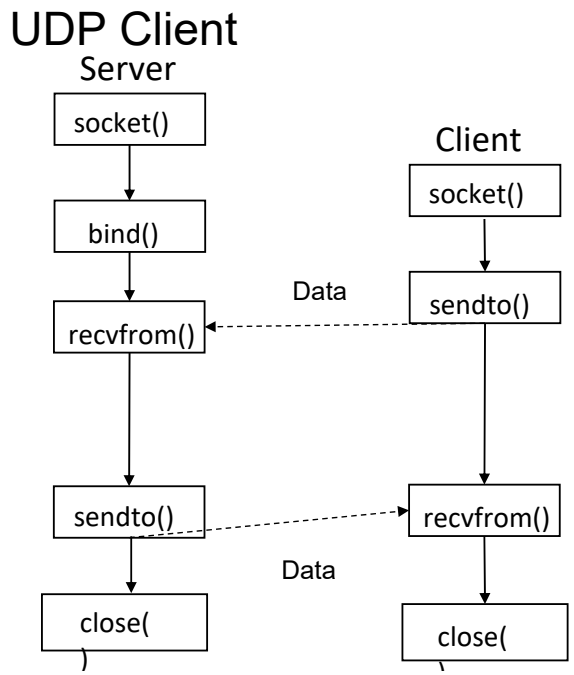
4. Server sends data to client
    a. sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)
    b. sockfd = socket created in step 1
    c. buf = char buffer to send data from data into
    d. len = bytes of buffer to send
    e. flags = options to receive data (0 for lab 3)
    f. dest_addr = struct of client address populated in recvfrom()
    g. addrlen = sizeof(dest_addr)
    h. returns number of bytes sent on success, -1 on failure



3. Client receives data from server(recvfrom is blocking)
    a. recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)
    b. sockfd = socket created in step 1
    c. buf = char buffer to receive data into
    d. len = size of char buffer
    e. flags = options to receive data (0 for lab 3)
    f. src_addr = server address struct
    g. returns number of bytes received on success, -1 on failure



5. Server closes socket
    a. Closes socket connection
    b. close(int fd)
    c. fd = socket to close
    d. returns number of 0 on success, -1 on failure



4. Client closes socket
    a. Closes socket connection
    b. close(int fd)
    c. fd = socket to close
    d. returns number of 0 on success, -1 on failure

Helpful Tips

- A TCP server accept() returns a socket, this should be used for communication with a client
- Use error checking on socket api calls
    - if(socket(PF_INET,SOCK_STREAM,0)<0) { perror("Socket Creation Failure: "); return -1;}
- Make sure the correct number of bytes are being sent from the buffer
- Don't forget to format IP addresses and port numbers using inet_addr() and htons()
- Make sure all data structures and buffers are 0 filled using memset or other method
- Use correct values for length on socket api calls
- Make sure you are using the correct inputs to socket api calls
    - Pointers vs values