TA: Xifan Zheng

Email: zhengxifan0403@gmail.com

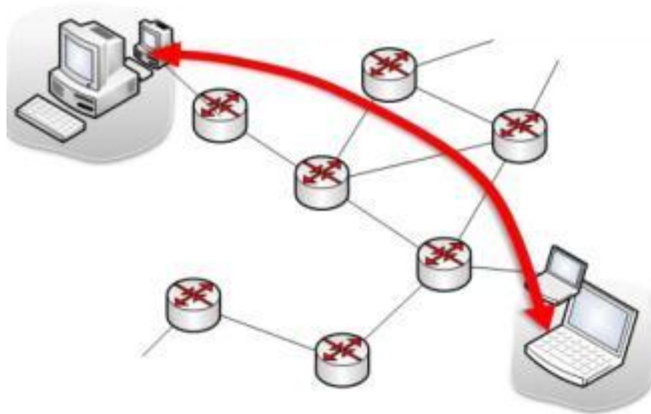
Welcome to CPSC 441!

Today's Tutorial

- Introduction to socket
- Address/port
- Socket creation
- Set up connection
- Communication through socket
- Example server/client
- Hint for assignment1

What is a socket?

- Socket is an interface between application and network (the lower levels of the protocol stack)
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - reliable vs. best effort
    - connection-oriented vs. connectionless



- electric outlet that one can plug into for network services

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

What is a socket?

- A host-local, application-created, OS-controlled interface (a “door”) into application process
- Once a socket is setup the application can
  - pass data to the socket for network transmission
  - receive data from the socket (transmitted through the network, received from some other host)

Most popular types of sockets

- **TCP socket**
  - Type: **SOCK_STREAM**
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional

We focus on TCP

- **UDP socket**
  - Type: **SOCK_DGRAM**
  - unreliable delivery
  - no order guarantees
  - no notion of "connection" – app indicates destination for each packet
  - can send or receive

Server and clients

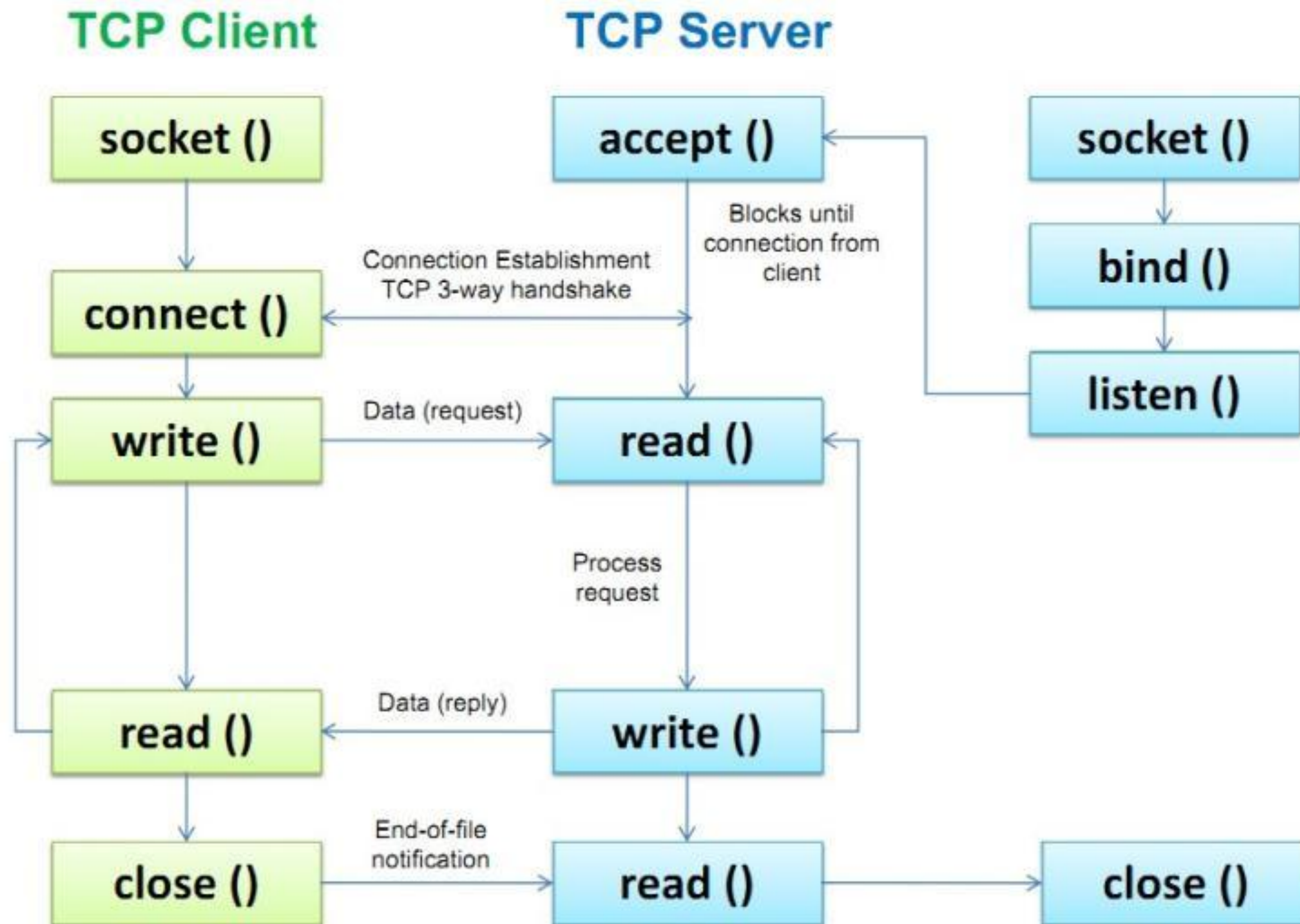
Typical TCP client:

1. Create a TCP socket using socket()
2. Establish a connection to server using connect()
3. Communicate using send() and recv()
4. Close the connection with close()

Typical TCP server:

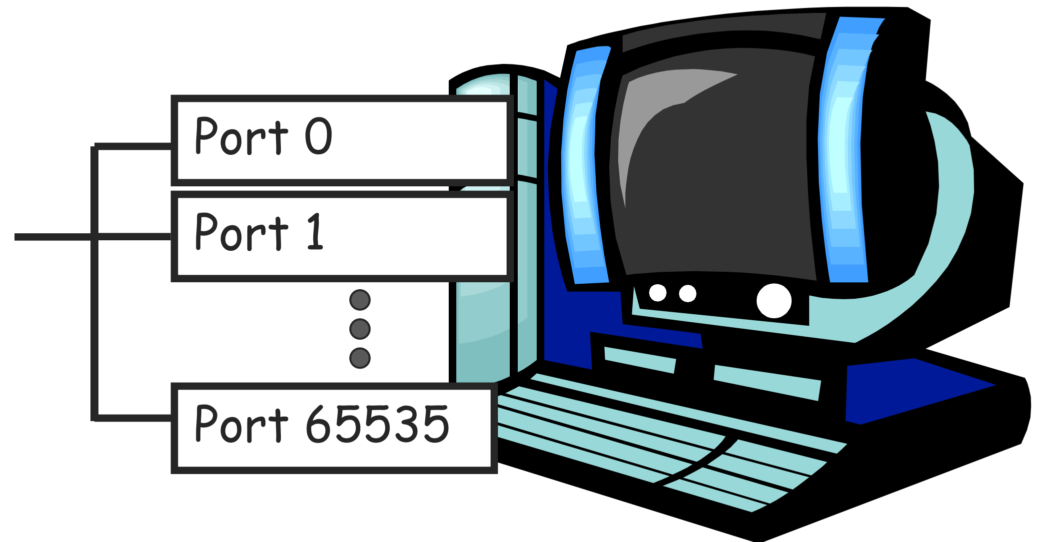
1. Create a TCP socket using socket()
2. Assign a port number to the socket with bind()
3. Tell the system to allow connections to be made to that port using listen()
4. Repeatedly do the following:
   call accept() to get a new socket for each client connection
   communicate with the client via that new socket using send () and recv()
   Close the client connection using close()

Server and clients



Ports

- Each host machine has an IP address (or more!)
- Each host has 65,536 ports (2^{16})
- Some ports are *reserved* for specific apps
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP
  - see RFC 1700 (about 2000 ports are reserved)



A socket provides an interface to send data to/from the network through a port

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

Addresses, Ports and Sockets

- Like apartments and mailboxes
  - You are the application
  - Your apartment building address is the address
  - Your mailbox is the port
  - The post-office is the network
  - The socket is the key that gives you access to the right mailbox (one difference: assume outgoing mail is placed by you in your mailbox)

- Q: How do you choose which port a socket connects to?

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

SOCKET CREATION IN C

- int s = socket(domain, type, protocol);
  - s: socket descriptor, an integer (like a file-handle)
  - domain: integer, communication domain
    - e.g., AF_INET (IPv4 protocol) – typically used
  - type: communication type
    - SOCK_STREAM: reliable, 2-way, connection-based service
    - SOCK_DGRAM: unreliable, connectionless,
    - other values: need root permission, rarely used, or obsolete
  - protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0, 0 is for IP

NOTE: socket call does not specify where data will be coming from, nor where it will be going to - it just creates the interface.

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

The bind function

- The bind function associates and (can exclusively) reserves a port for use by the socket
- int status = bind(sockid (struct sockaddr *) &servaddr, size);
  - status: error status, = -1 if bind failed
  - sockid: integer, socket descriptor
  - Sockaddr: the structure with the addresses and t he ports
  - size: the size (in bytes) of the servaddr structure

The bind function

- The sockaddr is the structure that is defined as a "container" for specifying the address and port.
  - ✓ **servaddr.sin_family = AF_INET; /* IPv4 protocol */**
  - ✓ **servaddr.sin_addr.s_addr = htonl(INADDR_ANY);**
    /*any incoming interface in server*/

    htonl() convert host IP address to network long address (Host to network long)
  - ✓ **servaddr.sin_port = htons(13);**
    /*well-known daytime port*/

    htons() convert host IP address to network short address (Host to network short)

Failed to bind?

Bind() can be failed, When and why?

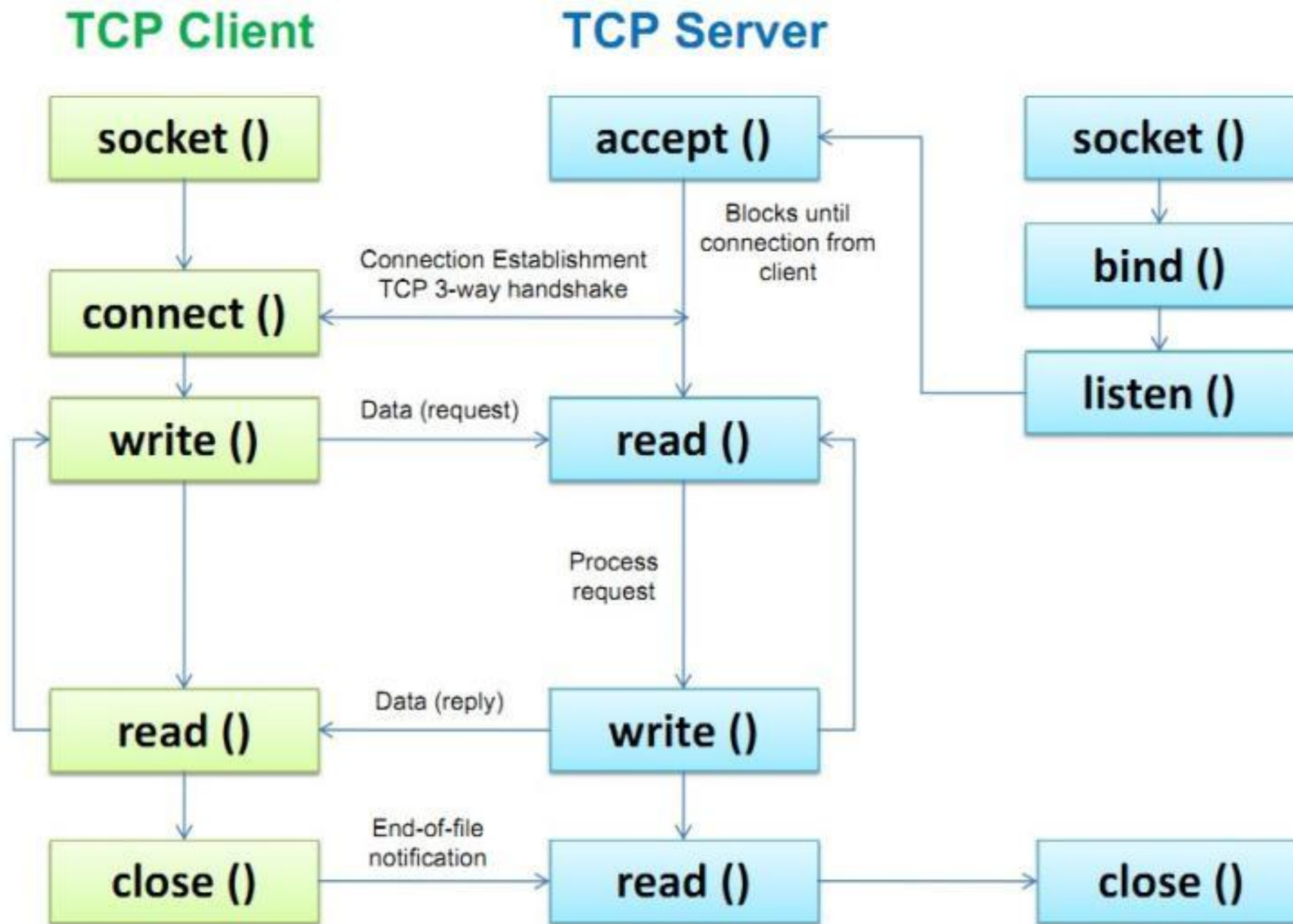
1. some other socket is already bound to the specified port

2. On some systems special privileges are required to bind to certain ports (typically those with numbers less than 1024)

Connection Setup

- A connection occurs between two ends
  - Server: waits for an active participant to request connection (listen)
  - Client: initiates connection request to passive side
- Once connection is established, server and client ends are “similar”
  - both can send & receive data
  - either can terminate the connection

Server and clients



SERVER SOCKET: LISTEN & ACCEPT

Called on server side:

- int status = listen(sock, queuelen);
  - status: 0 if listening, -1 if error
  - sock: integer, socket descriptor
  - queuelen: integer, # of active participants that can "wait" for a connection
  - listen is **non-blocking**: returns immediately
  - Before call to listen(), any incoming connection requests will be rejected

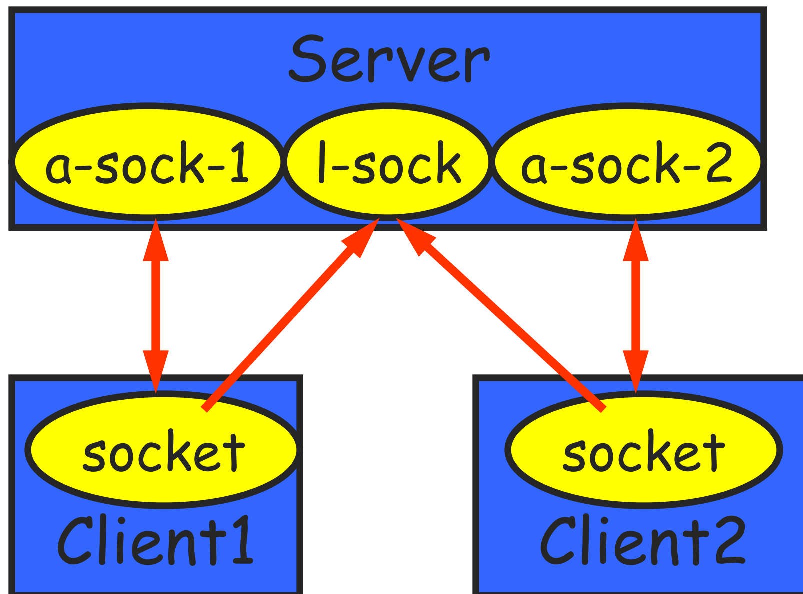
- int s = accept(sock, (struct sockaddr *) cliAddr, cliAddrLen);
  - s: integer, the new socket (used for data-transfer)
  - sock: integer, the orig. socket (being listened on)
  - struct sockaddr, address of the connected client will be stored here
  - If so, accept() returns a NEW SOCKET DESCRIPTOR ! Why ? Because the ol d socket descriptor (sock) is still queuing request from the network !
  - accept is **blocking**: blocks until an incoming connection is made to the listening socket's port number, then return a descriptor

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

Connection setup steps

- Client end:
  - step 2: request & establish connection
  - step 4: send/recv

- Server end:
  - step 1: listen (for incoming requests)
  - step 3: accept (a request)
  - step 4: send/recv



- The accepted connection is on a new socket

- The old socket continues to listen for other active participants

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

CONNECT

- int status = connect(sock, (sockaddr *) &servaddr, sizeof(servaddr));
  - status: 0 if successful connect, -1 otherwise
  - sock: integer, socket to be used in connection
  - servaddr :address of passive participant
  - sizeof(servaddr): integer

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

Sending / Receiving Data

- int count = send(sock, &buf, len, flags);
  - count: # bytes transmitted (-1 if error)
  - buf: char[], buffer to be transmitted
  - len: integer, length of buffer (in bytes) to transmit
  - flags: integer, special options, usually just 0
- int count = recv(sock, &buf, len, flags);
  - count: # bytes received (-1 if error)
  - buf: void[], stores received bytes
  - len: # bytes received
  - flags: integer, special options, usually just 0

- Calls are **blocking** [returns only after data is sent (to socket buf) / received]

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

close

- When finished using a socket, the socket should be closed:

- status = close(s);
  - status: 0 if successful, -1 if error
  - s: the file descriptor (socket being closed)

- Closing a socket
  - closes a connection
  - frees up the port used by the socket

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

The struct sockaddr

- The Internet-specific:

```
struct sockaddr_in {
    short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

  - sin_family = AF_INET // Specifies the address family
  - sin_port: // Specifies the port #(0-65535)
  - sin_addr: // Specifies the IP address
  - sin_zero: unused // unused!

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

Lets write some code!

- Sample socket program:
  - Client/server example.

FAQ 1

- Sometimes, an ungraceful exit from a program (e.g., ctrl-c) does not properly free up a port
- Eventually (after a few minutes), the port will be freed
- You can kill the process, or
- To reduce the likelihood of this problem, include the following code:
  - In header include:

```
#include <signal.h>
void cleanExit(){exit(0);}
```

  - In socket code:

```
signal(SIGTERM, cleanExit);
signal(SIGINT, cleanExit);
```

From: http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

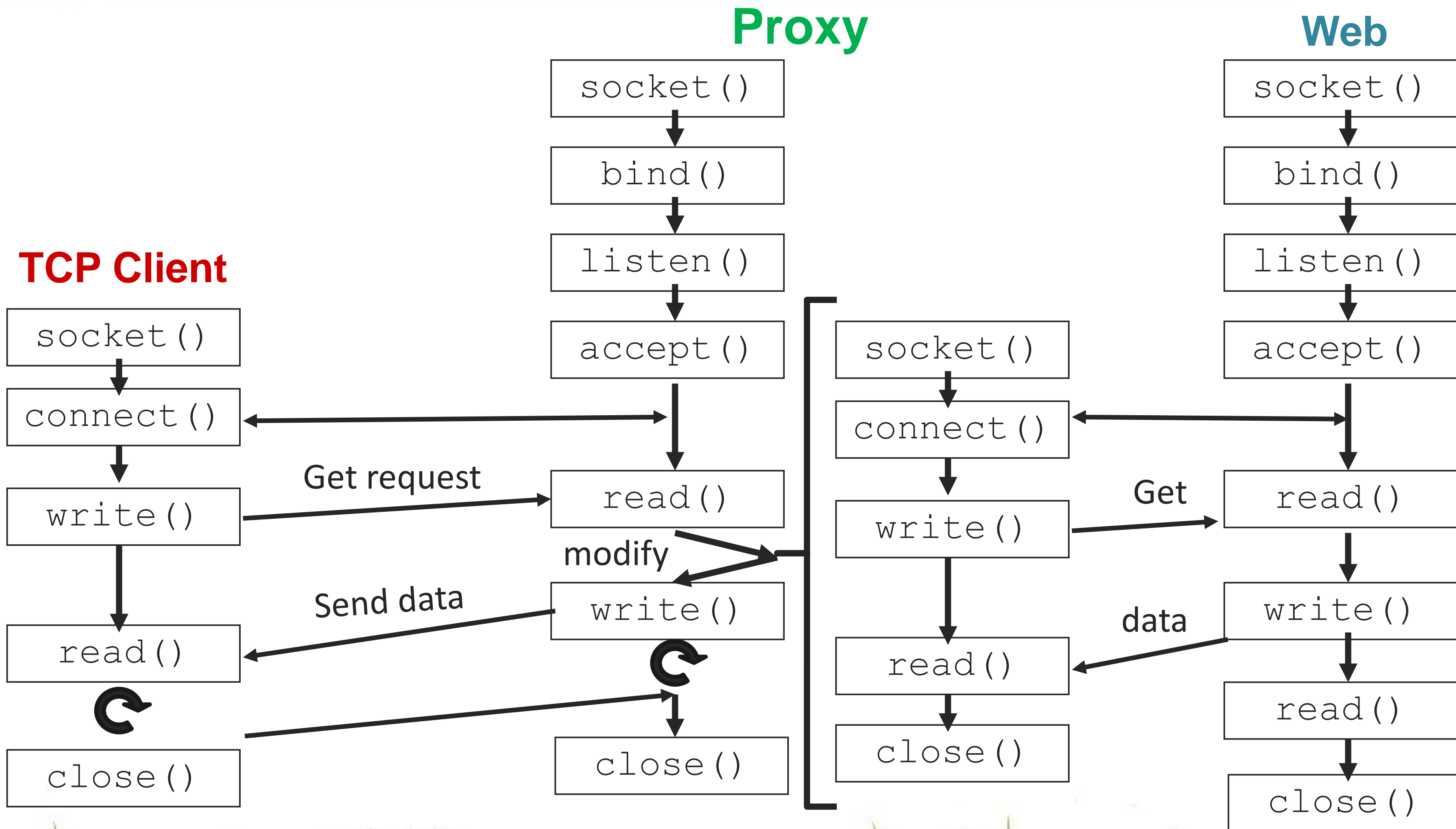
FAQ 2

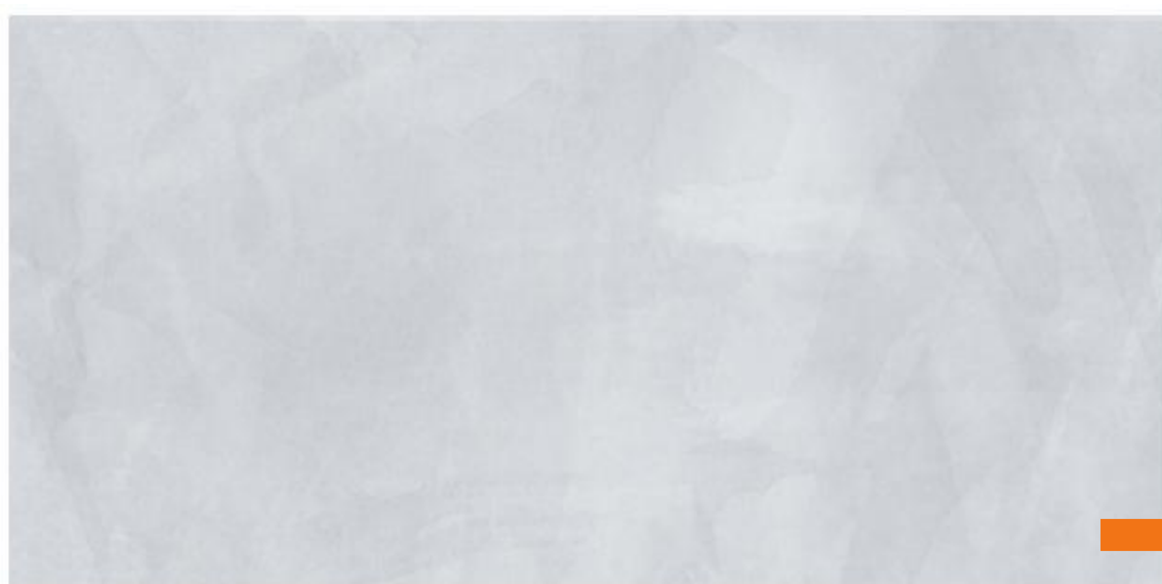
- Make sure to #include the header files that define used functions
  - Check Beej's Guide to Network Programming Using Internet Sockets
    http://beej.us/guide/bgnet/output/html/multipage/index.html
- Search the specification for the function you need to use for more info, or check the main pages.

references

- These are good references for further study of Socket programming with C:
  - Beej's Guide to Network Programming Using Internet Sockets
    http://beej.us/guide/bgnet/output/html/multipage/index.html
  - http://www.cs.columbia.edu/~danr/courses/6761/Summer03/intro/6761-1b-sockets.ppt

Tips for the assignment 1





Thanks for attending!