Socket Programing
In Linux - C

Introduction

- The goal of this presentation is to introduce you with concepts of socket programming.

  “ How network sockets works? “

- You are expected to explore further by yourself.

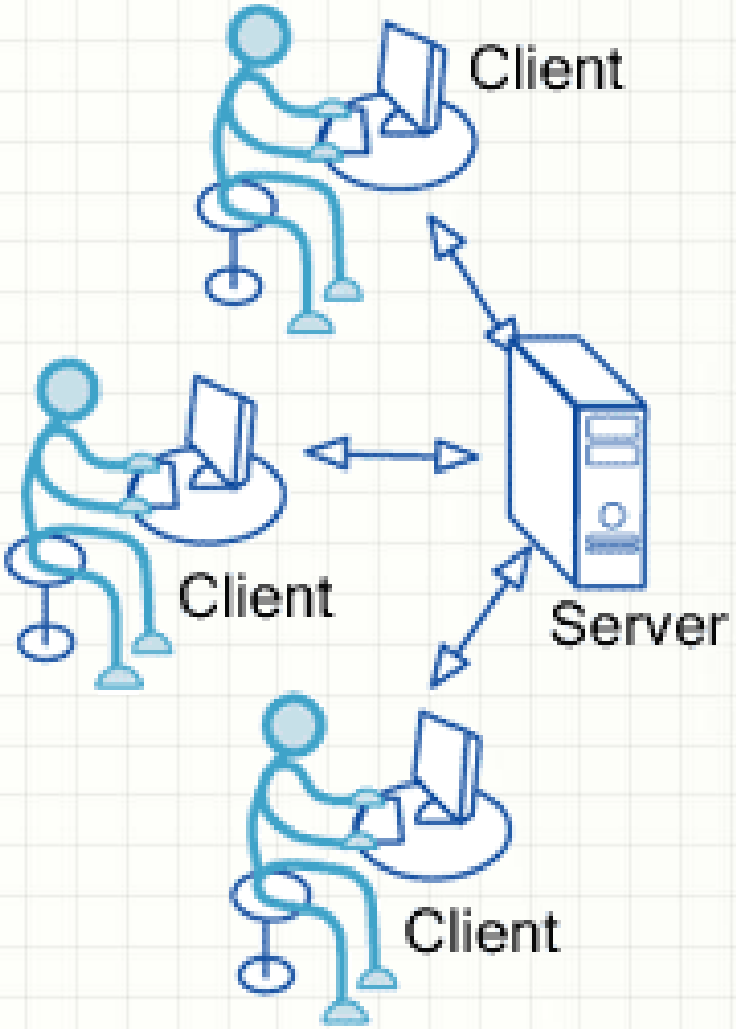
Today's Goals:

1. High-level Overview
2. Familiarize yourself with socket API
3. A simple socket connection scheme
4. Socket management
5. Multicast Socket
6. Introducing the programming assignment

High-level Overview

Client – Server

- Clients
  - Locates the server
  - Initiate connection
- Server:
  - Responder
  - Provides service



Client – Server: some key differences

Clients

- Simple
- (Usually) sequential
- Not performance sensitive
- Execute on-demand

Server

- Complex
- (Massively) concurrent
- High performance
- Always-on

Client – Server: Similarities

- Share common protocols
  - Application layer
  - Transport layer
  - Network layer
- Both rely on APIs for network access

What is API?

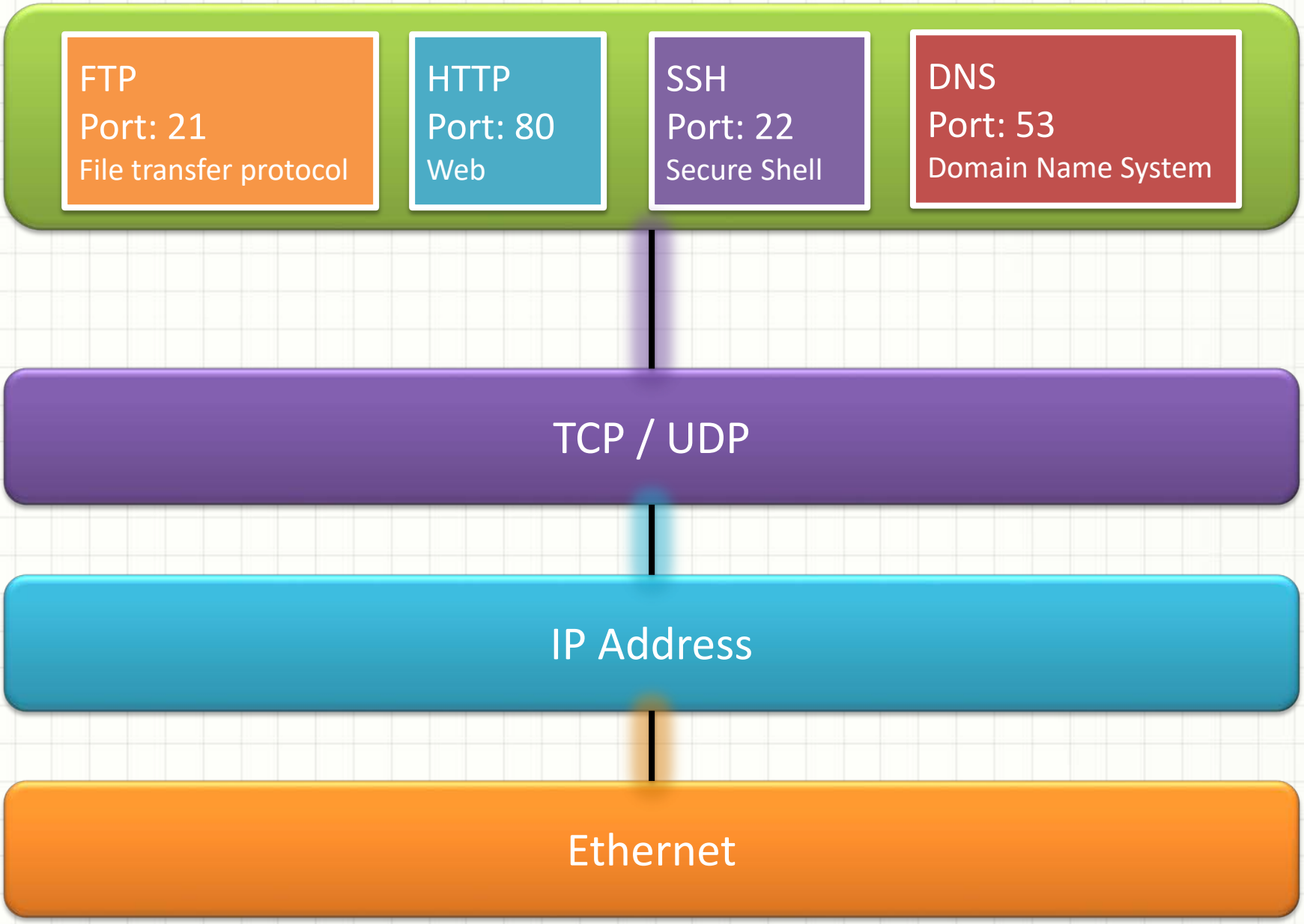
application programming interface (**API**) is a set of routines, protocols, and tools for building software applications

What is a network socket?

- It is an application Mailbox for the network messages.
- Implements an Incoming and Outgoing queues.
- Managed by the operating system.
- Represented as file descriptor.
- Used to pass messages among applications on different computers.
- Identified by IP address and port.

Addresses, Ports and Sockets

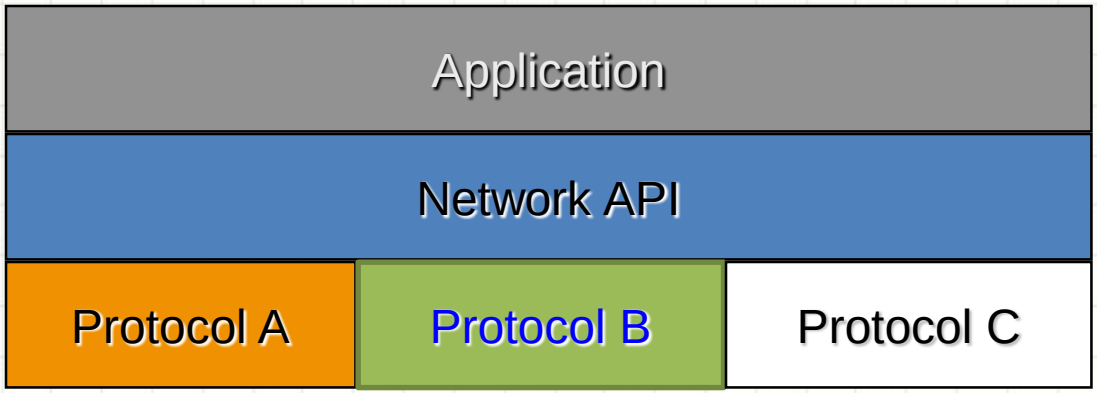
- Like apartments and mailboxes:
  - You are the application
  - Your apartment address is the IP address
  - Your mailbox is the port
  - The post-office is the network
  - Your postman is the operating system
  - The socket is the key that gives you access to the right mailbox


FTP
Port: 21
File transfer protocol
HTTP
Port: 80
Web
SSH
Port: 22
Secure Shell
DNS
Port: 53
Domain Name System
TCP / UDP
IP Address
Ethernet

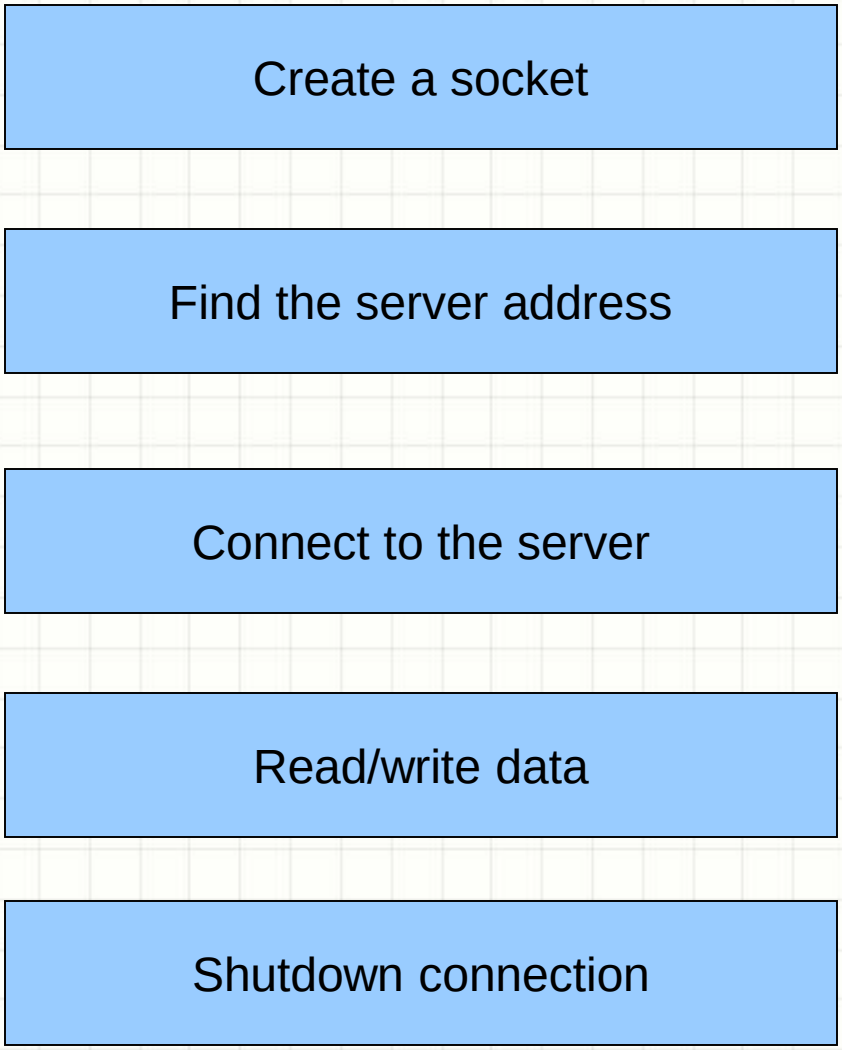
Socket API

Introduction

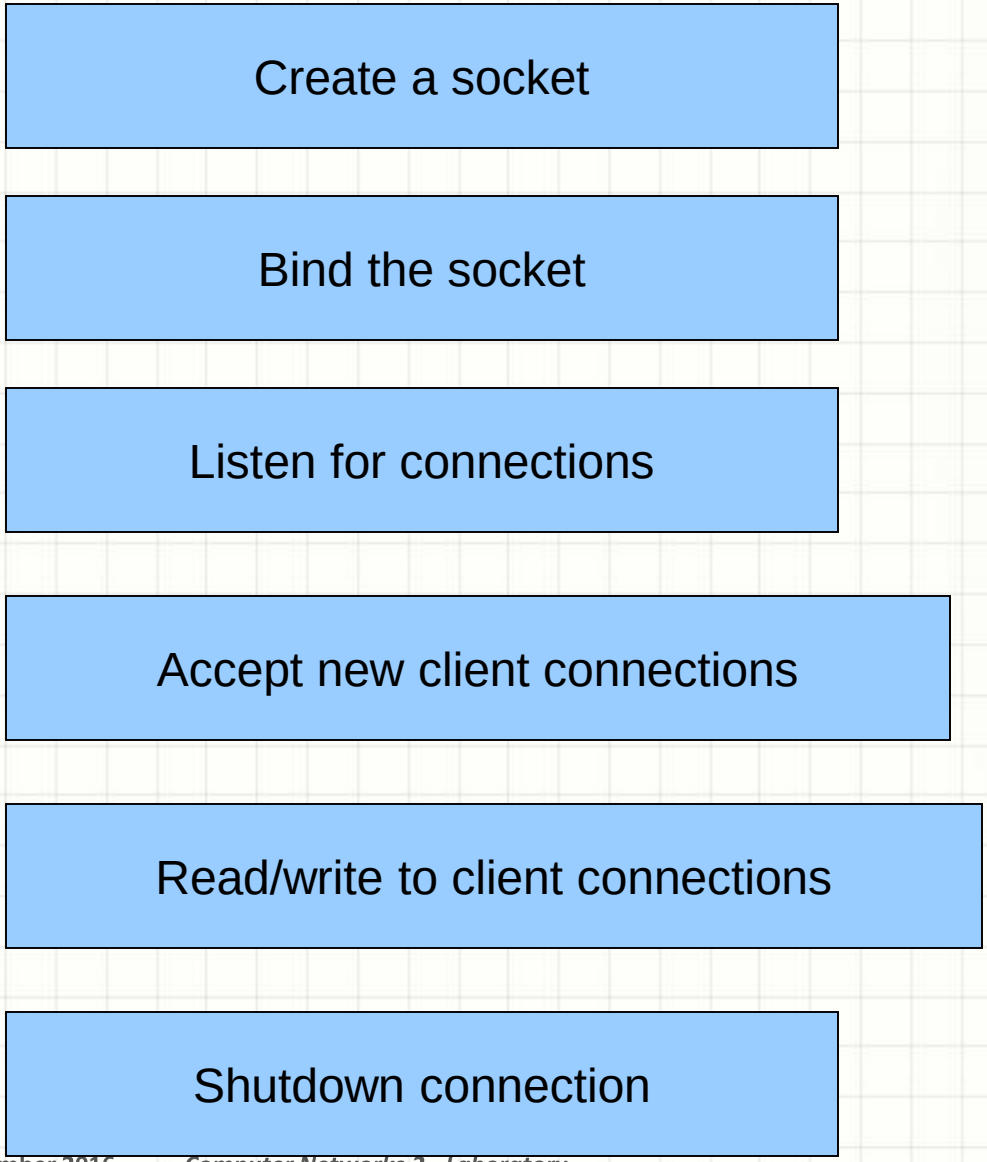
- A **socket API** is an application programming interface (API), provided by the operating system, that allows application programs to control and use network sockets.



Client


Create a socket
Find the server address
Connect to the server
Read/write data
Shutdown connection

Server


Create a socket
Bind the socket
Listen for connections
Accept new client connections
Read/write to client connections
Shutdown connection

socket: creates a socket of a given domain, type, protocol (buy a phone)
http://linux.die.net/man/2/socket

TCP: socket(AF_INET, SOCK_STREAM, 0)
UDP: socket(AF_INET, SOCK_DGRAM, 0)

bind: assigns a name to the socket (get a telephone number)
http://linux.die.net/man/2/bind

TCP: bind(lisen_sock, (struckt sockaddr *)&listen_addr, sizeof(listen_addr)

listen: specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)
http://linux.die.net/man/2/listen

TCP: listen(lisen_sock, SOMAXCONN)

```
struct sockaddr_in{
    uint8_t         sin_len;    /*length of the structure (16)*/
    sa_falimily_t   sin_family /* AF_INT*/
    in_port_t       sin_port    /* 16 bit TCP or UDP port number*/
    struct in_addr  sin_addr    /* 32 bit Ipv4 address */
    char            sin_zero(8)/* unused*/
}
```

accept: server accepts a connection request from a client (answer phone)
http://linux.die.net/man/2/accept

TCP: accept(lisen_sock, (struct sockaddr*) &client_addr, &client_addr_size)

connect: client requests a connection request to a server (call)
http://linux.die.net/man/2/connect

TCP: connect(server_sock, (struct sockaddr*) &server_addr, sizeof(server_addr))

send, sendto: write to connection (speak)
http://linux.die.net/man/2/send

TCP/UDP: send(sock, buffer, &buff_len, 0)

recv, recvfrom: read from connection (listen)
http://linux.die.net/man/2/recv

TCP/UDP: recv(sock, buffer, &buff_len, 0)

gethostbyname: find host's address (lookup in phone book)
http://linux.die.net/man/3/gethostbyname

TCP/UDP: struct * = gethostbyname(char *)

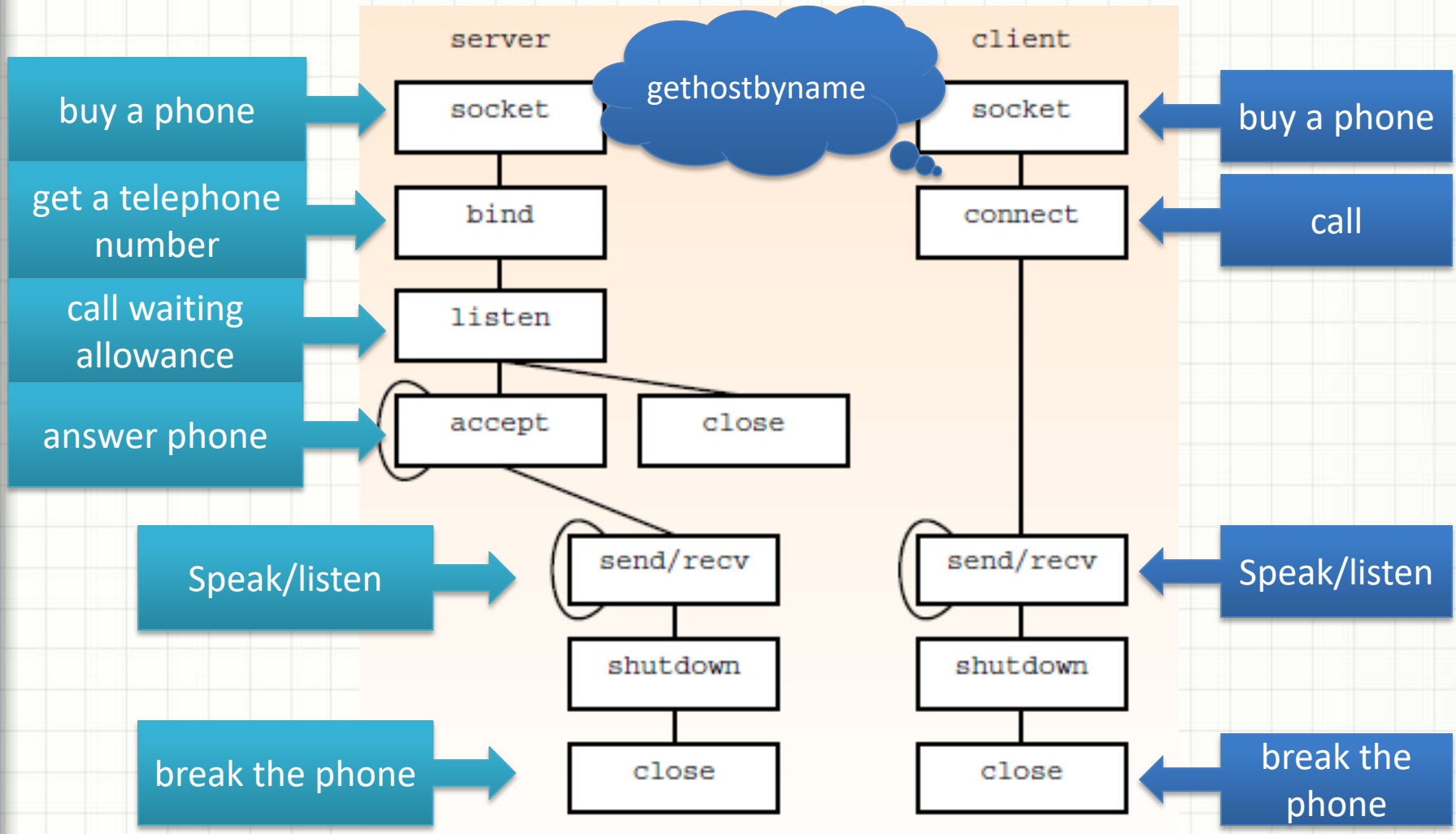
htons(), htonl(), ntohs(), ntohl(): convert to/from network address
http://linux.die.net/man/3/htons

close: close the socket (break the phone)
http://linux.die.net/man/2/close

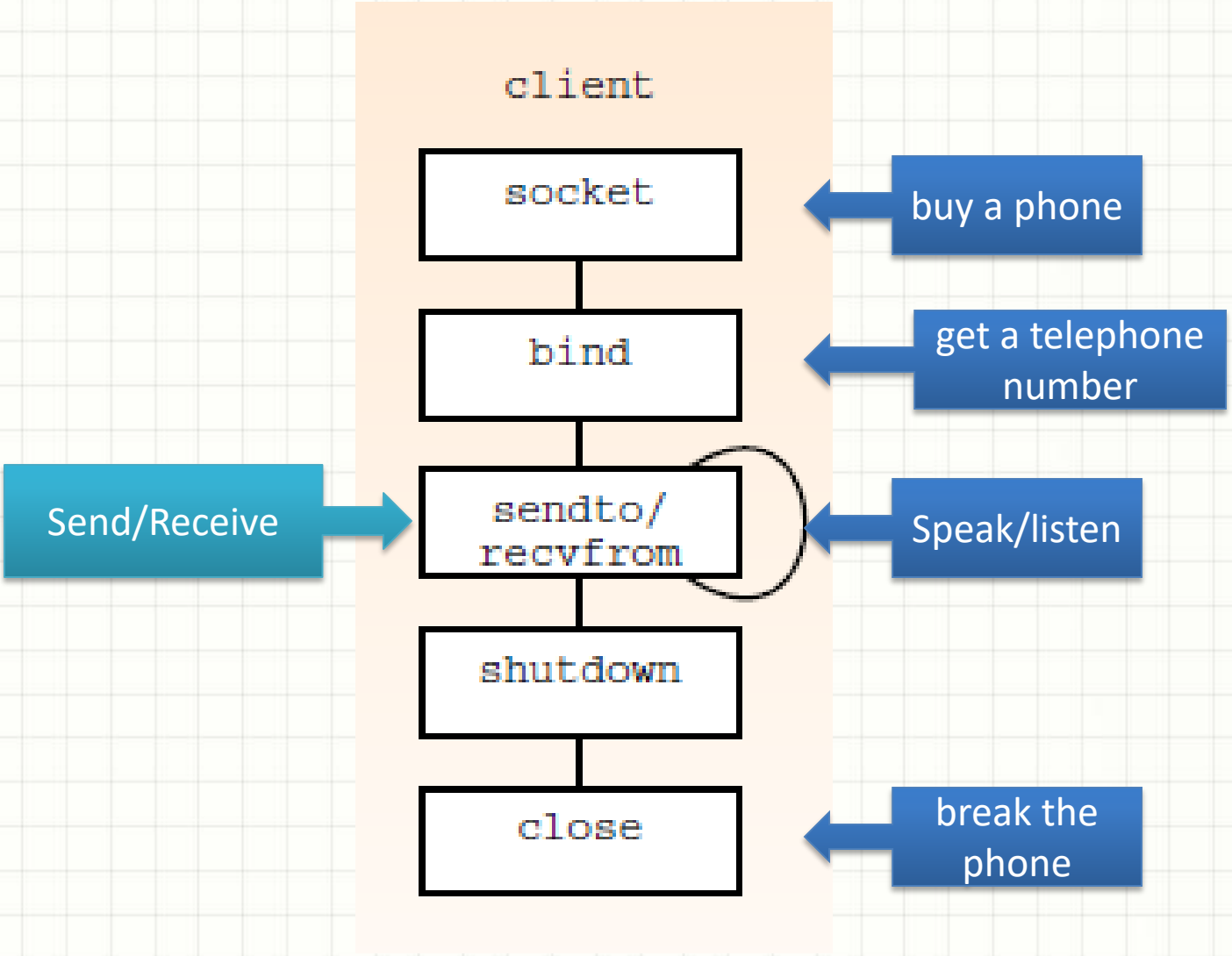
TCP/UDP: close(socket)

CONNECTION SCHEME

TCP


server
client
gethostbyname
socket
socket
bind
connect
listen
accept
close
send/recv
send/recv
shutdown
shutdown
close
close
buy a phone
get a telephone number
call waiting allowance
answer phone
buy a phone
call
Speak/listen
Speak/listen
break the phone
break the phone

UDP


client
socket
bind
sendto/
recvfrom
shutdown
close
buy a phone
get a telephone number
Speak/listen
break the phone
Send/Receive

SOCKET MANAGEMENT

Socket management methods

- Parallel: Threads / process
- Serial: select function

select: waiting for sockets change
http://linux.die.net/man/2/select

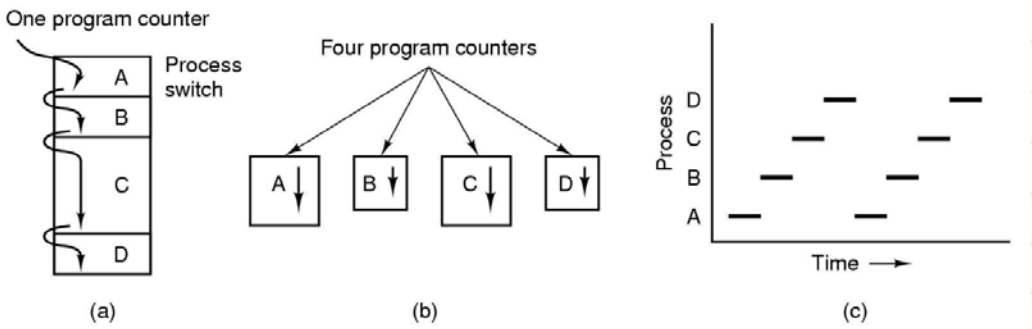
TCP/UDP: int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

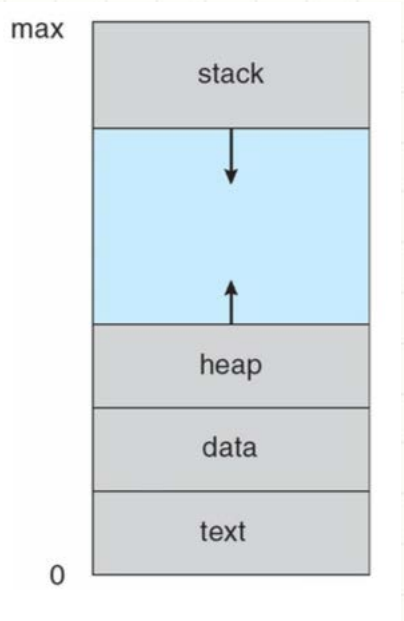
We define a set of sockets which in turn will be handled.

void FD_CLR(int *fd*, fd_set **set*);
int FD_ISSET(int *fd*, fd_set **set*);
void FD_SET(int *fd*, fd_set **set*);
void FD_ZERO(fd_set **set*);

Processes and Threads

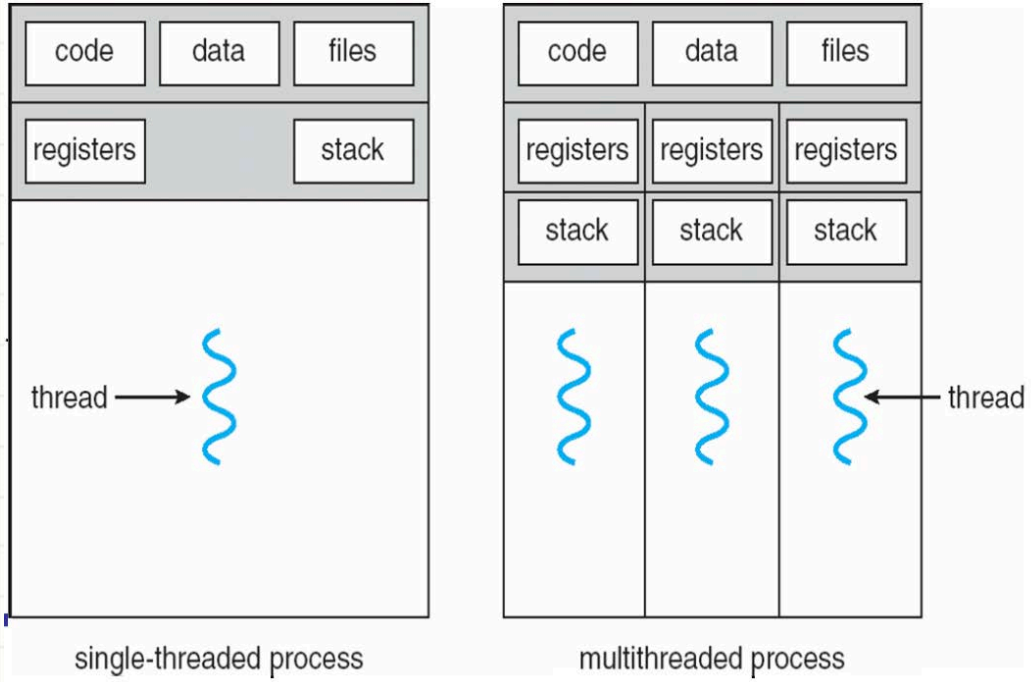
- A process is a program that runs sequentially
- When we have several processes





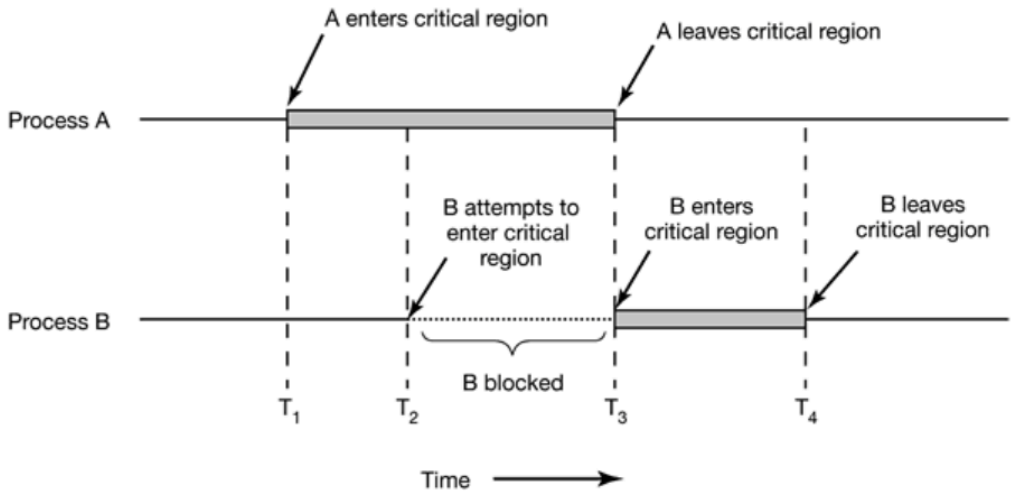
- Each process has
  - address space – code, variables and etc.
  - Single thread of control

- Multithreading
  - it is desirable to have multiple threads of control
  - in the same address space
  - running in quasi-parallel



- No need in context switch

Mutual Exclusion using Critical Regions



Mutual exclusion using critical regions

- Mutex: a variable that can be in one of two states
  - Unlocked
  - locked

Multicast Socket

Using Multicast capabilities

- Multicast source – Only need to send multicast packets with multicast address as destination

```
sock = socket(AF_INET, SOCK_DGRAM, 0);
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("239.0.0.1");
addr.sin_port = htons(6000);
sendto(sock, message, sizeof(message), 0, (struct sockaddr *) &addr, sizeof(addr));
```

- The network does all the work (PIM)

Using Multicast capabilities

- Multicast reciever – need to subscribe to the multicast group (tell the kernel which multicast groups you are interested in)

```
struct ip_mreq mreq;

sock = socket(AF_INET, SOCK_DGRAM, 0);
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = htonl(INADDR_ANY);
addr.sin_port = htons(6000);

bind(sock, (struct sockaddr *) &addr, sizeof(addr))
mreq.imr_multiaddr.s_addr = inet_addr("239.0.0.1");
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
setsockopt(sock, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq))

recvfrom(sock, message, sizeof(message), 0, (struct sockaddr *) &addr, &addrlen);
```

IP_DROP_MEMBERSHIP

- The network does all the work (IGMP, PIM)

Final Project:
Internet Radio Application

Radio over Multicast

- Implemented over the multicast lab
- Server
  - Multicast stations: each station a different multicast group, all use same port (UDP)
  - Sends info to clients (TCP)
- Two Clients
  - Listener: a simple UDP and TCP receiver
  - Controller: talk to server to find info (using TCP) and inform Listener (using TCP).

Internet Radio Application

- You will implement 3 programs, step by step:
  1. Client: Listener
  2. Client: Controller
  3. Server
- We will provide binaries of all 3 of them so you can check your programs one by one

Client: Listener

- UDP socket
- Receives audio packets from the server
- TCP socket
- Receives info commands from the Client Controller

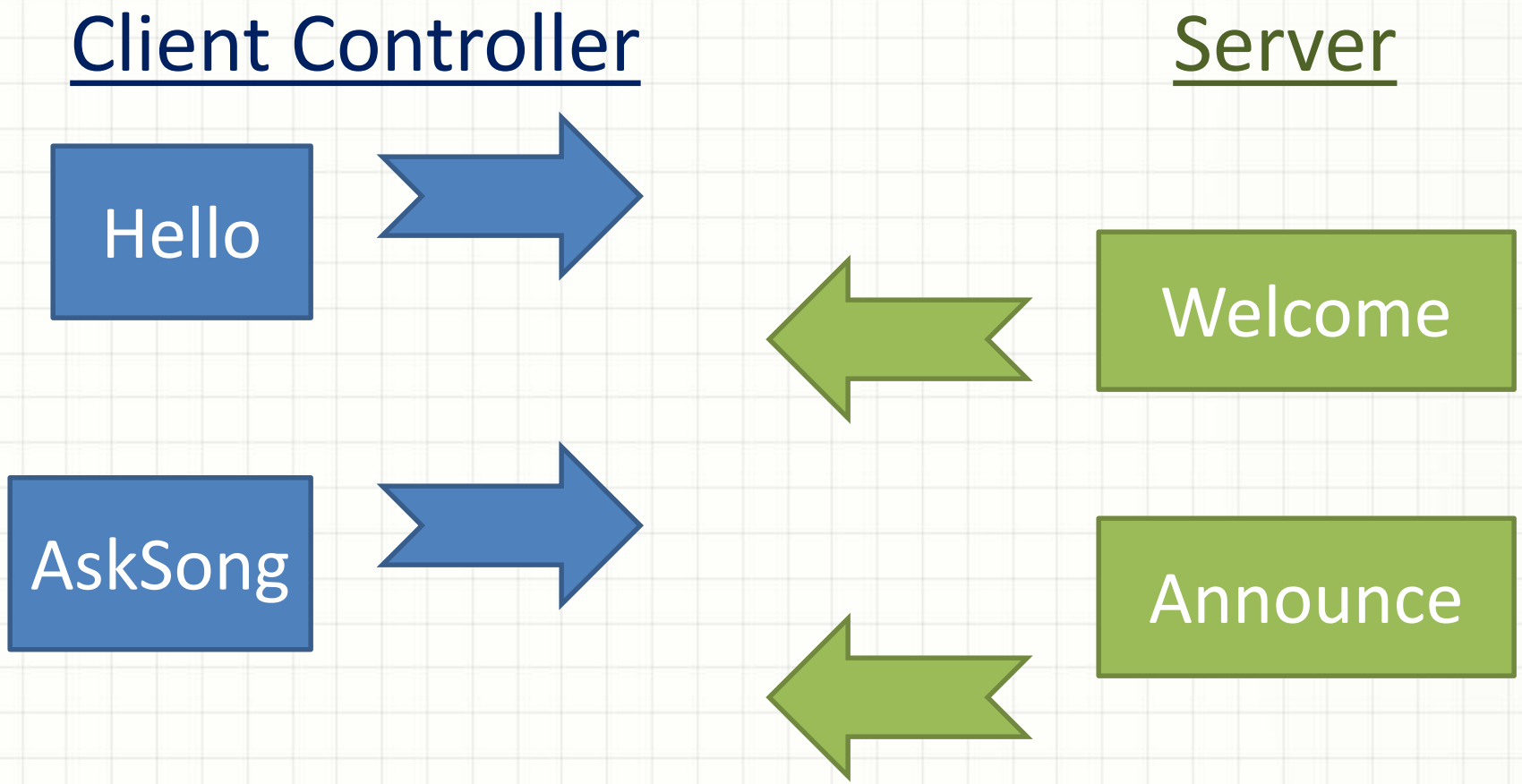
Client: Controller

- Two TCP sockets
- User input
- The user interface with the server
- Handsake with server to learn the multicast address
- Can ask about songs
- Can ask for listeners list
- Inform Listener for the proper song

Server

- TCP and UDP socket
- Maintains control connections with listeners
- Transmits stations via UDP to multicast
- Maintains several stations
- Important: Rate control

Server – Client Connection Protocol



Server – Client Connection Protocol

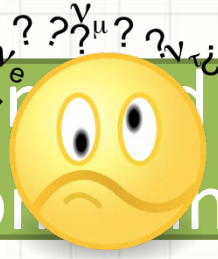
Client Controller

Server

AskList

%@^!

Listeners list



Client – Client Connection Protocol

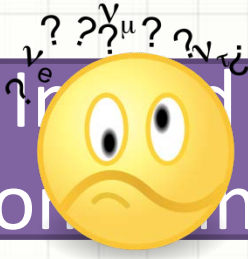
Client Controller

Client Listener

Inform Station

%@^!

Ack



Thanks to…

- Wikipedia
- http://www.ccs.neu.edu/home/cbw/4700/slides/4_C_Sockets.pptx
- http://www.cs.northwestern.edu/~agupta/cs340/sockets/sockets_intro.ppt
- http://parsys.eecs.uic.edu/~solworth/sockets.pdf