# Server-side Programming

CSE 333 Spring 2021

**Instructor:** Justin Hsia, Travis McGaha

**Teaching Assistants:**

| | | |
|---|---|---|
| Arthava Deodhar | Callum Walker | Cosmo Wang |
| Dylan Hartono | Elizabeth Haker | Eric Marnadi |
| Kyrie Dowling | Leo Liao | Markus Schiffer |
| Neha Nagvekar | Nonthakit Chaiwong | Ramya Challa |

Poll Everywhere

pollev.com/cse333travis

# Which concept has given you the most difficulty so far in the context of Homework 3?

A. Understanding the index file layout
B. C++ Classes & Inheritance
C. C++ STL
D. Query Processor Algorithm
E. Debugging/GDB
F. Style considerations
G. Prefer not to say

# Administrivia

- Exercise 10 released Wednesday, due Monday (5/24)
  - Client-side programming
- Exercise 11 released today, due Wednesday (5/26)
  - Server-side programming

- hw4 will be pushed & released today or tomorrow
  - Due last Thursday of quarter (6/3)
  - Demo today (at end of lecture)

# Socket API: Server TCP Connection

- Pretty similar to clients, but with additional steps:
  1) Figure out the IP address and port on which to listen
  2) Create a socket
  3) `bind()` the socket to the address(es) and port
  4) Tell the socket to `listen()` for incoming clients
  5) `accept()` a client connection
  6) `read()` and `write()` to that connection
  7) `close()` the client socket

# Servers

- Servers can have multiple IP addresses ("*multihoming*")
  - Usually have at least one externally-visible IP address, as well as a local-only address (127.0.0.1)

- The goals of a server socket are different than a client socket
  - Want to bind the socket to a particular *port* of one or more IP addresses of the server
  - Want to allow multiple clients to connect to the same port
    - OS uses client IP address and port numbers to direct I/O to the correct server file descriptor

# Step 1: Figure out IP address(es) & Port

- Step 1: `getaddrinfo()` invocation may or may not be needed (but we'll use it)
  - Do you know your IP address(es) already?
    - Static vs. dynamic IP address allocation
    - Even if the machine has a static IP address, don't wire it into the code – either look it up dynamically or use a configuration file
  - Can request listen on all local IP addresses by passing `NULL` as `hostname` and setting `AI_PASSIVE` in `hints.ai_flags`
    - Effect is to use address `0.0.0.0` (IPv4) or `::` (IPv6)

# Step 2: Create a Socket

- Step 2: `socket()` call is same as before
  - Can directly use constants or fields from result of `getaddrinfo()`
  - Recall that this just returns a file descriptor – IP address and port are not associated with socket yet

# Step 3: Bind the socket

- ```
  int bind(int sockfd, const struct sockaddr* addr,
           socklen_t addrlen);
  ```
  - Looks nearly identical to **`connect()`**!
  - Returns **`0`** on success, **`-1`** on error

- Some specifics for `addr`:
  - **Address family:** `AF_INET` or `AF_INET6`
    - What type of IP connections can we accept?
    - POSIX systems can handle IPv4 clients via IPv6 ☺
  - **Port:** port in network byte order (**`htons()`** is handy)
  - **Address:** specify *particular* IP address or *any* IP address
    - "Wildcard address" – `INADDR_ANY` (IPv4), `in6addr_any` (IPv6)
  ```

# Step 4: Listen for Incoming Clients

- `int listen(int sockfd, int backlog);`
  - Tells the OS that the socket is a listening socket that clients can connect to
  - `backlog`: maximum length of connection queue
    - Gets truncated, if necessary, to defined constant `SOMAXCONN`
    - The OS will refuse new connections once queue is full until server `accept()`s them (removing them from the queue)
  - Returns `0` on success, `-1` on error
  - Clients can start connecting to the socket as soon as `listen()` returns
    - Server can't use a connection until you `accept()` it

# Psuedocode exercise

- The plan for this question
  - Send everyone to breakouts
  - 1 person per breakout shares their screen and goes to the shared google slides here:
    - Lecture A: https://docs.google.com/presentation/d/1RbnVbZC0G-529qaAC0eucX10bcts63t2gsYLxADoFKQ/edit?usp=sharing
    - Lecture B: https://docs.google.com/presentation/d/1W-l1epRMB3V3jBNW0Oo6Qx5WdQJHiGt4QyzI9WWxsS8/edit?usp=sharing
  - Work with & Discuss with breakout room members!

# Pseudocode Time

- Assume we have set up `struct addrinfo hints` to get both IPv4 and IPv6 addresses
  - Write pseudocode to bind to and listen on the first socket that works

- Pieces you can use:
  - `Error(); // print msg and exit`
  - `retval = getaddrinfo(..., &res);`
  - `freeaddrinfo(res);`
  - `fd = socket(...);`
  - `retval = bind(fd, ...);`
  - `retval = listen(fd, SOMAXCONN);`
  - `close(fd);`

# Example #1

- See server_bind_listen.cc
  - Takes in a port number from the command line
  - Opens a server socket, prints info, then listens for connections for 20 seconds
    - Can connect to it using netcat (`nc`)

# Step 5: Accept a Client Connection

```
int accept(int sockfd, struct sockaddr* addr,
           socklen_t* addrlen);
```

- Returns an active, ready-to-use socket file descriptor connected to a client (or `-1` on error)
  - `sockfd` must have been created, bound, *and* listening
  - Pulls a queued connection or waits for an incoming one
- `addr` and `addrlen` are output parameters
  - `*addrlen` should initially be set to `sizeof(*addr)`, gets overwritten with the size of the client address
  - Address information of client is written into `*addr`
    - Use **`inet_ntop()`** to get the client's printable IP address
    - Use **`getnameinfo()`** to do a *reverse DNS lookup* on the client

# Example #2

- See server_accept_rw_close.cc
  - *Takes in a port number from the command line*
  - *Opens a server socket, prints info, then listens for connections*
    - *Can connect to it using netcat (`nc`)*
  - Accepts connections as they come
  - Echoes any data the client sends to it on `stdout` and also sends it back to the client

# Something to Note

- Our server code is not concurrent
  - Single thread of execution
  - The thread blocks while waiting for the next connection
  - The thread blocks waiting for the next message from the connection

- A crowd of clients is, by nature, concurrent
  - While our server is handling the next client, all other clients are stuck waiting for it ☹

# hw4 demo

- Multithreaded Web Server (333gle)
  - Don't worry – multithreading has mostly been written for you
  - `./http333d <port> <static files> <indices+>`
  - Some security bugs to fix, too

# Extra Exercise #1

- Write a program that:
  - Creates a listening socket that accepts connections from clients
  - Reads a line of text from the client
  - Parses the line of text as a DNS name
  - Does a DNS lookup on the name
  - Writes back to the client the list of IP addresses associated with the DNS name
  - Closes the connection to the client