About how long did Homework 3 take?

A. 0-12 Hours
B. 13-18 Hours
C. 19-24 Hours
D. 25-30 Hours
E. 31+ Hours
F. I didn’t finish / I prefer not to say
Administtrivia

❖ Exercise 15 due Monday

❖ Canvas updated with late days and HW1 + HW2 grades
  ▪ Let Hannah know if you can’t access

❖ HW3:
  ▪ Extra OH tonight! 4-6pm @ 4th floor breakout
  ▪ 1 late day = 8:59pm tonight; 2 late days = 8:59pm on Sunday

❖ HW4 posted and files will be pushed to repos today
  ▪ Due last Thursday of the quarter (12/5)
  ▪ Only 1 late day allowed for HW4 (hard deadline of 12/6)
Lecture Outline

❖ Client-side Networking
  ▪ Step 1: Figure out the IP/Port
    • What is a Network Address?
    • Data structures for address information
    • DNS (Domain Name System): finding IP addresses
  ▪ Step 2: Create a Socket
  ▪ Step 3: Connect the Socket
  ▪ Step 4: `read()` and `write()` Data
  ▪ Step 5: Close the Socket

❖ HW4 demo
Step 2: Creating a Socket

- `int socket(int domain, int type, int protocol);`
  - Creating a socket doesn’t bind it to a local address or port yet
  - Returns file descriptor or `-1` on error

```c
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <iostream>

int main(int argc, char **argv) {
    int socket_fd = socket(AF_INET, SOCK_STREAM, 0);
    if (socket_fd == -1) {
        std::cerr << strerror(errno) << std::endl;
        return EXIT_FAILURE;
    }
    close(socket_fd);
    return EXIT_SUCCESS;
}
```
Lecture Outline

❖ Client-side Networking
  ▪ Step 1: Figure out the IP/Port
    • What is a Network Address?
    • Data structures for address information
    • DNS (Domain Name System): finding IP addresses
  ▪ Step 2: Create a Socket
  ▪ **Step 3: Connect the Socket**
  ▪ Step 4: `read()` and `write()` Data
  ▪ Step 5: Close the Socket

❖ HW4 demo
Step 3: Connect to the Server

❖ The `connect()` system call establishes a connection to a remote host

```c
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

- **sockfd**: Socket file description from Step 2
- **addr** and **addrlen**: Usually from one of the address structures returned by `getaddrinfo` in Step 1 (DNS lookup)
- Returns 0 on success and -1 on error

❖ `connect()` may take some time to return

- It is a **blocking** call by default
- The network stack within the OS will communicate with the remote host to establish a TCP connection to it
  - This involves ~2 **round trips** across the network
Connect Example

❖ See connect.cc

```c
// Get an appropriate sockaddr structure.
struct sockaddr_storage addr;
size_t addrlen;
LookupName(argv[1], port, &addr, &addrlen);

// Create the socket.
int socket_fd = socket(addr.ss_family, SOCK_STREAM, 0);
if (socket_fd == -1) {
    cerr << "socket() failed: " << strerror(errno) << endl;
    return EXIT_FAILURE;
}

// Connect the socket to the remote host.
int res = connect(socket_fd,
        reinterpret_cast<sockaddr*>(addr),
        addrlen);
if (res == -1) {
    cerr << "connect() failed: " << strerror(errno) << endl;
}
```
Lecture Outline

❖ Client-side Networking
  ▪ Step 1: Figure out the IP/Port
    • What is a Network Address?
    • Data structures for address information
    • DNS (Domain Name System): finding IP addresses
  ▪ Step 2: Create a Socket
  ▪ Step 3: Connect the Socket
  ▪ Step 4: read() and write() Data
  ▪ Step 5: Close the Socket

❖ HW4 demo
How do we error check `read()` and `write()`?

A. `ferror()`  
B. Return value less than expected  
C. Return value of 0 or NULL  
D. Return value of -1  
E. I’m not sure...
Step 4: read()

- If there is data that has already been received by the network stack, then read will return immediately with it
  - `read()` might return with less data than you asked for

- If there is no data waiting for you, by default `read()` will `block` until something arrives
  - How might this cause `deadlock`?
  - Can `read()` return 0?
Step 4: `write()`

- `write()` queues your data in a send buffer in the OS and then returns
  - The OS transmits the data over the network in the background
  - When `write()` returns, the receiver probably has not yet received the data!

- If there is no more space left in the send buffer, by default `write()` will `block`
Read/Write Example

❖ See receivesend.cc

```c
while (1) {
    int wres = write(socket_fd, readbuf, res);
    if (wres == 0) {
        cerr << "socket closed prematurely" << endl;
        close(socket_fd);
        return EXIT_FAILURE;
    }
    if (wres == -1) {
        if (errno == EINTR)
            continue;
        cerr << "socket write failure: " << strerror(errno) << endl;
        close(socket_fd);
        return EXIT_FAILURE;
    }
    break;
}
```
Lecture Outline

❖ Client-side Networking
  ▪ Step 1: Figure out the IP/Port
    • What is a Network Address?
    • Data structures for address information
    • DNS (Domain Name System): finding IP addresses
  ▪ Step 2: Create a Socket
  ▪ Step 3: Connect the Socket
  ▪ Step 4: \texttt{read()} and \texttt{write()} Data
  ▪ Step 5: Close the Socket

❖ HW4 demo
Step 5: \texttt{close()}

- \texttt{int close(int fd);}

- Nothing special here – it’s the same function as with file I/O
- Shuts down the socket and frees resources and file descriptors associated with it on both ends of the connection
Lecture Outline

❖ Client-side Networking
  ▪ Roadmap
  ▪ Step 1: Figure out the IP/Port
    • What is a Network Address?
    • Data structures for address information
    • DNS (Domain Name System): finding IP addresses
  ▪ Step 2: Create a Socket
  ▪ Step 3: Connect the Socket
  ▪ Step 4: `read()` and `write()` Data
  ▪ Step 5: Close the Socket

❖ HW4 demo
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:
  ▪ Reads DNS names, one per line, from stdin
  ▪ Translates each name to one or more IP addresses
  ▪ Prints out each IP address to stdout, one per line