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

socket.cc

```
#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;
}
```


Step 3: Connect to the Server

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

Poll Everywhere

pollev.com/cse33320su

- When we call **`write()`**, what data do we need to pass to it when writing over the network?

A. Any data our application needs to send

B. All of the above + TCP info (sequence number, port, …)

C. All of the above + IP info (source & dest IP addresses…)

D. All of the above + Ethernet info (source & dest MAC addresses)

E. We're lost…

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

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