CSE 333 Section 7 - Client-Side Networking

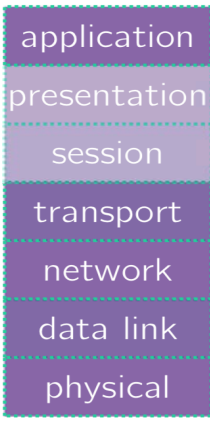
Welcome back to section! We're glad that you're here :)

Computer Networking Review

Exercise 1

a) Match the following protocols to what they are used for. (Bonus: In what *layer* of the networking stack is it found?)

DNS	Reliable transport protocol on top of IP.
IP	Translating between IP addresses and host names.
TCP	Sending websites and data over the Internet.
UDP	Unreliable transport protocol on top of IP.
HTTP	Routing packets across the Internet.



b) Why would you want to use TCP over UDP?

c) Why would you want to use UDP over TCP?

```
struct sockaddr {                // size: really small
  sa_family_t     sa_family;     // Address family (AF_* constants)
  . . .
}
struct sockaddr_in {             // size: small
  sa_family_t     sin_family;    // Address family: AF_INET
  in_port_t       sin_port;      // Port in network byte order
  struct in_addr  sin_addr;      // IPv4 address
  . . .
}
struct sockaddr_in6 {            // size: quite large
  sa_family_t     sin6_family;   // Address family: AF_INET6
  in_port_t       sin6_port;     // Port number
  struct in6_addr sin6_addr;     // IPv6 address
  . . .
}
struct sockaddr_storage {        // size: really large
  sa_family_t     ss_family;     // Address family  (AF_* constants)
  . . .
}
```

Step-by-step Client-Side Networking

Step 1. Figure out what IP address and port to talk to. (`getaddrinfo()`)

```
// returns 0 on success, negative number on failure
int getaddrinfo(const char *hostname,    // hostname to lookup
           const char *servname,         // service name
           const struct addrinfo *hints, // desired output (optional)
           struct addrinfo **res);       // results structure

struct addrinfo {
     int ai_flags;              // additional flags
     int ai_family;             // AF_INET, AF_INET6, AF_UNSPEC
     int ai_socktype;           // SOCK_STREAM, SOCK_DGRAM, 0
     int ai_protocol;           // IPPROTO_TCP, IPPROTO_UDP, 0
     size_t ai_addrlen;         // length of socket addr in bytes
     struct sockaddr* ai_addr;  // pointer to socket addr
     char* ai_canonname;        // canonical name
     struct addrinfo* ai_next;  // can have linked list of records
}
```

Step 2. Create a socket. (`socket()`)

```
// returns file descriptor on success, -1 on failure (errno set)
int socket(int domain,          // AF_INET, AF_INET6, etc.
           int type,            // SOCK_STREAM, SOCK_DGRAM, etc.
           int protocol);       // usually 0
```

Step 3. Connect to the server. (`connect()`)

```
// returns 0 on success, -1 on failure (errno set)
int connect(int sockfd,                  // fd from step 2
            struct sockaddr *serv_addr,  // socket addr from step 1
            socklen_t addrlen);          // size of serv_addr
```

Step 4. Transfer data through the socket. (`read()` and `write()`)

```
// returns amount read, 0 for EOF, -1 on failure (errno set)
ssize_t read(int fd, void *buf, size_t count);

// returns amount written, -1 on failure (errno set)
ssize_t write(int fd, void *buf, size_t count);
```

These are the same POSIX calls used for files, so remember to deal with partial reads/writes!

Step 5. Close the socket when done. (`close()`)

```
// returns 0 for success, -1 on failure (errno set)
int close(int fd);
```

Exercise 2

Fitting the Pieces Together. The following diagram depicts the basic skeleton of a C/C++ program for client-side networking, with arrows representing the flow of data between them. Fill in the names of the functions being called, and the arguments being passed. Then, for each arrow in the diagram, fill in the type and/or data that it represents.