CSE 333 Section 7 - Client-Side Networking

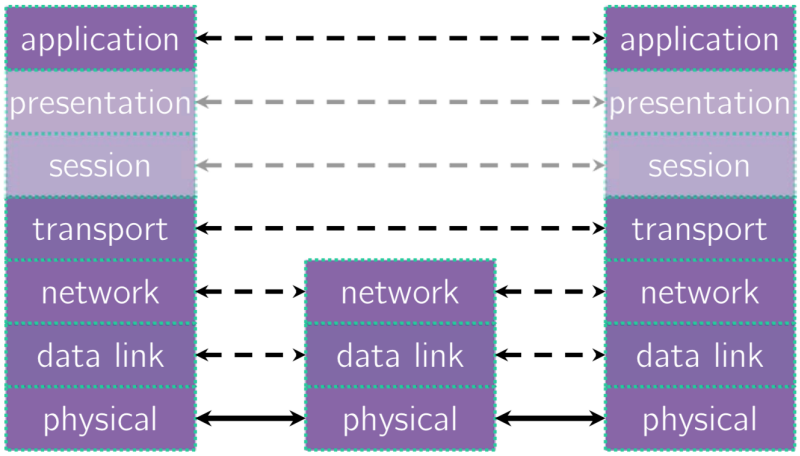
Welcome back to section! We're glad that you're here :)

Networking Quick Review

Exercise 1

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

- DNS
- IP
- TCP
- UDP
- HTTP



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

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

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

When reading from a file you can determine the exact amount of data to read based on the file size. In networking, you do not know how much data you may read across the network. Complete the code below that reads data across the network and prints it to stdout until the connection is closed.

```
int main(int argc, char **argv) {
  int socket_fd;
  char readbuf[512];
  int res;
  ...
  // Assume code to set up the network connection is included

  // Read data from the server until the connection is closed
  while (_____________________________________ != 0) {
    if (res == -1) {
      if(_____________________________________) {
        continue;
      }
      close(socket_fd);
      return EXIT_FAILURE;
    }

    // Write the data we read to stdout
    int pos = 0;
    while (res > 0) {
      int write_res = ____________________________________________
      if (write_res == -1) {
        if (_____________________________________) {
          continue;
        } else {

          ____________________
          return EXIT_FAILURE;
        }
      }
      res -= write_res;
      pos += write_res;
  }

  ____________________
  return EXIT_SUCCESS;
}
```