Lecture 35 (4-22-26)

Networking (System Calls)

- Interview tips:
  - reading 8, reading 9 for interview
  - raaddir, closdir,
  - stat
- On the final:
  - you will need to make a put request

Motivating Questions:

- What is **networking?**
  - What is an **IP address?**
  - What is a **port?**
  - What is a **domian name?**
  - What is a **URL?**
- What is a **protocol?** Examples?

Networking: Sockets/HTTP Client

- HTTP Client:
  - `getaddrinfo`
  - `socket`
  - `connect`
  - `write`
  - `read`
  - `close`

`http_client.c`

```c
/* http_client.c: simple HTTP client */

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>

FILE *socket_dial(const char *host, const char *port) {
    /* Lookup server address information */
    struct addrinfo *results;
    struct addrinfo  hints = {
        .ai_family   = AF_UNSPEC,   /* Return IPv4 and IPv6 choices */
        .ai_socktype = SOCK_STREAM, /* Use TCP */
    };
    int status;
    if ((status = getaddrinfo(host, port, &hints, &results)) != 0) {
         fprintf(stderr, "getaddrinfo failed: %s\n", gai_strerror(status));
     return NULL;
    }

    /* For each server entry, allocate socket and try to connect */
    int client_fd = -1;
    for (struct addrinfo *p = results; p != NULL && client_fd < 0; p = p-
>ai_next) {
     /* Allocate socket */
     if ((client_fd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) <
0) {
         fprintf(stderr, "Unable to make socket: %s\n", strerror(errno));
         continue;
     }

     /* Connect to host */
     if (connect(client_fd, p->ai_addr, p->ai_addrlen) < 0) {
         close(client_fd);
         client_fd = -1;
         continue;
     }
    }

    /* Release allocate address information */
    freeaddrinfo(results);
```

```c
    if (client_fd < 0) {
     fprintf(stderr, "Unable to connect to %s:%s: %s\n", host, port,
strerror(errno));
          return NULL;
    }

    /* Open file stream from socket file descriptor */
    FILE *client_file = fdopen(client_fd, "w+");
    if (!client_file) {
        fprintf(stderr, "Unable to fdopen: %s\n", strerror(errno));
        close(client_fd);
        return NULL;
    }

    return client_file;
}

int main(int argc, char *argv[]) {
    char *host = "example.com";
    char *port = "80";

    /* Parse command line arguments */
    if (argc > 1) host = argv[1];
    if (argc > 2) port = argv[2];

    /* Connect to remote machine */
    FILE *client_file = socket_dial(host, port);
    if (!client_file) {
        return EXIT_FAILURE;
    }

    /* Send HTTP Request */
    fprintf(client_file, "GET / HTTP/1.0\r\n");
    fprintf(client_file, "Host: %s\r\n", host);
    fprintf(client_file, "\r\n");                    // Must end Request with
empty line

    /* Read HTTP Response */
    char buffer[BUFSIZ];
    while (fgets(buffer, BUFSIZ, client_file)) {
        fputs(buffer, stdout);
    }

    /* Close socket */
    fclose(client_file);
```

```
    return EXIT_SUCCESS;
}

/* vim: set expandtab sts=4 sw=4 ts=8 ft=c: */
```

HTTP request example

```
### Client to server ###
### The following is the `write` ###

GET /PATH HTTP/1.0\r\n
Host: google.com\r\n
User-Agent: curlit/0.1alpha\r\n
\r\n

### Server to Client ###
### the following is the `read` ###
## status code is returned (look for 200 OK) ##

HTTP/1.o 200 OK\r\n
Content-Length: 13\r\n
\r\n
Hello, World! #these are the contents
```