Lecture 34 (4-20-26)

Networking (System Calls)

Notes for final:

- review `findit` homework
- watch start of recoding of this lecture to hear all the hints

---

Motivating Questions

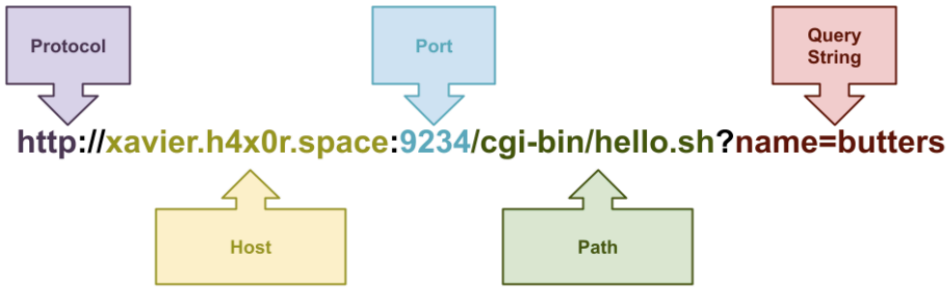
- What is **networking?**
  - what is an **IP address**
  - what is a **port**
  - what is a **domain name**
  - what is a **URL**
- What is a **protocol?** Examples?

- In every connection you have some sort of **client** and some sort of **server**
- The server has **IP addresses:**
  - local host
  - public ip address
- DNS translates domain name to IP address
- client also needs to know a port as well
- In our class **server** means an individual application/process
- so client needs to know the server's IP address/domain name and port
- You can run any service on any port
- the client also has an:
  - Ip address
  - port
- so that when the server sends stuff back it goes to that ip address / port
- The way messages go back and forth:
  - **TCP (transmission control protocol)** (there are others like UDP, but we will use TCP)

        - enables us to use `fgets()`, etc.
    - the client sends **packets** to the server
        - these packets can be dropped
- The web browser is a HTTP client

URL

> A **URL** is a **uniform resource locator** that corresponds to a resource on the Internet:



- Host can also be called domain
- Path can also be called resource
- Query string can also be called parameters

Networking : Sockets / Echo Client

now these system calls and the order

- In typical chronological order:
    - In C the system call we want to make for DNS is `getaddrinf0()`
        - translates domain name into ip address
        - need to use in a loop because you will get multiple ip addresses
    - `socket`: returns file descriptor
    - `connect`: to the other end
        - once connected you usually do one of the following (might happen in a loop):
            - `write`
            - `read`
    - `close`

---

echo_client.c

```c
/* echo_client_refactored.c: simple TCP echo client (refactored) */

#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>

const char *HOST = "localhost";
const char *PORT = "9416";

FILE *socket_dial(const char *host, const char *port) {
    /* Lookup server address information */
    struct addrinfo *results
    struct addrinfo *hints = {
         .ai_family = AF_UNSPEC, // looks for ipv4 and ipv6
         .ai_socketype = SOCK_STREAM,  // use TCP as underlying socket type
    }

    int status;
    if ((status = getaddrinfo(host, port, &results)) != 0){
         fprintf(stderr, "getaddrinfo: %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 && client_fd < 0; p=p->ai_next){
         /* Allocate socket */
         client_fd = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
         if (client_fd < 0){
              perror("socket");
              continue;
         }
         /* Connect to host */
         if (connect(client_fd, p->ai_addr, p->ai_addrlen) < 0){
              close(client_fd);
              client_fd = -1;
              continue;
         }
```

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

    if (client_fd < 0){ // couldn't connect
        return NULL
    }

    /* Open file stream from socket file descriptor */

    return fdopen(cleint_fd, "r+");
}

int main(int argc, char *argv[]) {
    /* Connect to remote machine */
    FILE *client_file = socket_dial(HOST, PORT);
    if (!client_filed){
        return EXIT_FAILURE;
    }
    /* Read from stdin and send to server */
    char buffer[BUFSIZ];
    while (fgets(buffer, BUFSIZ, stdin)){
        fputs(buffer, client_file);
        fgets(buffer, BUFSIZ, client_file);
        fputs(buffer, stdout)
    }

    fclose(client_file);
    return EXIT_SUCCESS;
}

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