CSE 29
Lecture 18 Summary

March 5, 2026

📣 Logistical Things

- Nothing! Check Piazza for new announcements!

🧠 Review Questions

Answers in the next slide!

Q1: Given the following request string:

```
char request[] = "GET /add?s=goodbye HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts "`goodbye`" into a variable called `value`. Use any functions you want, but also refer to `sscanf` and `strstr` from the handout if needed!

Q2: Given the following request string

```
char request[] = "GET /add-number?n=367 HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts the number `367` into an `int` variable called `n`

Q3: Our HTTP server gives an `int client` value to the handler. What is that value used for?

🧠 Review Question Answers

Q1:

- `char value[256];`
- `value = strstr(request, “s=”); strtok(.....)`

or

- `char value[256];`
- `sscanf(request, “GET /add?s=%s”, value);`

Q2:

- `int n;`
- `sscanf(request, “GET /add-number?=%d”, &n);`

🧠 Review Question Answers

Q3: It is a file descriptor for a socket connected back to client.

We use it with write. `write(client, data, len)`

🧠 RQ Follow-up: What is a socket?

A socket is an abstraction for a connection to another computer or process. It's like a process ID for the network

- int client is like a file descriptor for a socket
  - In the OS, a file descriptor is a thing you can send and receive bytes to/from

Side Note: The write function

write(client, buffer, len)

- If you put
  - client = 1, you can send bytes to stdout (print out to your terminal)
  - client = 2, you can send bytes to stderr

Questions

- Are ports places you can put sockets?
  - Ports are more public than sockets
  - For example, there is one port for YouTube (port 443) and we all connect to that one port
  - When there's a new connection to the port, a new socket is created for that computer
- What is the difference between a port and a socket?
  - A port is something that is externally accessible, the socket is the specific connection between you and the server.
  - ie when we all did the chat server in lab together, we (the clients) all used port 8250 (or otherwise all the same) whereas each client had its own socket.

start_server

The code in `start_server`

Inside start_server

```
// http-server.c
typedef void (*RequestHandler)(char *, int);
void start_server(RequestHandler handler, int port) {

  char buffer[2048];

  ... set up on the given port ...

  while(1) {
      // not the real system call names
      int client = accept_connection_from_internet();
      read_from_internet(buffer, client);
      buffer[bytes] = '\0';
      (*handler)(buffer, client);
      close(client);
  }
}
```

```
// string-server.c
void handle(char *request, int client) { /* all of PA5 */  }

int main() { start_server(&handle, 2900); }
```

Making the type `(char*,int)` into one thing called `*RequestHandler`

An equivalent function header without the typedef would be
void start_server(void(*handler)(char*, int), int port){

Fake pseudocode functions

Let's try writing this in Java

typedef RequestHandler = void (*)(char*, int)

Inside start_server

```
// http-server.c
typedef void (*RequestHandler)(char *, int);
void start_server(RequestHandler handler, int port) {

  char buffer[2048];

  ... set up on the given port ...

  while(1) {
      // not the real system call names
      int client = accept_connection_from_internet();
int bytes = read_from_internet(buffer, client);
      buffer[bytes] = '\0';
      (*handler)(buffer, client);   <- this calls "our" code
      close(client);                   your PA5 code
  }
}
```

```
// string server.c
void handle(char *request, int client) { /* all of PA5 */ }

int main() { start_server(&handle, 2900); }
```

```
interface RequestHandler {
    void handle(char[] req, int client);
}
class Server {
    static void start_server(RequestHandler handler,
                             int port) {
        while(1) {...
            handler.handle(buffer, client);
        }
    }
}
class PA5Handler implements RequestHandler {
    void handle(char[] req, int client) {...}
}
```

Why not use Java?

While we could write this part of our program in C, we want the functionality of http-server.c to be in a low level language like C for efficiency/speed.

Java is nicer to use was the consensus reached in lecture

Continuing Building our Server

Here is our goal

Everytime the URL is accessed (with the proper formatting of `/add?s=`) we want to append that string onto a list of strings included before.

Program 4: Keeping state across requests

Goal is to have this kind of behavior:

```
$ curl "localhost:2900/add?s=hello"
hello
$ curl "localhost:2900/add?s=world"
hello
world
$ curl "localhost:2900/add?s=goodbye"
hello
world
goodbye
```

The Issue to Address

- We expect web servers to run forever
- Data that persists between requests can’t live on the stack, it needs to be in global variables or on the heap
- We need to be able to use a little bit of stack space and also store some data long term
- How can we do that?

Let's try implementing `add_string`

The goal of the function is to populate the `char*` array: `strings`. Think about how we might complete this method.

```
12 #define MAX_STRINGS 100
13
14 char *strings[MAX_STRINGS];
15 int num_strings = 0;
16
17 void add_string(char *s) {
18     if (num_strings >= MAX_STRINGS) { return; }
21     num_strings++;
22 }
23
```

```
32 void handle(char *request, int client) {
33     char path[256];
34     sscanf(request, "GET %s", path);
35
36     if (strncmp(path, "/add", 4) == 0) {
37         char *query_start = strstr(path, "?s=");
38         if (query_start) {
39             char *string_start = query_start + 3;
40             add_string(string_start);
41             respond_with_list(client);
42         } else {
43             send_404(client, "Missing ?s= parameter");
44         }
45     } else {
46         send_404(client, "Unknown path");
47     }
48 }
49
50 int main() { start_server(&handle, 2900); }
```

Would this `add_string` implementation work?

What if `add_string` just stores the pointer?

```
void add_string(char *s) {
    strings[num_strings] = s;
    num_strings++;
}
```

Hint on next slide!

Would this `add_string` implementation work?

What if `add_string` just stores the pointer?

```
void add_string(char *s) {
    strings[num_strings] = s;
    num_strings++;
}
```

What local variable does `s` in `add_string` point to in the code to the right?

```
32 void handle(char *request, int client) {
33     char path[256];
34     sscanf(request, "GET %s", path);
35
36     if (strncmp(path, "/add", 4) == 0) {
37         char *query_start = strstr(path, "?s=");
38         if (query_start) {
39             char *string_start = query_start + 3;
40             add_string(string_start);
41             respond_with_list(client);
42         } else {
43             send_404(client, "Missing ?s= parameter");
44         }
45     } else {
46         send_404(client, "Unknown path");
47     }
48 }
49
50 int main() { start_server(&handle, 2900); }
```

Would this `add_string` implementation work?

What if `add_string` just stores the pointer?

```
void add_string(char *s) {
    strings[num_strings] = s;
    num_strings++;
}
```

- This would **NOT** work.
- The pointer of the variable `s` is pointing to an address inside of the `path` variable
- `handle` likely overwrites `path` on each call, which would change what `s` is

Let's try implementing `add_string`

With that in mind, how would you fill in `add_string` so as to avoid the problem of `s` being a pointer to a variable on the stack?

```
12 #define MAX_STRINGS 100
13
14 char *strings[MAX_STRINGS];
15 int num_strings = 0;
16
17 void add_string(char *s) {
18     if (num_strings >= MAX_STRINGS) { return; }
21     num_strings++;
22 }
23
```

```
32 void handle(char *request, int client) {
33     char path[256];
34     sscanf(request, "GET %s", path);
35
36     if (strncmp(path, "/add", 4) == 0) {
37         char *query_start = strstr(path, "?s=");
38         if (query_start) {
39             char *string_start = query_start + 3;
40             add_string(string_start);
41             respond_with_list(client);
42         } else {
43             send_404(client, "Missing ?s= parameter");
44         }
45     } else {
46         send_404(client, "Unknown path");
47     }
48 }
49
50 int main() { start_server(&handle, 2900); }
```

A correct implementation of `add_string`

```
17 void add_string(char *s) {
18     if (num_strings >= MAX_STRINGS) { return; }
       char* str = malloc(strlen(s)+1);
       strcpy(str, s);
       strings[num_strings] = str;
21     num_strings++;
22 }
```

We are storing malloc'd data in a global variable, which is kind of risky. We would have to worry about use-after-free issues

Let's write `repsond_with_list`

We are sending back a newline-separated list of strings to the client.

Fill in the function. (solution on next slide)

```
24 void respond_with_list(int client) {
25     write(client, HTTP_200, strlen(HTTP_200));
26     for (int i = 0; i < num_strings; i++) {



28         write(client, "\n", 1);
29     }
30 }
```

Let's write `repsond_with_list`

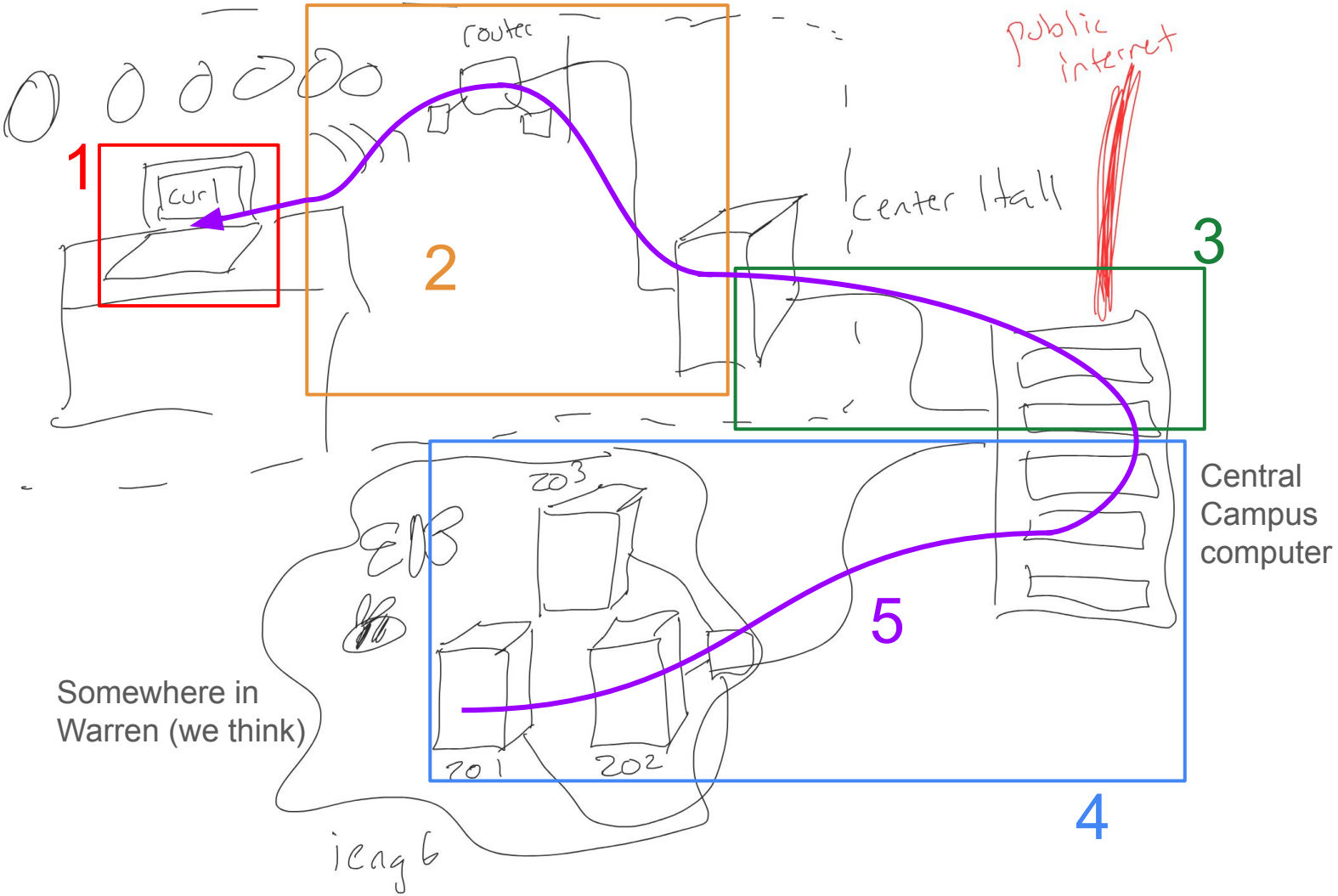
We are sending back a newline-separated list of strings to the client

```
24 void respond_with_list(int client) {
25     write(client, HTTP_200, strlen(HTTP_200));
26     for (int i = 0; i < num_strings; i++) {
           write(client, strings[i], strlen(strings[i]));
28         write(client, "\n", 1);
29     }
30 }
```

The Internet

How does the Internet work? (Picture on next slide)

- 1 We curl from our computer, try to connect to ieng6-201
- 2 A signal is sent over to the WiFi receivers, which has a wire connecting to a router in the building (a router is a fancy server)
- 3 The router has a cable going to a big rack of computers in a central campus server
- 4 The big rack of computer has a cable going to Warren College (School of Engineering) where the ieng6-201, ieng6-202, ieng6-203 servers exist
- 5 Ieng6 sends a response back through the same route it took


1
curl
router
2
public internet
center 1tall
3
Central
Campus
computer
EIB
203
201
202
5
Somewhere in
Warren (we think)
4
ieng6

Questions

- What about sshing off-campus
  - Some of the wires in the central campus goes to the public internet
  - The wire goes to the correct server
  - These are usually fiber optic cables
  - Goes across the country, underwater, etc.
- Does this have to be UCSD-PROTECTED or UCSD-GUEST
  - The WiFi receivers accept both protected and guest. That info is sent over
- Why are there 6 wifi receivers in one room
  - Wifi signals are not targeted and there's a lot of signals going on in a room
  - It is addressing the density of the devices in a room to sort out the signals
- Is it similar to how cellular data works
  - Yes, the 5G radio signal in your phone is stronger than your computer sending a signal to the wifi and it sent over to a cell tower
  - That’s also why your cell provider knows where you are at all times

Questions

- How does a VPN work?
  - VPN - Virtual Private Network
  - Instead of your computer makes a connection to where you want to, your computer makes a connections to a VPN
  - The response body to the VPN asks it to make a connection
  - The requests are to the servers in different places and then the stuff goes back
- How do websites detect you are using a VPN
  - There is a timing difference between a London computer connecting to a server in London and a California computer connecting to a server in London

Joe's Notes (11am)

Review Questions

1. Given the following request string:

```
char request[] = "GET /add?s=goodbye HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts **"goodbye"** into a variable called **value**. Use any functions you want, but also refer to **sscanf** and **strstr** from the handout if needed!

```
char str[256];
sscanf(request, "GET /add?s=%s", str);
```

2. Given the following request string:

```
char request[] = "GET /add-number?n=367 HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts the number 367 into an **int** variable called **n**.

```
int n;
sscanf(request, "GET /add-number?n=%d", &n);
```

3. Our http server gives an **int client** value to the handler. What is that value used for?

It is a file descriptor for a socket connected back to client

We use it with write write(client, data, len)

typedef RequestHandler = void (*)(char*, int)

Inside start_server

```
// http-server.c
typedef void (*RequestHandler)(char *, int);
void start_server(RequestHandler handler, int port) {

  char buffer[2048];

  ... set up on the given port ...

  while(1) {
      // not the real system call names
      int client = accept_connection_from_internet();
int bytes = read_from_internet(buffer, client);
      buffer[bytes] = '\0';
      (*handler)(buffer, client);   <- this calls "our" code
      close(client);                   your PA5 code
  }
}
```

```
// string-server.c
void handle(char *request, int client) { /* all of PA5 */ }

int main() { start_server(&handle, 2900); }
```

```
interface RequestHandler {
    void handle(char[] req, int client);
}
class Server {
    static void start_server(RequestHandler handler,
                             int port) {
        while(1) {...
            handler.handle(buffer, client);
        }
    }
}
class PA5Handler implements RequestHandler {
    void handle(char[] req, int client) {...}
}
```

Joe's Notes (11am)

Program 4: Keeping state across requests

Goal is to have this kind of behavior:

```
$ curl "localhost:2900/add?s=hello"
hello
$ curl "localhost:2900/add?s=world"
hello
world
$ curl "localhost:2900/add?s=goodbye"
hello
world
goodbye
```

A server runs forever. Data that persists between requests can't live on the stack — it needs to be in global variables or on the heap.

What if **add_string** just stores the pointer?

```
void add_string(char *s) {
    strings[num_strings] = s;
    num_strings++;
}
```

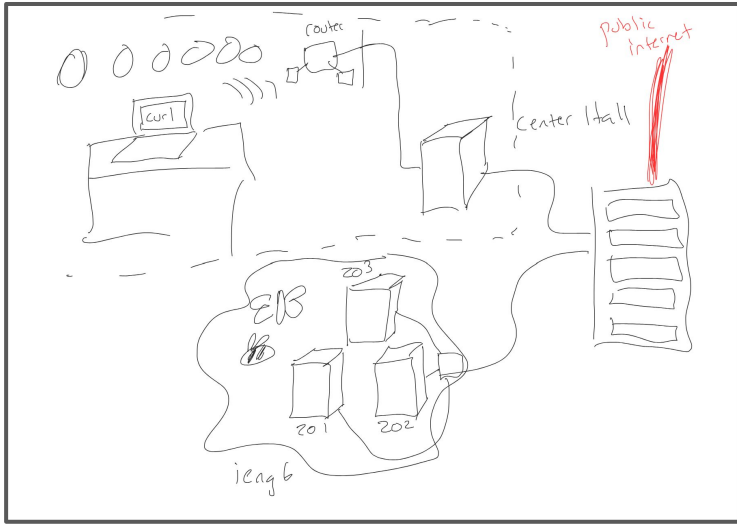
Where does s "point to"?

What address?

some small offset into path[256]

likely overwritten on each call to handle (pointer into stack)

```
4  char *HTTP_200 = "HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\n";
5  char *HTTP_404 = "HTTP/1.1 404 Not Found\r\nContent-Type: text/plain\r\n\r\n";
6
7  void send_404(int client, char *message) {
8      write(client, HTTP_404, strlen(HTTP_404));
9      write(client, message, strlen(message));
10 }
11
12 #define MAX_STRINGS 100
13
14 char *strings[MAX_STRINGS];
15 int num_strings = 0;
16
17 void add_string(char *s) {
18     if (num_strings >= MAX_STRINGS) { return; }
           char* str = malloc(strlen(s)+1);
           strcpy(str, s);
           strings[num_strings] = str;
21     num_strings++;
22 }
23
24 void respond_with_list(int client) {
25     write(client, HTTP_200, strlen(HTTP_200));
26     for (int i = 0; i < num_strings; i++) {
           write(client, strings[i], strlen(strings[i]));
28         write(client, "\n", 1);
29     }
30 }
31
32 void handle(char *request, int client) {
33     char path[256];
34     sscanf(request, "GET %s", path);
                          copies into path
35
36     if (strncmp(path, "/add", 4) == 0) {
37         char *query_start = strstr(path, "?s=");
38         if (query_start) {
39             char *string_start = query_start + 3;
40             add_string(string_start);
41             respond_with_list(client);
42         } else {
43             send_404(client, "Missing ?s= parameter");
44         }
45     } else {
46         send_404(client, "Unknown path");
47     }
48 }
49
50 int main() { start_server(&handle, 2900); }
```



Joe's Notes (12:30pm)

Review Questions

1. Given the following request string:

```
char request[] = "GET /add?s=goodbye HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts "goodbye" into a variable called **value**. Use any functions you want, but also refer to **sscanf** and **strstr** from the handout if needed!

strstr(request, "s=")
... strtok ...

```
char str[256];
sscanf(request, "GET /add?s=%s", str);
```

2. Given the following request string:

```
char request[] = "GET /add-number?n=367 HTTP/1.1\r\nHost: localhost:2900\r\n";
```

Write code that extracts the number 367 into an **int** variable called **n**.

```
int n;
sscanf(request, "GET /add-number?n=%d", &n);
```

3. Our http server gives an **int client** value to the handler. What is that value used for?

"socket" abstraction for a connection to another computer or process

int client = file descriptor for a socket

a thing you can send + receive bytes to/from

write(client, buffer, len)

1 = stdout

Inside start_server

```
// http-server.c   "a pointer to a function w/2 args returning void..."
typedef void (*RequestHandler)(char *, int);
void start_server(RequestHandler handler, int port) {

  char buffer[2048];

  ... set up on the given port ...

  while(1) {
      // not the real system call names
      int client = accept_connection_from_internet();
int bytes = read_from_internet(buffer, client);
      buffer[bytes] = '\0';
      (*handler)(buffer, client);      Calls "our" code
      close(client);
  }
}
```

```
// string-server.c
void handle(char *request, int client) { /* all of PA5 */ }

int main() { start_server(&handle, 2900); }
```

```
interface RequestHandler {
    void handle(char[] req, int client);
}
class HTTPServer {
    static void startServer(RequestHandler h, int port) {
        ... port setup...
        while(true) {
            ...
            h.handle(buffer, client);
            ...
        }
    }
}
- - - - - - - - - - - - - - - - - - - - - -
class StringServer implements RequestHandler {
    public void handle(char[] req, int client) { /* our code */ }
}
class Main {
    HTTPServer.startServer(new StringServer(), 2900);
}
```

Reference: sscanf

HTTP Request and Response

Joe's Notes (12:30pm)