

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); }
```

Reference: sscanf

Reads formatted data from a string — the inverse of `sprintf`.

```
char name[50]; int age;
sscanf("Alice 30", "%s %d", name, &age);
// name is "Alice", age is 30
// %s stops at whitespace
```

Reference: strstr

`strstr(haystack, needle)` returns a pointer to the first occurrence of `needle` in `haystack`, or `NULL`.

```
char *p = strstr("key=value", "=");
// p points to "=value"
// p + 1 points to "value"
```

HTTP Request and Response

A request from `curl "localhost:2900/add?s=hello"`:

```
GET /add?s=hello HTTP/1.1\r\n
Host: localhost:2900\r\n
User-Agent: curl/8.7.1\r\n
```

A response sent back with `write(client, ...)`:

```
HTTP/1.1 200 OK\r\n
Content-Type: text/plain\r\n
\r\n
...response body...
```

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++;
}
```

```
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; }
19
20     num_strings++;
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++) {
27
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);
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); }
```

Review Questions

1. Given the following request string:

```
char request[] =  
    "GET /add?s=goodbye HTTP/1.1\r\n"  
    "Host: 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!

2. Given the following request string:

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

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

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

Function Pointers: Java vs C

In Java, you pass behavior using an *interface*:

```
// http-server.java
interface RequestHandler {
    void handle(String request, int client);
}

class Server {
    static void startServer(RequestHandler handler, int port) {
        // server implementation
        ... handler.handle(requestFromClient, clientFileDescriptor);
        ...
    }
}
```

```
// string-server.java
class StringHandler implements RequestHandler {
    public void handle(String request, int client) { /* ... */ }
}

class Main {
    public static void main(String[] args) {
        Server.startServer(new StringHandler(), 2900);
    }
}
```

In C, there are no interfaces. We pass a *function pointer* — the address of a function. We can use a *typedef* to name the type to make it look more like Java:

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

void start_server(RequestHandler handler, int port) {
    // server implementation
    ... (*handler)(request_from_client, client_file_descriptor); ...
}
```

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

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

In Python, functions can be passed directly, which is more like C:

```
# http-server.py
def start_server(handler, port):
    ... handler(request_from_client, client_file_descriptor) ...

# string-server.py
def handle(request, client): ...

start_server(handle, 2900)
```