

abr 01, 19 17:07

## server\_template.h

Page 1/1

```

1  #ifndef SERVER_TEMPLATE_H
2  #define SERVER_TEMPLATE_H
3
4  struct server_template {
5      char* text;
6      char* to_replace;
7  };
8
9
10 bool server_template_create(struct server_template *self, char* filename);
11 char* server_template_cat(struct server_template *self, char* replacement);
12 void server_template_destoy(struct server_template *self);
13
14 #endif

```

abr 01, 19 17:07

## server\_template.c

Page 1/1

```

1
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <errno.h>
6  #include <stdbool.h>
7  #include "server_template.h"
8
9  #define TO_REPLACE "{{datos}}"
10 #define SIZE_TO_REPLACE 9
11 #define MAX_LEN_REPLY 2000
12
13 bool server_template_create(struct server_template *self, char* filename) {
14     self->text = malloc(MAX_LEN_REPLY);
15     if ( !self->text ) return false;
16     FILE* file = fopen(filename, "r");
17     if ( !file ) {
18         free(self->text);
19         return false;
20     }
21
22     size_t i = 0;
23     while ( !feof(file) ^ i < MAX_LEN_REPLY ) {
24         self->text[i] = (char) fgetc(file);
25         i++;
26     }
27     self->text[i-1] = '\0';
28     self->to_replace = strstr(self->text, TO_REPLACE);
29     fclose(file);
30     return true;
31 }
32
33 char* server_template_cat(struct server_template *self, char* replacement) {
34     char* reply = malloc(MAX_LEN_REPLY);
35     if ( !reply ) return NULL;
36     char aux[MAX_LEN_REPLY];
37     snprintf(aux, MAX_LEN_REPLY, "%s", self->text);
38
39     char* to_replace = strstr(aux, TO_REPLACE);
40
41     snprintf(to_replace, strlen(to_replace) - 1, "%s", replacement);
42     snprintf(reply, MAX_LEN_REPLY - 1, "%s", aux);
43
44     int len = strlen(&to_replace[SIZE_TO_REPLACE]);
45     snprintf(&reply[strlen(reply)], len, "%s", &to_replace[SIZE_TO_REPLACE]);
46     return reply;
47 }
48
49 void server_template_destoy(struct server_template *self) {
50     free(self->text);
51 }

```

abr 01, 19 17:07

## server\_socket.h

Page 1/1

```

1  #ifndef SERVER_SOCKET_H
2  #define SERVER_SOCKET_H
3  #include <stdlib.h>
4  #include <stdbool.h>
5
6  struct server_socket {
7      char* port;
8      int skt;
9      int current_peerskt;
10 };
11
12 /*
13 Crea e inicializa el socket definiendo la familia, el tipo de socket y el
14 protocolo para poder conectarse al cliente por medio del port y host indicados
15 */
16 bool server_socket_create(struct server_socket *self, char* _port);
17
18 /*
19 Almacena los parametros necesarios para la inicializaci3n del socket.
20 */
21 bool server_socket_start(struct server_socket *self);
22
23 void server_socket_destroy(struct server_socket *self);
24 int server_socket_accept_client(struct server_socket *self);
25
26 char* server_socket_receive_message(struct server_socket *self);
27 int server_socket_send_message(struct server_socket *self, char* buf, int size);
28
29 /*
30 Desactiva las operaciones de env3o y recepci3n para el cliente y para si mismo
31 */
32 void server_socket_disable_client(struct server_socket *self);
33
34 #endif

```

abr 01, 19 17:07

## server\_socket.c

Page 1/3

```

1  #define _POSIX_C_SOURCE 200112L
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #include <errno.h>
7  #include <stdbool.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <netdb.h>
12 #include <unistd.h>
13 #include "server_socket.h"
14
15 #define LEN_PORT 6
16 #define MAX_WAITING_CLIENTS 20
17 #define MAX_LEN_BUF 2000
18
19 bool server_socket_create(struct server_socket *self, char* _port) {
20     self->port = malloc(LEN_PORT);
21     if ( !self->port ) return false;
22     snprintf(self->port, LEN_PORT, "%s", _port);
23     self->current_peerskt = 0;
24     return true;
25 }
26
27 bool server_socket_start(struct server_socket *self) {
28     int s = 0;
29
30     struct addrinfo hints;
31     struct addrinfo *ptr;
32
33     int skt = 0;
34
35     memset(&hints, 0, sizeof(struct addrinfo));
36     hints.ai_family = AF_INET;
37     hints.ai_socktype = SOCK_STREAM;
38     hints.ai_flags = AI_PASSIVE;
39
40     s = getaddrinfo(NULL, self->port, &hints, &ptr);
41
42     if (s != 0) {
43         return false;
44     }
45
46     skt = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
47
48     if (skt == -1) {
49         freeaddrinfo(ptr);
50         return false;
51     }
52
53     int val = 1;
54     s = setsockopt(skt, SOL_SOCKET, SO_REUSEADDR, &val, sizeof(val));
55     if (s == -1) {
56         close(skt);
57         freeaddrinfo(ptr);
58         return false;
59     }
60
61     s = bind(skt, ptr->ai_addr, ptr->ai_addrlen);
62     if (s == -1) {
63         close(skt);
64         freeaddrinfo(ptr);
65         return false;
66     }

```

abr 01, 19 17:07

server\_socket.c

Page 2/3

```

67     }
68
69     freeaddrinfo(ptr);
70     s = listen(skt, MAX_WAITING_CLIENTS);
71     if (s == -1) {
72         close(skt);
73         return false;
74     }
75     self->skt = skt;
76     return true;
77 }
78
79 // -1 si falla
80 int server_socket_accept_client(struct server_socket *self){
81     int peerskt = accept(self->skt, NULL, NULL);
82     self->current_peerskt = peerskt;
83     return peerskt;
84 }
85
86 void server_socket_destroy(struct server_socket *self) {
87     free(self->port);
88     shutdown(self->skt, SHUT_RDWR);
89     close(self->skt);
90 }
91
92 char* server_socket_receive_message(struct server_socket *self) {
93     char* buf = malloc(MAX_LEN_BUF);
94     if ( !buf ) return NULL;
95     memset(buf, 0, MAX_LEN_BUF);
96     int received = 0;
97     int s = 0;
98     bool is_the_socket_valid = true;
99     while (received < MAX_LEN_BUF ^ is_the_socket_valid) {
100         s = recv(self->current_peerskt, buf + received, \
101                 MAX_LEN_BUF - received, MSG_NOSIGNAL);
102         if (s == 0) {
103             is_the_socket_valid = false;
104         } else if (s < 0) {
105             is_the_socket_valid = false;
106         } else {
107             received += s;
108         }
109     }
110     return buf;
111 }
112
113 int server_socket_send_message(struct server_socket *self, char* buf, \
114                               int size) {
115     int sent = 0;
116     int s = 0;
117     bool is_the_socket_valid = true;
118
119     while (sent < size ^ is_the_socket_valid) {
120         s = send(self->current_peerskt, &buf[sent], size-sent, MSG_NOSIGNAL);
121         if (s <= 0) {
122             return -1;
123         } else {
124             sent += s;
125         }
126     }
127     return sent;
128 }
129
130 void server_socket_disable_client(struct server_socket *self) {
131     shutdown(self->current_peerskt, SHUT_RDWR);
132     close(self->current_peerskt);

```

abr 01, 19 17:07

server\_socket.c

Page 3/3

133 }

abr 01, 19 17:07

server\_sensor.h

Page 1/1

```

1  #ifndef SERVER_SENSOR_H
2  #define SERVER_SENSOR_H
3  #include <stdbool.h>
4
5  struct server_sensor {
6      FILE* file;
7  };
8
9  bool server_sensor_create(struct server_sensor* self, char* filename);
10 /*
11 Se ocupa de leer de del archivo binario file una temperatura
12 almacenada en 16 bits y formato big-endian.
13 La misma la interpreta de la siguiente forma: Temperatura = (datos - 2000)/100
14 */
15 char* server_sensor_read(struct server_sensor* self);
16
17 //comunica si quedan o no temperaturas por leer.
18 bool does_the_sensor_still_have_temperatures(struct server_sensor* self);
19
20 void server_sensor_destroy(struct server_sensor* self);
21
22 #endif
23

```

abr 01, 19 17:07

server\_sensor.c

Page 1/1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <errno.h>
5  #include <stdbool.h>
6  #include <arpa/inet.h>
7
8  #include "server_sensor.h"
9
10 #define SIZE_OF_TEMPERATURE 2
11 #define MAX_LEN_TEMPERATURE_MESSAGE 200
12 #define SIZE_TO_REPLACE 9
13
14
15 bool server_sensor_create(struct server_sensor* self, char* filename) {
16     FILE* file = fopen(filename, "r+b");
17     if (!file) {
18         return false;
19     }
20     self->file = file;
21     return true;
22 }
23
24
25 char* server_sensor_read(struct server_sensor* self) {
26     unsigned short int read;
27     size_t len = fread((void*)&read, SIZE_OF_TEMPERATURE, 1, self->file);
28     if (!len) return NULL;
29     float temperature = (float) ntohs(read);
30     temperature = (temperature - 2000) / 100;
31     char* message = malloc(MAX_LEN_TEMPERATURE_MESSAGE);
32     if (!message) return NULL;
33     snprintf(message, SIZE_TO_REPLACE, "%.2f", temperature);
34     return message;
35 }
36
37 bool does_the_sensor_still_have_temperatures(struct server_sensor* self) {
38     return !feof(self->file);
39 }
40
41
42 void server_sensor_destroy(struct server_sensor* self) {
43     fclose(self->file);
44 }

```

abr 01, 19 17:07

server\_request\_processor.h

Page 1/1

```

1  #ifndef SERVER_REQUEST_PROCESSOR_H
2  #define SERVER_REQUEST_PROCESSOR_H
3  #include <stdbool.h>
4
5
6  struct server_req_proc {
7      char* request;
8      bool is_method_resource_valid;
9  };
10
11
12  bool req_proc_create(struct server_req_proc* self, char* request);
13  void req_proc_destroy(struct server_req_proc* self);
14
15  /*
16  Verifica que el método utilizado sea del tipo "GET"
17  y el recurso sea "/sensor".
18
19  Si el método no es "GET", la respuesta será; un
20  error de tipo "400 Bad request"
21  Si el recurso no es "/sensor", la respuesta será; un error
22  de tipo "404 Not found".
23  Si el método y recurso son válidos, la respuesta es
24  de tipo "200 OK".
25  */
26  char* req_porc_method_resource(struct server_req_proc* self);
27  bool req_porc_is_method_resource_valid(struct server_req_proc* self);
28
29  /*
30  Busca y devuelve el valor del user-agent en el
31  request con formato clave:valor.
32  */
33  char* req_porc_user_agent(struct server_req_proc* self);
34
35  #endif

```

abr 01, 19 17:07

server\_request\_processor.c

Page 1/2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include <errno.h>
6  #include <stdbool.h>
7  #include "server_request_processor.h"
8
9  #define METHOD_OFFSET 0
10 #define METHOD_ERROR_MESSAGE "HTTP/1.1 400 Bad request\n"
11 #define LEN_METHOD 3
12 #define CORRECT_METHOD "GET"
13
14 #define RESOURCE_OFFSET 4
15 #define RESOURCE_ERROR_MESSAGE "HTTP/1.1 404 Not found\n"
16
17 #define LEN_RESOURCE 7
18 #define CORRECT_RESOURCE "/sensor"
19
20 #define MAX_LEN_REQUEST 2000
21
22 #define METHOD_ERROR_MESSAGE "HTTP/1.1 400 Bad request\n"
23 #define RESOURCE_ERROR_MESSAGE "HTTP/1.1 404 Not found\n"
24 #define METHRES_SUCCESS_MESSAGE "HTTP/1.1 200 OK\n\n"
25 #define MAX_LEN_MESSAGE 26
26
27 #define USER_AGENT_KEY "User-Agent:"
28 #define USER_AGENT_VAL_OFFSET 12
29 #define END_USER_AGENT_VAL "\n"
30 #define MAX_LEN_USER_AGENT_VALUE 200
31
32
33
34
35 bool req_proc_create(struct server_req_proc* self, char* request) {
36     self->request = malloc(MAX_LEN_REQUEST);
37     if ( !self->request ) return false;
38     snprintf(self->request, MAX_LEN_REQUEST, "%s", request);
39     self->is_method_resource_valid = false;
40     return true;
41 }
42
43 void req_proc_destroy(struct server_req_proc* self) {
44     free(self->request);
45 }
46
47 //Verifica que el comando str se encuentre en request
48 //De ser así- devuelve 0
49 //en caso contrariodevuelve 1.
50 int str_check(const char* request, size_t len, const char* str) { //, char* err) {
51     for (int i = 0; i < len; ++i) {
52         if (request[i] != str[i]) {
53             return 1;
54         }
55     }
56     return 0;
57 }
58
59 char* req_porc_method_resource(struct server_req_proc* self) {
60     char* answer = malloc(MAX_LEN_MESSAGE);
61     if ( !answer ) return NULL;
62     enum error {METHRES_SUCCESS, METHOD_ERROR, RESOURCE_ERROR};
63     char* position = self->request + METHOD_OFFSET;
64     if (str_check(position, LEN_METHOD, CORRECT_METHOD)) {
65         snprintf(answer, MAX_LEN_MESSAGE, "%s", METHOD_ERROR_MESSAGE);
66         self->is_method_resource_valid = false; //no es necesario,

```

abr 01, 19 17:07

## server\_request\_processor.c

Page 2/2

```

67     return answer;                                //pero si llegase a cambiar
68 }                                                  //implementacion podria serlo
69 position = self->request + RESOURCE_OFFSET;
70 if (str_check(position, LEN_RESOURCE, CORRECT_RESOURCE)) {
71     snprintf(answer, MAX_LEN_MESSAGE, "%s", RESOURCE_ERROR_MESSAGE);
72     self->is_method_resource_valid = false;
73     return answer;
74 }
75 snprintf(answer, MAX_LEN_MESSAGE, "%s", METHRES_SUCCESS_MESSAGE);
76 self->is_method_resource_valid = true;
77 return answer;
78 }
79
80
81 bool req_porc_is_method_resource_valid(struct server_req_proc* self) {
82     return self->is_method_resource_valid;
83 }
84
85
86 char* req_porc_user_agent(struct server_req_proc* self) {
87     char* key_start = strstr(self->request, USER_AGENT_KEY);
88     if (!key_start) {
89         return NULL;
90     }
91     char* value_start = key_start + USER_AGENT_VAL_OFFSET;
92     size_t len_value = 0;
93     while (value_start[len_value] ^ \
94         strcmp(value_start + len_value, END_USER_AGENT_VAL)) {
95         len_value++;
96     }
97     char* value = malloc(MAX_LEN_USER_AGENT_VALUE);
98     if (!value) return NULL;
99     snprintf(value, MAX_LEN_USER_AGENT_VALUE, "%s", value_start);
100     char* value_end = strstr(value, "\n");
101     if (value_end) { //sino eof => strstr deja \n
102         *value_end = '\0';
103     }
104     return value;
105 }

```

abr 01, 19 17:07

## server\_list.h

Page 1/1

```

1  #ifndef SERVER_LIST_H
2  #define SERVER_LIST_H
3  #include <stdbool.h>
4
5  struct nodo;
6
7  struct List {
8      struct nodo* first;
9      struct nodo* last;
10     size_t len;
11 };
12
13 struct List;
14 void list_create(struct List *self);
15 void list_destroy(struct List *self);
16
17 /*
18 Dada una lista y una clave se agrega la clave a la lista
19 y se inicializa su valor.
20 En caso de ya existir la clave solamente se aumenta en uno su valor.
21 */
22 bool list_insert(struct List *self, char* _key);
23
24 /*
25 Imprime todos los datos almacenados de la forma
26 * <clave1>: <valor1>
27 * <clave2>: <valor2>
28 */
29 void list_print(struct List *self);
30
31 #endif
32

```

abr 01, 19 17:07

server\_list.c

Page 1/2

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include "server_list.h"
6
7  struct nodo {
8      char* key;
9      long int value;
10     struct nodo* next;
11 } typedef nodo_t;
12
13
14
15 bool nodo_create(nodo_t* self, char* _key) {
16     self->value = 1;
17     self->next = NULL;
18     size_t len_key = strlen(_key) + 1;
19     self->key = malloc(len_key);
20     if ( !self->key ) return false;
21     snprintf(self->key, len_key, "%s", _key);
22     return true;
23 }
24
25 void list_create(struct List *self) {
26     self->first = NULL;
27     self->last = NULL;
28     //self->len = 0;
29 }
30
31 void list_destroy(struct List *self) {
32     nodo_t* current = self->first;
33     nodo_t* aux;
34     while (current) {
35         aux = current->next;
36         free(current->key);
37         free(current);
38         current = aux;
39     }
40 }
41
42 bool list_insert(struct List *self, char* _key) {
43     bool status = true;
44     nodo_t* current = self->first;
45     while (current) {
46         if ( !strcmp(current->key, _key) ){
47             current->value++;
48             return true;
49         }
50         current = current->next;
51     }
52
53     nodo_t* nodo = malloc(sizeof(nodo_t));
54     if (!nodo) return false;
55     status = nodo_create(nodo, _key);
56     if ( !status ) {
57         free(nodo);
58         return false;
59     }
60     if (!self->first) {
61         self->first = nodo;
62         self->last = nodo;
63         return true;
64     }
65     self->last->next = nodo;
66     self->last = nodo;

```

abr 01, 19 17:07

server\_list.c

Page 2/2

```

67     return true;
68 }
69
70 void list_print(struct List *self){
71     //printf("# Estadisticas de visitantes\n");
72     nodo_t* current = self->first;
73     while (current) {
74         printf("\n* %s: %ld", current->key, current->value);
75         current = current->next;
76     }
77     printf("\n");
78 }

```

abr 01, 19 17:07

server.c

Page 1/3

```

1  #define _POSIX_C_SOURCE 200112L
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include <errno.h>
7  #include <stdbool.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <netdb.h>
12 #include <unistd.h>
13
14 #include <arpa/inet.h>
15
16
17 #include "server_list.h"
18 #include "server_socket.h"
19 #include "server_sensor.h"
20 #include "server_request_processor.h"
21 #include "server_tamplate.h"
22
23 bool process_client(struct List* list, char* temperature, \
24                   struct server_template *template, \
25                   struct server_socket* socket) {
26     bool status = true;
27     char* buffer = server_socket_receive_message(socket);
28     if ( !buffer ) return false;
29
30     struct server_req_proc processor;
31     status = req_proc_create(&processor, buffer);
32     if ( !status ) {
33         free(buffer);
34         return false;
35     }
36     char* answer = req_porc_method_resource(&processor);
37     if ( !answer ) return false;
38     server_socket_send_message(socket, answer, strlen(answer));
39
40     if ( ! req_porc_is_method_resource_valid(&processor) ) {
41         req_proc_destroy(&processor);
42         free(buffer);
43         free(answer);
44         return false;
45     }
46     char* us_ag = req_porc_user_agent(&processor);
47     if ( !us_ag ) return false;
48     status = list_insert(list, us_ag);
49     if ( !status ) {
50         free(buffer);
51         free(answer);
52         free(us_ag);
53         return false;
54     }
55
56     char* reply = server_template_cat(template, temperature);
57     if ( !reply ) {
58         free(buffer);
59         free(answer);
60         free(us_ag);
61         return false;
62     }
63
64     server_socket_send_message(socket, reply, strlen(reply));
65
66     req_proc_destroy(&processor);

```

abr 01, 19 17:07

server.c

Page 2/3

```

67     free(reply);
68     free(buffer);
69     free(answer);
70     free(us_ag);
71     return true;
72 }
73
74 int main(int argc, char* argv[]) {
75     if (argc != 4) {
76         fprintf(stderr, "Uso:\n./server <puerto> <input> [<template>]\n");
77         return 1;
78     }
79
80     char* port = argv[1];
81     char* sensor_filename = argv[2];
82     char* template_filename = argv[3];
83
84     struct server_sensor sensor;
85     if ( !server_sensor_create(&sensor, sensor_filename) ) return 1;
86
87     struct List list;
88     list_create(&list);
89
90     struct server_socket socket;
91     if ( !server_socket_create(&socket, port) ) {
92         server_sensor_destroy(&sensor);
93         list_destroy(&list);
94         return 1;
95     }
96     if ( !server_socket_start(&socket) ) {
97         server_socket_destroy(&socket);
98         server_sensor_destroy(&sensor);
99         list_destroy(&list);
100        return 1;
101    }
102
103    struct server_template template;
104    if ( !server_template_create(&template, template_filename) ) {
105        server_socket_destroy(&socket);
106        server_sensor_destroy(&sensor);
107        list_destroy(&list);
108        return 1;
109    }
110    bool was_last_client_valid = true;
111    char* temperature;
112    bool is_there_an_error = false;
113
114    while (true) {
115        if (was_last_client_valid) {
116            temperature = server_sensor_read(&sensor);
117        }
118        if ( !does_the_sensor_still_have_temperatures(&sensor) ) {
119            break;
120        }
121        if ( !temperature ) {
122            is_there_an_error = true;
123            break;
124        }
125        int s = server_socket_accept_client(&socket);
126        if (s == -1) {
127            is_there_an_error = true;
128            break;
129        }
130        was_last_client_valid = process_client(&list, temperature, \
131                                             &template, &socket);
132        server_socket_disable_client(&socket);

```



abr 01, 19 17:07

server.c

Page 3/3

```

133     if (was_last_client_valid) free(temperature);
134 }
135
136 printf( "# Estadisticas de visitantes\n" );
137 list_print(&list);
138
139 list_destroy(&list);
140 server_socket_destroy(&socket);
141 server_sensor_destroy(&sensor);
142 server_template_destoy(&template);
143 if (is_there_an_error) {
144     return 1;
145 }
146
147 return 0;
148 }

```

abr 01, 19 17:07

client\_socket.h

Page 1/1

```

1  #ifndef CLIENT_SOCKET_H
2  #define CLIENT_SOCKET_H
3  #include <stdlib.h>
4
5  struct client_socket {
6      size_t request_len;
7      char* request;
8      char* host;
9      char* port;
10     int skt;
11 };
12
13 /*
14  Guarda los parametros que se necesitaran al inciar el socket
15 */
16 void client_socket_create(struct client_socket *self, size_t _request_len,\
17                          char* _request, char* _host, char* _port);
18 void client_socket_destroy(struct client_socket *self);
19 /*
20  Crea el socket definiendo la familia, el tipo de socket y el protocolo
21  para poder conectarse al servidor por medio del port y host indicados.
22 */
23 bool client_socket_start(struct client_socket *self);
24 bool client_socket_receive_reponse(struct client_socket *self);
25 bool client_socket_send_request(struct client_socket *self);
26
27 #endif

```

abr 01, 19 17:07

client\_socket.c

Page 1/2

```

1  #define _POSIX_C_SOURCE 200112L
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #include <errno.h>
7  #include <stdbool.h>
8
9  #include <sys/types.h>
10 #include <sys/socket.h>
11 #include <netdb.h>
12 #include <unistd.h>
13 #include "client_socket.h"
14
15 #define REQUEST_MAX_LEN 2000
16 #define RESPONSE_MAX_LEN 2000
17 #define LEN_HOST 13
18 #define LEN_PORT 6
19
20
21 bool client_socket_send_request(struct client_socket *self) {
22     int s = 0;
23     int bytes_sent = 0;
24
25     while (bytes_sent < self->request_len) {
26         s = send(self->skt, &self->request[bytes_sent], \
27             self->request_len - bytes_sent, MSG_NOSIGNAL);
28
29         if (s <= 0) {
30             shutdown(self->skt, SHUT_RDWR);
31             close(self->skt);
32             return false;
33         } else {
34             bytes_sent += s;
35         }
36     }
37     shutdown(self->skt, SHUT_WR);
38     return true;
39 }
40
41 bool client_socket_receive_reponse(struct client_socket *self) {
42     int s = 0;
43     int bytes_receive = 0;
44     char response[RESPONSE_MAX_LEN];
45
46     while ( true ) {
47         s = recv(self->skt, &response[bytes_receive], \
48             RESPONSE_MAX_LEN - bytes_receive - 1, MSG_NOSIGNAL);
49         if (s < 0) { //socket error
50             return false;
51         } else if (s == 0) {
52             break;
53         } else {
54             bytes_receive = s;
55             response[bytes_receive] = 0;
56             printf("%s", response);
57             bytes_receive = 0;
58         }
59     }
60     printf("\n");
61     return true;
62 }
63
64 bool client_socket_start(struct client_socket *self) {
65     int s = 0;
66     bool are_we_connected = false;

```

abr 01, 19 17:07

client\_socket.c

Page 2/2

```

67
68     struct addrinfo hints;
69     struct addrinfo *result, *ptr;
70
71     self->skt = 0;
72
73     memset(&hints, 0, sizeof(struct addrinfo));
74     hints.ai_family = AF_INET;
75     hints.ai_socktype = SOCK_STREAM;
76     hints.ai_flags = 0;
77
78     s = getaddrinfo(self->host, self->port, &hints, &result);
79
80     if (s != 0) {
81         return false;
82     }
83
84
85     for (ptr = result; ptr != NULL ^ are_we_connected == false; \
86         ptr = ptr->ai_next) {
87         self->skt = socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol);
88         if (self->skt == -1) {
89             } else {
90                 s = connect(self->skt, ptr->ai_addr, ptr->ai_addrlen);
91                 if (s == -1) {
92                     close(self->skt);
93                 }
94                 are_we_connected = (s != -1);
95             }
96         }
97
98     freeaddrinfo(result);
99
100     return are_we_connected;
101 }
102
103 void client_socket_create(struct client_socket *self, size_t _request_len, \
104     char* _request, char* _host, char* _port) {
105     self->host = malloc(LEN_HOST);
106     self->port = malloc(LEN_PORT);
107     self->request = malloc(_request_len + 1);
108
109     self->request_len = _request_len;
110     snprintf(self->request, _request_len + 1, "%s", _request);
111     snprintf(self->host, LEN_HOST, "%s", _host);
112     snprintf(self->port, LEN_PORT, "%s", _port);
113 }
114
115 void client_socket_destroy(struct client_socket *self) {
116     shutdown(self->skt, SHUT_RDWR);
117     close(self->skt);
118
119     free(self->request);
120     free(self->host);
121     free(self->port);
122 }

```

abr 01, 19 17:07

## client\_file\_copier.h

Page 1/1

```

1  #ifndef CLIENT_FILE_COPIER_H
2  #define CLIENT_FILE_COPIER_H
3  #include <stdint.h>
4
5  struct file_copier {
6      char* filename;
7      char* path;
8      size_t* path_len;
9  };
10
11 void file_copier_create(struct file_copier* self,\
12                        char* filename, size_t* path_len, char* path);
13
14 /*
15  abre el archivo cuyo nombre tiene almacenado como atributo
16  y copia su contenido en un buffer tambi n almacenado como atributo
17  */
18 bool file_copier_start(struct file_copier* self);
19
20 #endif

```

abr 01, 19 17:07

## client\_file\_copier.c

Page 1/1

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <errno.h>
5  #include <stdbool.h>
6  #include "client_file_copier.h"
7
8  #define MAX_LEN_FILE 2000
9
10 void file_copier_create(struct file_copier* self,\
11                        char* filename, size_t* path_len, char* path){
12     self->filename = filename;
13     self->path = path;
14     self->path_len = path_len;
15 }
16
17 bool file_copier_start(struct file_copier* self) {
18     FILE* file = fopen(self->filename, "r");
19     if (!file) {
20         return false;
21     }
22
23     int i = 0;
24     while (!feof(file) ^ i < MAX_LEN_FILE) {
25         int c = fgetc(file);
26         self->path[i] = c;
27         i++;
28     }
29     self->path[i-2] = (int)'\0'; //me estaba leyendo un \n y un eof o algo asi
30     fclose(file);
31     *self->path_len = i - 2;
32     return true;
33 }
34

```

abr 01, 19 17:07

client.c

Page 1/1

```

1  #define _POSIX_C_SOURCE 200112L
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <errno.h>
6  #include <stdbool.h>
7
8  #include <sys/types.h>
9  #include <sys/socket.h>
10 #include <netdb.h>
11 #include <unistd.h>
12
13 #include "client_socket.h"
14
15 #include "client_file_copier.h"
16
17 #define MAX_LEN_FILENAME 100
18 #define MAX_LEN_FILE 2000
19
20 int main(int argc, char* argv[]) {
21     if (argc != 3 ^ argc != 4) {
22         fprintf(stderr, "Uso:\n/client <direccion> <puerto> [<input>]\n");
23         return 1;
24     }
25     char filename[MAX_LEN_FILENAME];
26     if (argc == 3) {
27         char* status = fgets(filename, MAX_LEN_FILENAME, stdin);
28         filename[strlen(filename) - 1] = '\0';
29         if (!status) {
30             return 1;
31         }
32     } else {
33         snprintf(filename, MAX_LEN_FILENAME, "%s", argv[3]);
34     }
35
36     struct file_copier copier;
37     char path[MAX_LEN_FILE];
38     size_t path_len;
39     file_copier_create(&copier, filename, &path_len, path);
40
41     if (!file_copier_start(&copier)){
42         return 1;
43     }
44
45     char* host = argv[1];
46     char* port = argv[2];
47     struct client_socket socket;
48     client_socket_create(&socket, path_len, path, host, port);
49     if (!client_socket_start(&socket)) {
50         return 1;
51     }
52     client_socket_send_request(&socket);
53     client_socket_receive_reponse(&socket);
54     client_socket_destroy(&socket);
55     return 0;
56 }

```

abr 01, 19 17:07

Table of Content

Page 1/1

1	Table of Contents					
2	1 server_template.h...	sheets	1 to	1 ( 1) pages	1- 1	15 lines
3	2 server_template.c...	sheets	1 to	1 ( 1) pages	2- 2	52 lines
4	3 server_socket.h.....	sheets	2 to	2 ( 1) pages	3- 3	37 lines
5	4 server_socket.c.....	sheets	2 to	3 ( 2) pages	4- 6	134 lines
6	5 server_sensor.h.....	sheets	4 to	4 ( 1) pages	7- 7	24 lines
7	6 server_sensor.c.....	sheets	4 to	4 ( 1) pages	8- 8	45 lines
8	7 server_request_processor.h	sheets	5 to	5 ( 1) pages	9- 9	36 lines
9	8 server_request_processor.c	sheets	5 to	6 ( 2) pages	10- 11	106 lines
10	9 server_list.h.....	sheets	6 to	6 ( 1) pages	12- 12	33 lines
11	10 server_list.c.....	sheets	7 to	7 ( 1) pages	13- 14	79 lines
12	11 server.c.....	sheets	8 to	9 ( 2) pages	15- 17	149 lines
13	12 client_socket.h.....	sheets	9 to	9 ( 1) pages	18- 18	28 lines
14	13 client_socket.c.....	sheets	10 to	10 ( 1) pages	19- 20	123 lines
15	14 client_file_copier.h	sheets	11 to	11 ( 1) pages	21- 21	21 lines
16	15 client_file_copier.c	sheets	11 to	11 ( 1) pages	22- 22	35 lines
17	16 client.c.....	sheets	12 to	12 ( 1) pages	23- 23	57 lines