

## Sistema manejador de bases de datos

### Introducción

El proyecto tiene como objetivo crear un Sistema Manejador de Bases de Datos (DBMS) en una arquitectura cliente-servidor, con sockets TCP/IP y con conexión remota. Además, se solicitó que el código estuviera escrito en lenguaje C.

Para el uso de sockets en C, se partió del siguiente diagrama, el cual describe el flujo típico de una aplicación y los métodos correspondientes a cada etapa cuando usamos sockets.

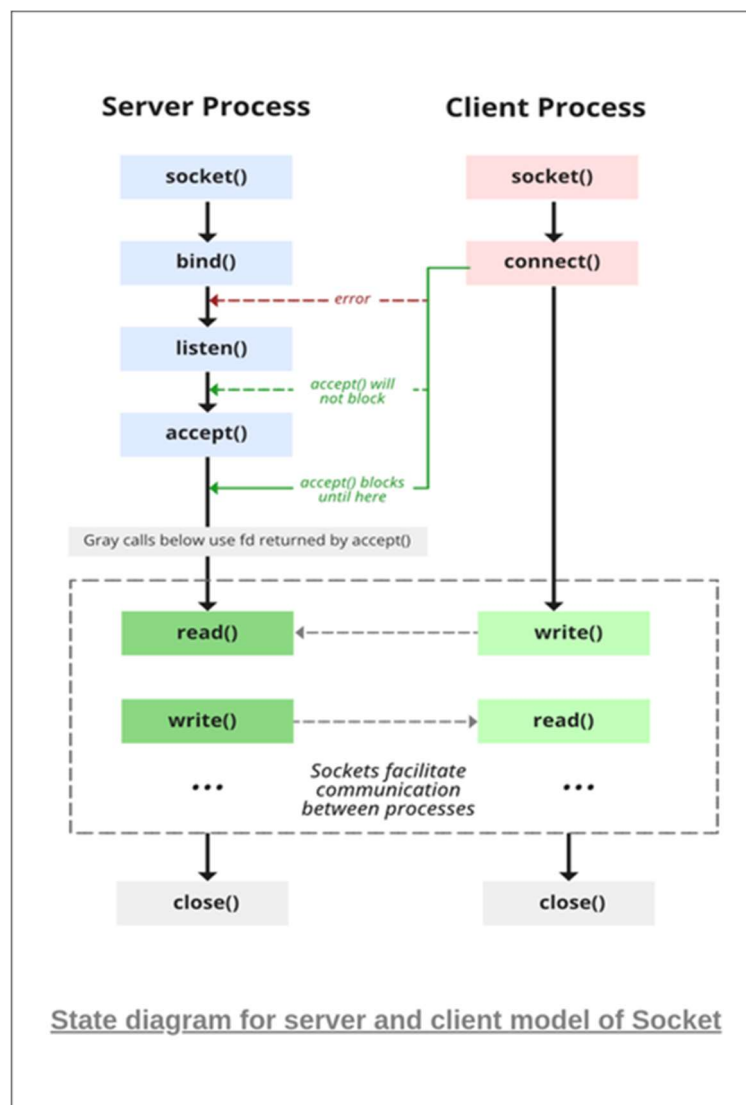


Figura 1. Diagrama de estados del cliente y servidor

## Estructura de archivos

El proyecto se dividió en archivos para facilitar su entendimiento, además de los 2 archivos esenciales (cliente y servidor), se crearon 2 bibliotecas.

A continuación, se describen los archivos que componen el sistema y una breve descripción de ellos:

- server.c: Se encarga de manejar la lógica del servidor, siguiendo el diagrama de la Figura 1. Aquí definimos que nuestro socket será de tipo TCP/IP (SOCK\_STREAM) y que usaremos direcciones IPv4 (AF\_INET)
- cliente.c: Se encarga de manejar la lógica del cliente, siguiendo el diagrama de la Figura 1
- query.c: Descompone el comando ingresado por el usuario en los campos necesarios para ejecutar la sentencia SQL y posteriormente llama al método que se encargue de ejecutar la acción
- CRUD.c: Define las operaciones CRUD sobre archivos que usaremos para simular las tablas

## Código

### ○ Servidor

```
28 void sigchld_handler(int s) {  
29     while(wait(NULL) > 0);  
30 }
```

Figura 2. Función sichld\_handler

```
32 void create_socket() {  
33     int yes = 1;  
34     // Socket creation  
35     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {  
36         perror("Server-socket() error");  
37         exit(1);  
38     } else  
39     |     printf("Server-socket() sockfd is OK...\n");  
40     // Set socket options <Prevents 'address already in use'>  
41     if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1) {  
42         perror("Server-setsockopt() error");  
43         exit(1);  
44     }  
45     else  
46     |     printf("Server-setsockopt is OK...\n");  
47 }
```

Figura 3. Función para la llamada create()

```
49 void bind_socket() {
50     struct sockaddr_in my_addr;           // Información sobre mi dirección
51     char *ip = "192.168.0.141";
52
53     my_addr.sin_family = AF_INET;         // Ordenación de bytes de la máquina
54     my_addr.sin_port = htons(MYPORT);     // short, Ordenación de bytes de la red
55     my_addr.sin_addr.s_addr = inet_addr(ip); // Rellenar con mi dirección IP
56
57     printf("Server-Using %s and port %d...\n", inet_ntoa(my_addr.sin_addr), MYPORT);
58
59     memset(&(my_addr.sin_zero), '\0', 8); // Poner a cero el resto de la estructura
60
61     if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
62         perror("Server-bind() error");
63         exit(1);
64     } else
65         printf("Server-bind() is OK...\n");
66 }
67 }
```

Figura 4. Función para la llamada bind()

```
69 void listen_socket() {
70     if (listen(sockfd, BACKLOG) == -1) {
71         perror("Server-listen() error");
72         exit(1);
73     }
74     printf("Server-listen() is OK...Listening...\n");
75 }
```

Figura 5. Función para la llamada listen()

```
77 void sigaction_socket() {
78     struct sigaction sa;
79
80     sa.sa_handler = sigchld_handler; // Eliminar procesos muertos
81     sigemptyset(&sa.sa_mask);
82     sa.sa_flags = SA_RESTART;
83     if (sigaction(SIGCHLD, &sa, NULL) == -1) {
84         perror("Server-sigaction() error");
85         exit(1);
86     } else
87         printf("Server-sigaction() is OK...\n");
88 }
```

Figura 6. Función para eliminar los procesos muertos

```
90 void accept_socket() {
91     struct sockaddr_in their_addr; // información sobre la dirección del cliente
92     int sin_size;
93
94     sin_size = sizeof(struct sockaddr_in);
95     if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -1) {
96         perror("Server-accept() error");
97     } else
98         printf("Server-accept() is OK...\n");
99     printf("Server-new socket, new_fd is OK...\n");
100    printf("Server: Got connection from %s\n", inet_ntoa(their_addr.sin_addr));
```

Figura 7. Función para la llamada accept()

```
104 void receive_message() {
105     int numbytes;
106     char buf[MAXDATASIZE];
107
108     if((numbytes = recv(new_fd, buf, MAXDATASIZE-1, 0)) == -1) {
109         perror("recv()");
110         exit(1);
111     } else
112         printf("Servidor-The recv() is OK...\n");
113
114     buf[numbytes] = '\0';
115     printf("Servidor-Received: %s", buf);
116     define_operation(buf, new_fd);
117 }
```

Figura 8. Función para recibir mensajes de los clientes

```
1  /*
2  |   Modificado por Andrea Garcia Ruiz, el 4 de octubre de 2022
3  |   server.c -- Ejemplo de servidor de sockets de flujo
4  */
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <unistd.h>
8  #include <errno.h>
9  #include <string.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <netinet/in.h>
13 #include <arpa/inet.h>
14 #include <sys/wait.h>
15 #include <signal.h>
16 #include "query.h"
17
18 #define MYPOR 3490    // Puerto al que conectarán los usuarios
19
20 #define BACKLOG 100   // Cuántas conexiones pendientes se mantienen en cola
21
22 #define LINE_MAX 200
23
24 #define MAXDATASIZE 300
25
26 int sockfd, new_fd;
27
28 > void sigchld_handler(int s) {~
31
32 > void create_socket() {~
48
49 > void bind_socket() {~
68
69 > void listen_socket() {~
76
77 > void sigaction_socket() {~
89
90 > void accept_socket() {~
103
104 > void receive_message() {~
118
119 int main(int argc, char *argv[])
120 {
121     int numbytes;      // Escuchar sobre sock_fd, nuevas conexiones sobre new_fd
122     char buf[MAXDATASIZE];
123
124     create_socket();
125     bind_socket();
126     listen_socket();
127     sigaction_socket();
128
129     while(1) { // main accept() loop
130         accept_socket();
131         if (!fork()) { // Este es el proceso hijo
132             close(sockfd); // El hijo no necesita este descriptor
133             while (1)
134             {
135                 receive_message();
136             }
137         }
138         printf("Este es el proceso padre, cierra el descriptor del socket cliente y se regresa a esperar otro cliente\n");
139         close(new_fd); // El proceso padre no necesita este descriptor
140         printf("Server-new socket, new_fd closed successfully...\n");
141     }
142
143     return 0;
144 }
145
```

Figura 9. Código completo del servidor

## ○ Cliente

```
28  /*
29  |  Obtiene la informacion del servidor al que nos estamos conectando
30  */
31  void get_host_info() {
32  |  if((he = gethostbyname(ip_address)) == NULL)
33  |  {
34  |      perror("gethostbyname()");
35  |      exit(1);
36  |  }
37  |  else
38  |      printf("Client-The remote host is: %s\n", ip_address);
39  }
```

Figura 10. Obtiene la información del host al que nos conectamos

```
42  void create_socket() {
43  |  if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
44  |  {
45  |      perror("socket()");
46  |      exit(1);
47  |  }
48  |  else
49  |      printf("Client-The socket() sockfd is OK...\n");
50  }
```

Figura 11. Función para la llamada create()

```
52  void connect_socket() {
53  |  struct sockaddr_in their_addr;
54  |  // host byte order
55  |  their_addr.sin_family = AF_INET;
56  |  // short, network byte order
57  |  printf("Server-Using %s and port %d...\n", ip_address, PORT);
58  |  their_addr.sin_port = htons(PORT);
59  |  their_addr.sin_addr = *((struct in_addr *)he->h_addr);
60  |  // zero the rest of the struct
61  |  memset(&(their_addr.sin_zero), '\0', 8);
62  |  if(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
63  |  {
64  |      perror("connect()");
65  |      exit(1);
66  |  }
67  |  else
68  |      printf("Client-The connect() is OK...\n");
69  }
```

Figura 12. Función para la llamada connect()



```
71 void send_message(char const *message) {  
72     if (send(sockfd, message, strlen(message), 0) == -1)  
73     | perror("Server-send() error");  
74 }
```

Figura 13. Función para enviar mensajes al servidor

```
76 void receive_message() {  
77     int numbytes;  
78     char buf[MAXDATASIZE];  
79  
80     if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1)  
81     {  
82         perror("recv()");  
83         exit(1);  
84     }  
85     else  
86     | printf("Client-The recv() is OK...\n");  
87  
88     buf[numbytes] = '\0';  
89     printf("Client-Received: %s", buf);  
90 }
```

Figura 14. Función para recibir mensajes del servidor

```
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <errno.h>
10 #include <string.h>
11 #include <netdb.h>
12 #include <sys/types.h>
13 #include <netinet/in.h>
14 #include <sys/socket.h>
15
16 // the port client will be connecting to
17 #define PORT 3490
18 // max number of bytes we can get at once
19 #define MAXDATASIZE 300
20
21 #define LINE_MAX 200
22
23 struct hostent *he;
24
25 int sockfd;
26
27 char* ip_address;
28
29 /*
30 | Obtiene la informacion del servidor al que nos estamos conectando
31 */
32 > void get_host_info() { ...
41
42 > void create_socket() { ...
51
52 > void connect_socket() { ...
70
71 > void send_message(char const *message) { ...
75
76 > void receive_message() { ...
91
92 int main(int argc, char *argv[])
93 {
94     int salir = 1;
95     char linea1[LINE_MAX];
96     // Si no enviamos como parametro la direccion ip del servidor, lanza un error
97     if(argc != 2)
98     {
99         fprintf(stderr, "Client-Usage: %s hostname_del_servidor\n", argv[0]);
100        exit(1);
101    }
102    // -----
103    ip_address = argv[1];
104    get_host_info();
105    create_socket();
106    connect_socket();
107
108    while (salir)
109    {
110        printf("Escribe un mensaje a enviar\n");
111        fgets(linea1, LINE_MAX, stdin);
112        send_message(linea1);
113        if (strcmp(linea1, "SALIR\n") == 0)
114        {
115            printf("Client-Closing sockfd\n");
116            close(sockfd);
117            return 0;
118        } else
119        {
120            receive_message();
121        }
122    }
123 }
```

Figura 15. Código completo del cliente



## ○ CRUD

```
C: CRUD.h > ...
1  #ifndef _LIBRERIA
2  #define _LIBRERIA
3
4  void set_socket(int s);
5  void create_table(char const *name, char const *columns);
6  void insert_values(char const *table, char const *values);
7  void select_all(char const *table);
8  void update_record(char const *table, char const *column, char const *new_value, char const *where);
9  void delete_record(char const *table, char const *where);
10
11 #include "CRUD.c"
12 #endif
```

Figura 16. Archivo encabezado de CRUD.c

```
13  int my_socket;
14
15  /*
16   |   Define el socket al que le comunicaremos los resultados
17  */
18  void set_socket(int s) {
19   |   my_socket = s;
20  }
21
22  /*
23   |   Nos ayuda a enviarle mensajes al cliente
24  */
25  void send_message(char const *message) {
26   |   if (send(my_socket, message, strlen(message), 0) == -1)
27   |       perror("Server-send() error");
28  }
```

Figura 17. Funciones que permiten enviar retroalimentación a través de un socket

```
37  /*
38   |   Crea un archivo con el nombre especificado y coloca como primera linea lo que contiene el parametro
39  */
40  void create_table(char const *name, char const *columns) {
41   |   FILE* file;
42   |   file = fopen(name, "wt");
43   |   fprintf(file, "%s\n", columns);
44   |   fclose(file);
45   |   send_message("[-] Table created\n");
46  }
```

Figura 18. Función que permite crear una tabla

```
48  /*
49  |   Agrega un nuevo registro al final del archivo, incrementando el valor del ultimo id
50  */
51  void insert_values(char const *table, char const *values) {
52      FILE* file;
53      char line[200];
54      int num_lines = 0, id;
55      // Abre el archivo en modo lectura
56      file = fopen(table, "rt");
57      // Cuenta las filas de la tabla indicada, es decir, cuenta el numero de lineas
58      while(fgets(line, 200, file) != NULL)
59      |   num_lines++;
60      fclose(file);
61      // Abre el archivo nuevamente pero en modo adición
62      file = fopen(table, "at");
63      if (num_lines > 1)
64      {
65          // Si no es el primer registro, obtiene el ultimo y le suma 1
66          char* token = strtok(line, ",");
67          id = string_to_int(token) + 1;
68      } else {
69          // Si es el primer registro, le corresponde el id 1
70          id = 1;
71      }
72      // Cuando ya se conoce el id, se agregan los valores a la tabla
73      fprintf(file, "%d,%s\n", id, values);
74      fclose(file);
75      send_message("[-] Record added\n");
76  }
```

Figura 19. Función que permite insertar registros

```
78  /*
79  |   Devuelve el contenido del archivo indicado
80  */
81  void select_all(char const *table) {
82      FILE* file;
83      char line[200];
84      void send_message(const char *message)
85      |   Nos ayuda a enviarle mensajes al cliente
86      send_message("\n-----\n");
87      send_message(line);
88      send_message("-----\n");
89      while (fgets(line, 200, file) != NULL) {
90          |   send_message(line);
91      }
92      fclose(file);
93  }
```

Figura 20. Función que permite mostrar los registros de una tabla

```
95 void update_record(char const *table, char const *column, char const *new_value, char const *where) {
96     FILE* file;
97     FILE* temp;
98     char line[200];
99     char table_structure[200];
100     char* table_ptr = table;
101     char* token_column;
102     char* token_column;
103     char* token_value;
104     char* current_id;
105     int num_line = 1, column_exists = -1, id, first_time = 1, num_column = 0;
106     // Archivo del que leemos el contenido de la tabla
107     file = fopen(table, "rt");
108     // Archivo auxiliar en el que colocamos los registros actualizados
109     temp = fopen("temp", "wt");
110     id = string_to_int(where);
```

Figura 21. Declaración de la función update e inicialización de variables

```
111 if(id > 0) {
112     if (strcmp(column, "Id") == 0) {
113         send_message("[~] Operation update failed <Id cannot be updated>\n");
114     } else {
115         // Obtiene las columnas de la tabla
116         fgets(table_structure, 200, file);
117         // Guarda el contenido de table_structure en temp
118         fputs(table_structure, temp);
119         // Quita el salto de linea al final del renglon
120         table_st_sin_salto = strtok(table_structure, "\n");
121         // Separa la cadena por comas y guarda en token_column el nombre de la primera columna
122         token_column = strtok(table_st_sin_salto, ",");
123         while (token_column != NULL) {
124             // Compara el nombre de la columna con el que dio el usuario y guarda el numero de columna
125             if(strcmp(token_column, column) == 0) {
126                 column_exists = num_column;
127             }
128             num_column++;
129             // Obtiene la siguiente columna
130             token_column = strtok(NULL, ",");
131         }
132
133         if (column_exists != -1) {
134             // Itera sobre todos los registros guardado en el archivo
135             while (fgets(line, 200, file) != NULL)
136             {
137                 // Quita el salto de linea de cada linea del archivo
138                 line_sin_salto = strtok(line, "\n");
139                 if (num_line == id) {
140                     token_value = strtok(line_sin_salto, ",");
141                     fprintf(temp, "%d", id);
142                     // Itera sobre todas las columnas de la linea
143                     for (int i = 1; i < num_column; i++) {
144                         token_value = strtok(NULL, ",");
145                         if (i == column_exists) {
146                             // Cuando la columna coincide con la especificada, coloca el nuevo valor
147                             fprintf(temp, "%s", new_value);
148                             send_message("[~] Record updated\n");
149                         } else {
150                             // Si no coincide coloca el valor del archivo
151                             fprintf(temp, "%s", token_value);
152                         }
153                     }
154                     fputs("\n", temp);
155                 } else {
156                     fprintf(temp, "%s\n", line);
157                 }
158                 num_line++;
159             }
160             if (num_line-1 < id) {
161                 send_message("[~] Operation update failed <Id doesn't exist>\n");
162             }
163             // Cuando hay cambios...
164             // Se borra el archivo original
165             remove(table);
166             // Y temp se convierte en la tabla
167             rename("temp", table);
168         } else {
169             send_message("[~] Operation update failed <Column doesn't exist>\n");
170         }
171     }
172 } else {
173     send_message("[~] Operation update failed <Invalid id>\n");
174 }
175 fclose(file);
176 fclose(temp);
177 }
```

Figura 22. Cuerpo de la función update

### ○ Reconocer consultas

```
1  #ifndef _LIBRERIA2
2  #define _LIBRERIA2
3
4  void define_operation(char query[], int socket);
5
6  #include "query.c"
7  #endif
8
```

Figura 23. Archivo encabezado de query.c

```
1  /*
2   Creado por Andrea Garcia Ruiz, el 05/10/2022
3
4   Las funciones descritas en este archivo separan la sentencia SQL y llaman a las operaciones CRUD
5  */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10 #include "query.h"
11 #include "CRUD.h"
12 /*
13  Dependiendo de la operacion deseada se recuperan los parametros necesarios.
14
15  La sintaxis sobre la que se basa es la siguiente:
16
17  CREATE -> CREATE nombre_tabla (columna1, columna2, ...)
18  INSERT -> INSERT INTO nombre_tabla VALUES (valor1, valor2, ...)
19  UPDATE -> UPDATE nombre_tabla SET nombre_columna=nuevo_valor WHERE ID=id_registro
20  DELETE -> DELETE FROM nombre_tabla WHERE ID=id_registro
21  SELECT -> SELECT * FROM nombre_tabla
22  */
23 void define_operation(char query[], int socket) {
24     char* piece;
25     char* table_name;
26     char* values;
27     char* column;
28     char* column_name;
29     char* id;
30     set_socket(socket);
31     piece = strtok(query, " ");
32     if (strcmp(piece, "CREATE") == 0) { ...
50     else if(strcmp(piece, "INSERT") == 0) { ...
68     else if(strcmp(query, "UPDATE") == 0) { ...
91     else if(strcmp(query, "SELECT") == 0) { ...
106    else if(strcmp(query, "DELETE") == 0) { ...
124    else
125    |    send_message("[-] Unrecognized command\n");
126 }
127
```

Figura 24. Contenido del archivo query.c



```
32     if (strcmp(piece, "CREATE") == 0) {
33         for (int i = 0; i < 3; i++)
34         {
35             if (i == 1)
36             {
37                 table_name = strtok(NULL, " ");
38             } else if (i == 2)
39             {
40                 values = strtok(NULL, " ");
41             } else {
42                 strtok(NULL, " ");
43             }
44         }
45     }
46     memmove(&values[0], &values[1], strlen(values));
47     values[strlen(values)-2] = '\\0';
48     create_table(table_name, values);
49 }
```

Figura 25. Fragmento de código para reconocer una sentencia CREATE

```
50     else if (strcmp(piece, "INSERT") == 0) {
51         for (int i = 0; i < 4; i++)
52         {
53             if (i == 1)
54             {
55                 table_name = strtok(NULL, " ");
56             } else if (i == 3)
57             {
58                 values = strtok(NULL, " ");
59             } else {
60                 strtok(NULL, " ");
61             }
62         }
63     }
64     memmove(&values[0], &values[1], strlen(values));
65     values[strlen(values)-2] = '\\0';
66     insert_values(table_name, values);
67 }
```

Figura 26. Fragmento de código para reconocer una sentencia INSERT



```
68     else if(strcmp(query, "UPDATE") == 0) {
69         for (int i = 0; i < 5; i++)
70         {
71             if (i == 0)
72             {
73                 table_name = strtok(NULL, " ");
74             } else if (i == 2)
75             {
76                 column = strtok(NULL, " ");
77             } else if (i == 4)
78             {
79                 id = strtok(NULL, " ");
80             }
81             else {
82                 strtok(NULL, " ");
83             }
84         }
85         column_name = strtok(column, "=");
86         values = strtok(NULL, "=");
87         strtok(id, "=");
88         update_record(table_name, column_name, values, strtok(NULL, "="));
89     }
90 }
```

Figura 27. Fragmento de código para reconocer una sentencia UPDATE

```
91     else if(strcmp(query, "SELECT") == 0) {
92         for (int i = 0; i < 3; i++)
93         {
94             if (i == 2)
95             {
96                 table_name = strtok(NULL, " ");
97             }
98             else {
99                 strtok(NULL, " ");
100             }
101         }
102         select_all(strtok(table_name, "\n"));
103     }
104 }
105 }
```

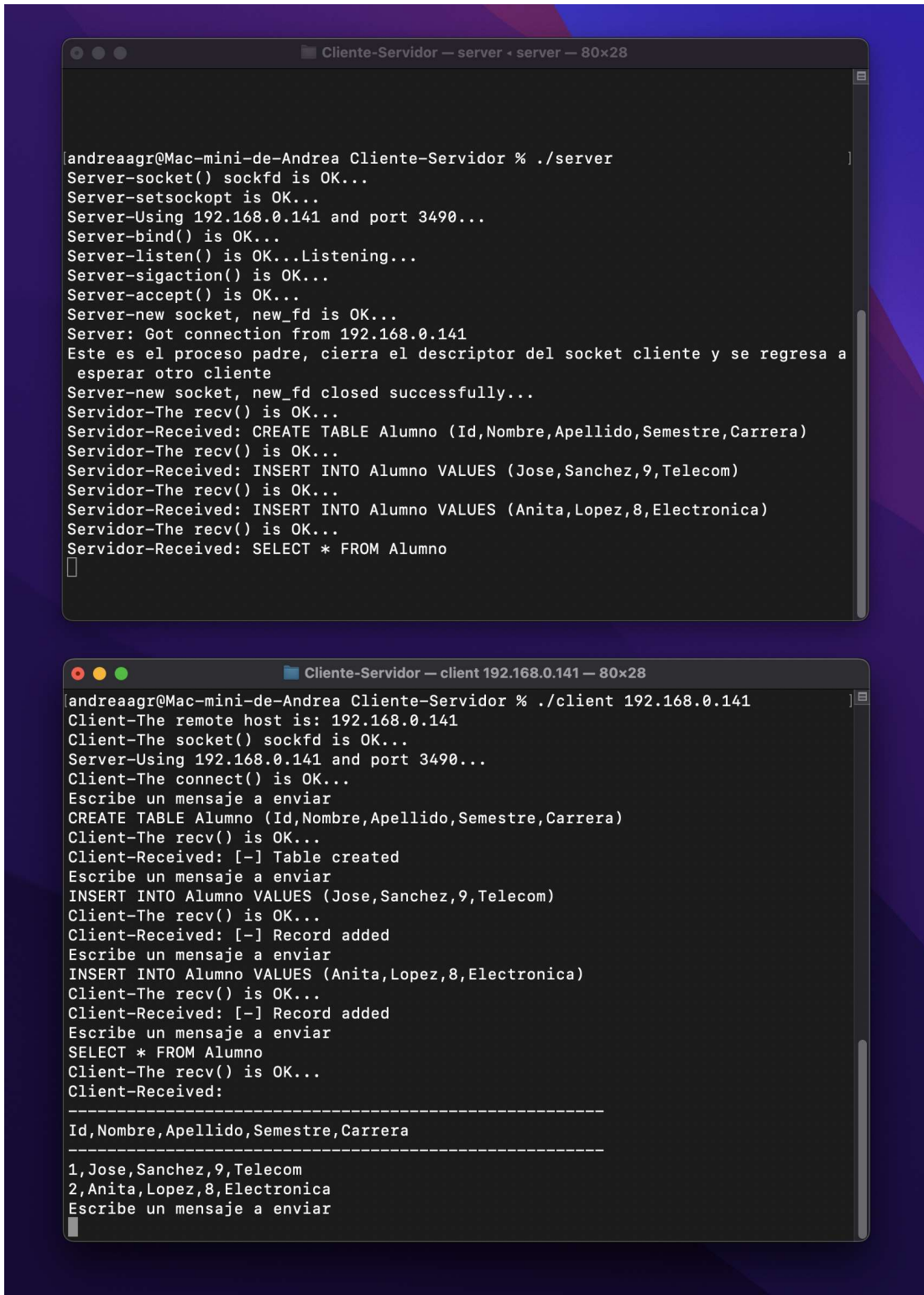
Figura 28. Fragmento de código para reconocer una sentencia SELECT

```
106  ✓   else if(strcmp(query, "DELETE") == 0) {
107  ✓       for (int i = 0; i < 4; i++)
108           {
109  ✓         if (i == 1)
110             {
111                 table_name = strtok(NULL, " ");
112             }
113  ✓         else if (i == 3)
114             {
115                 id = strtok(NULL, " ");
116             }
117  ✓         else {
118             }
119             strtok(NULL, " ");
120         }
121     strtok(id, "=");
122     delete_record(table_name, strtok(NULL, "="));
123 }
```

Figura 29. Fragmento de código para reconocer una sentencia DELETE

## Pruebas

### ○ CREATE, INSERT y SELECT



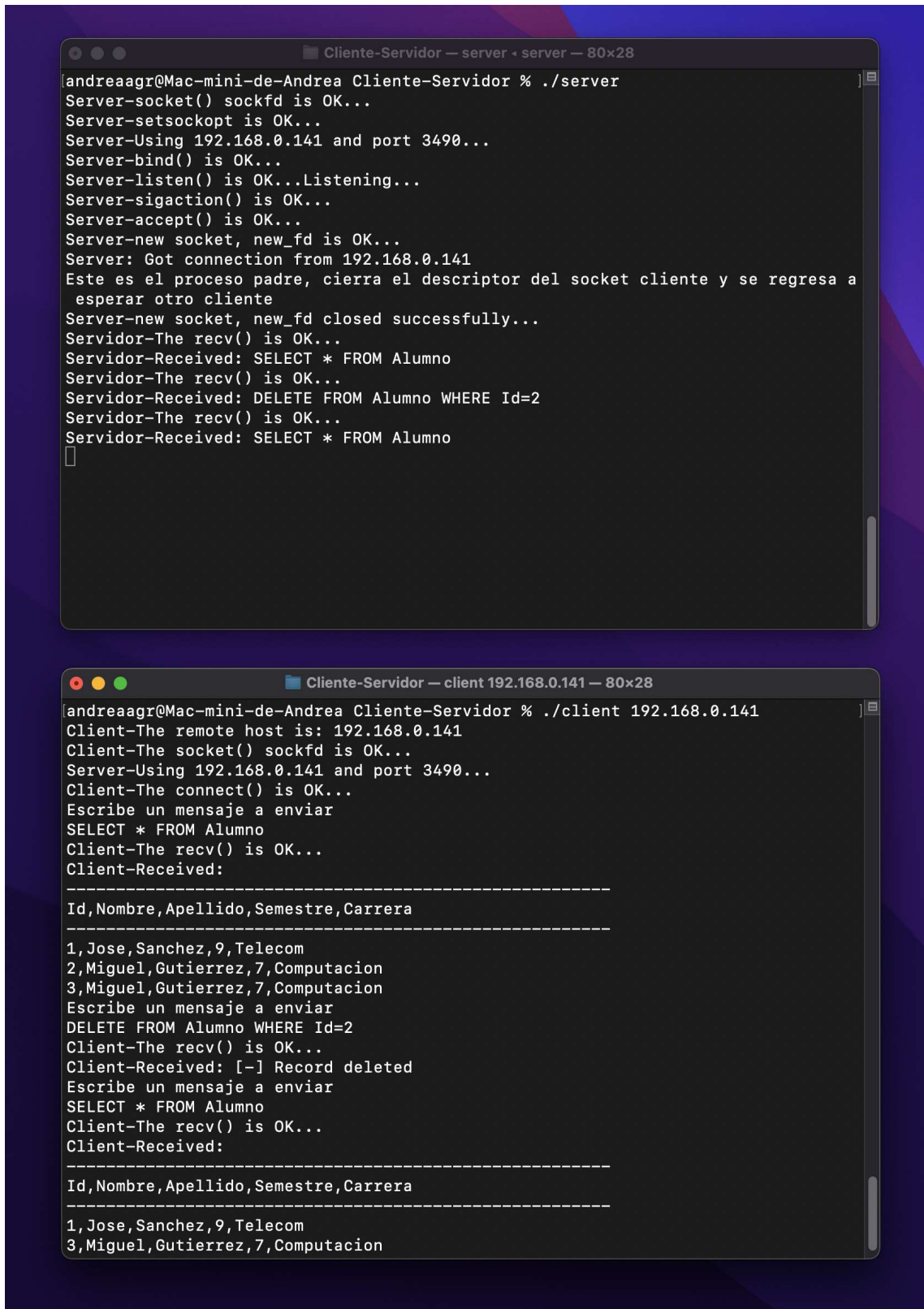
```
andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./server
Server-socket() sockfd is OK...
Server-setsockopt is OK...
Server-Using 192.168.0.141 and port 3490...
Server-bind() is OK...
Server-listen() is OK...Listening...
Server-sigaction() is OK...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 192.168.0.141
Este es el proceso padre, cierra el descriptor del socket cliente y se regresa a
esperar otro cliente
Server-new socket, new_fd closed successfully...
Servidor-The recv() is OK...
Servidor-Received: CREATE TABLE Alumno (Id,Nombre,Apellido,Semestre,Carrera)
Servidor-The recv() is OK...
Servidor-Received: INSERT INTO Alumno VALUES (Jose,Sanchez,9,Telecom)
Servidor-The recv() is OK...
Servidor-Received: INSERT INTO Alumno VALUES (Anita,Lopez,8,Electronica)
Servidor-The recv() is OK...
Servidor-Received: SELECT * FROM Alumno
█

andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./client 192.168.0.141
Client-The remote host is: 192.168.0.141
Client-The socket() sockfd is OK...
Server-Using 192.168.0.141 and port 3490...
Client-The connect() is OK...
Escribe un mensaje a enviar
CREATE TABLE Alumno (Id,Nombre,Apellido,Semestre,Carrera)
Client-The recv() is OK...
Client-Received: [-] Table created
Escribe un mensaje a enviar
INSERT INTO Alumno VALUES (Jose,Sanchez,9,Telecom)
Client-The recv() is OK...
Client-Received: [-] Record added
Escribe un mensaje a enviar
INSERT INTO Alumno VALUES (Anita,Lopez,8,Electronica)
Client-The recv() is OK...
Client-Received: [-] Record added
Escribe un mensaje a enviar
SELECT * FROM Alumno
Client-The recv() is OK...
Client-Received:

-----
Id,Nombre,Apellido,Semestre,Carrera
-----
1,Jose,Sanchez,9,Telecom
2,Anita,Lopez,8,Electronica
Escribe un mensaje a enviar
```

Figura 30. Creación de una tabla, inserción de 2 registros y despliegue de la información

## ○ Delete y select



The image shows two terminal windows from a macOS environment. The top window is titled 'Cliente-Servidor — server — 80x28' and shows the server's execution of a C program. The bottom window is titled 'Cliente-Servidor — client 192.168.0.141 — 80x28' and shows the client's execution of a C program.

**Server Terminal Output:**

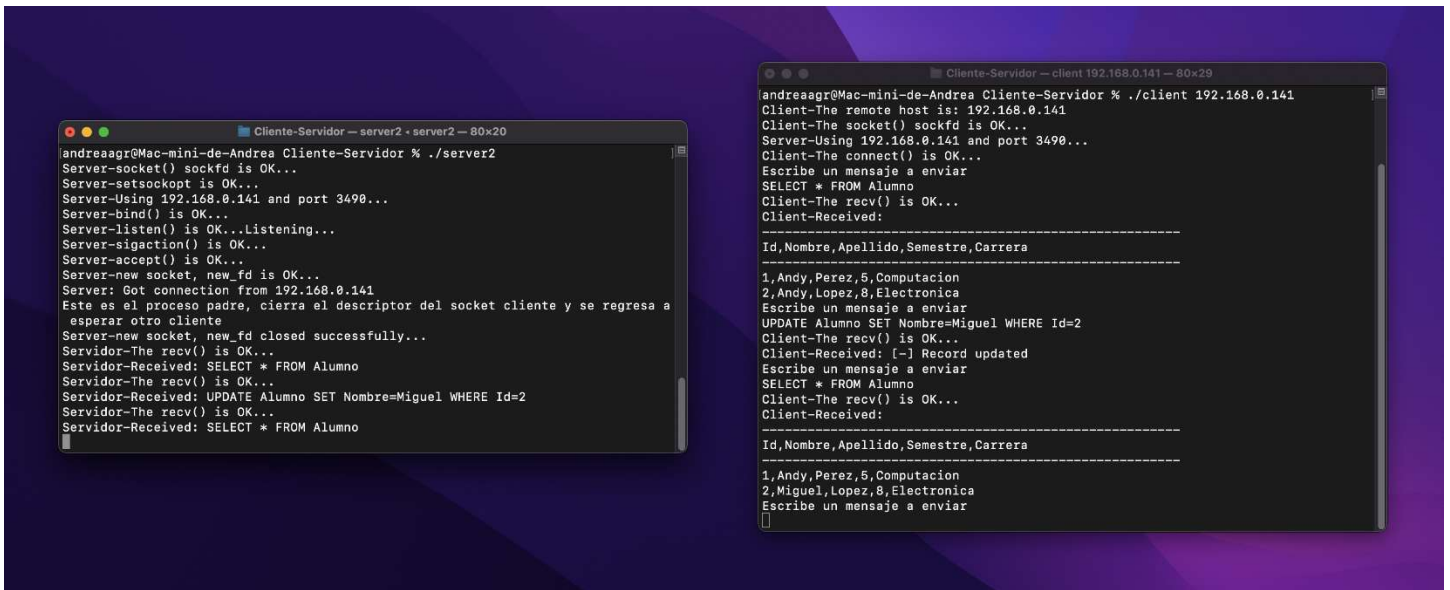
```
andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./server
Server-socket() sockfd is OK...
Server-setsockopt is OK...
Server-Using 192.168.0.141 and port 3490...
Server-bind() is OK...
Server-listen() is OK...Listening...
Server-sigaction() is OK...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 192.168.0.141
Este es el proceso padre, cierra el descriptor del socket cliente y se regresa a
esperar otro cliente
Server-new socket, new_fd closed successfully...
Servidor-The recv() is OK...
Servidor-Received: SELECT * FROM Alumno
Servidor-The recv() is OK...
Servidor-Received: DELETE FROM Alumno WHERE Id=2
Servidor-The recv() is OK...
Servidor-Received: SELECT * FROM Alumno
█
```

**Client Terminal Output:**

```
andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./client 192.168.0.141
Client-The remote host is: 192.168.0.141
Client-The socket() sockfd is OK...
Server-Using 192.168.0.141 and port 3490...
Client-The connect() is OK...
Escribe un mensaje a enviar
SELECT * FROM Alumno
Client-The recv() is OK...
Client-Received:
-----
Id,Nombre,Apellido,Semestre,Carrera
-----
1,Jose,Sanchez,9,Telecom
2,Miguel,Gutierrez,7,Computacion
3,Miguel,Gutierrez,7,Computacion
Escribe un mensaje a enviar
DELETE FROM Alumno WHERE Id=2
Client-The recv() is OK...
Client-Received: [-] Record deleted
Escribe un mensaje a enviar
SELECT * FROM Alumno
Client-The recv() is OK...
Client-Received:
-----
Id,Nombre,Apellido,Semestre,Carrera
-----
1,Jose,Sanchez,9,Telecom
3,Miguel,Gutierrez,7,Computacion
```

Figura 31. Eliminación de un registro

## ○ UPDATE y SELECT



The image shows two terminal windows side-by-side, illustrating a client-server interaction for a database application. The left window is the server process, and the right window is the client process.

**Left Window (Server):**

```
andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./server2
Server-socket() sockfd is OK...
Server-setsockopt is OK...
Server-Using 192.168.0.141 and port 3490...
Server-bind() is OK...
Server-listen() is OK...Listening...
Server-sigaction() is OK...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 192.168.0.141
Este es el proceso padre, cierra el descriptor del socket cliente y se regresa a
esperar otro cliente
Server-new socket, new_fd closed successfully...
Servidor-The recv() is OK...
Servidor-Received: SELECT * FROM Alumno
Servidor-The recv() is OK...
Servidor-Received: UPDATE Alumno SET Nombre=Miguel WHERE Id=2
Servidor-The recv() is OK...
Servidor-Received: SELECT * FROM Alumno
```

**Right Window (Client):**

```
andreaagr@Mac-mini-de-Andrea Cliente-Servidor % ./client 192.168.0.141
Client-The remote host is: 192.168.0.141
Client-The socket() sockfd is OK...
Server-Using 192.168.0.141 and port 3490...
Client-The connect() is OK...
Escribe un mensaje a enviar
SELECT * FROM Alumno
Client-The recv() is OK...
Client-Received:
-----
Id,Nombre,Apellido,Semestre,Carrera
-----
1,Andy,Perez,5,Computacion
2,Andy,Lopez,8,Electronica
Escribe un mensaje a enviar
UPDATE Alumno SET Nombre=Miguel WHERE Id=2
Client-The recv() is OK...
Client-Received: [-] Record updated
Escribe un mensaje a enviar
SELECT * FROM Alumno
Client-The recv() is OK...
Client-Received:
-----
Id,Nombre,Apellido,Semestre,Carrera
-----
1,Andy,Perez,5,Computacion
2,Miguel,Lopez,8,Electronica
Escribe un mensaje a enviar
```

Figura 32. Actualización de un registro