



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ingeniería

Ingeniería en Computación

ARQUITECTURAS CLIENTES/SERVIDOR

Grupo 01

Documentación del Proyecto Final

Profesor: ING. CARLOS ALBERTO ROMÁN ZAMITIZ

Salinas Gutiérrez Gerardo

Valeriano Barrios Cristian

Semestre 2021-2

Fecha: 13/08/2021

Inicialmente programamos la base de datos en C para probar su funcionamiento.

Código database.c:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #define MAXDATASIZE 100
5.
6. char entrada[256]; //Mensaje del cliente
7. char numcta[9]; //primary key
8. char apPat[20];
9. char apMat[20];
10. char nombres[40];
11. char buffer[MAXDATASIZE];
12.
13. ///////////////////////////////////////////////////
14. int insert_cmd()
15. {
16.     //Cadena que recibirá el nombre del archivo
17.     char nuevoArchivo[14];
18.     //Variable del archivo
19.     FILE *nuevo;
20.
21.     //El número de cuenta ingresado pasa como el nombre del nuevo archivo
22.     sprintf(nuevoArchivo, "%s.txt", numcta);
23.
24.     //Se abre el archivo en modo escritura
25.     nuevo = fopen(nuevoArchivo, "w"); //FILE * fopen (const char *filename, const
        char *opentype);
26.
27.     //El contenido de la variable "buffer" se escribe en el archivo
28.     fputs (buffer, nuevo);
29.
30.     fclose(nuevo); //Cierre del archivo
31.     printf("\nINSERT EXITOSO\n");
32.     fflush(stdin);
33. }
34.
35. ///////////////////////////////////////////////////
36. int select_cmd()
37. {
38.     char filename[14];
39.     //Función que pasa el nombre de archivo
40.     sprintf(filename, "%s.txt", numcta); //filename=numcta+.txt
41.
42.     //Variable del archivo
43.     FILE *archivo;
44.     char character;
45.
46.     //Se abre el archivo en modo lectura
47.     archivo = fopen(filename, "r");
48.
```

```

49. //Si el archivo no se encuentra
50. if (archivo == NULL){
51.     printf("\nNo existen datos para el num. de cuenta. \n\n");
52. }
53. //Si existe, se lee el archivo encontrado
54. else{
55.     printf("\nEl contenido del archivo es: \n\n");
56.     //Se imprime el contenido del archivo caracter por caracter
57.     while((caracter = fgetc(archivo)) != EOF){
58.         printf("%c",caracter);
59.     }
60. }
61. fclose(archivo); //Cierre del archivo
62. }
63.
64. /*
65. * Función principal main
66. */
67. int main(){
68.
69.     char entcpy[256], //Copia del mensaje para preservar el original
70.     comando[6];      //INSERT o SELECT
71.
72.     int i=0;
73.     char *array[10];
74.
75.     scanf( "%[^\\n]" , entrada); //Lee el mensaje del cliente
76.
77.     char *token = strtok(entrada, " ");
78.     if(token != NULL){
79.         while(token != NULL){
80.             array[i++]=token; //Guardando cada token en un arreglo
81.             token = strtok(NULL, " ");
82.         }
83.     }
84.
85.     //depuracion
86.     for (int j=0; j<i;j++)
87.         printf("Token[%i]: %s\\n",j, array[j]);
88.
89.     /***ASIGNAR LOS TOKENS A LAS VARIABLES GLOBALES***/
90.     strcpy(comando, array[0]);
91.     printf("\\ncomando: %s\\n", comando); //depuracion
92.
93.     strcpy(numcta,array[1]);
94.     printf("num cuenta: %s\\n", numcta); //depuracion
95.
96.     if((i>2)&&(i<9)) //Para evitar segmentation fault
97.     {
98.         strcpy(apPat,array[2]);
99.         printf("ap pat: %s\\n", apPat); //depuracion
100.
101.         strcpy(apMat,array[3]);

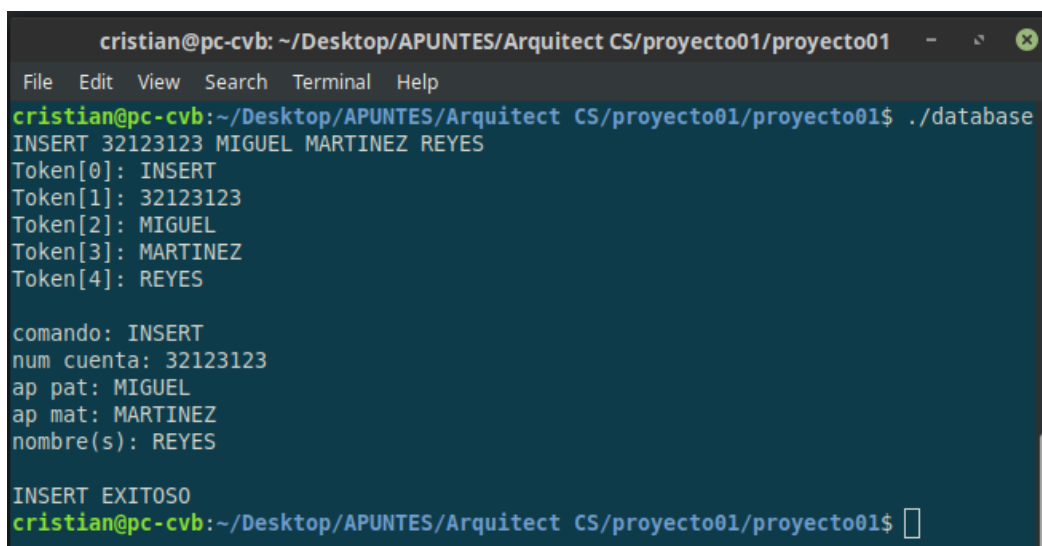
```

```

102.         printf("ap mat: %s\n", apMat); //depuracion
103.
104.         strcpy(nombres,array[4]);
105.         for (int j=5; j<i; j++)
106.         {
107.             strcat(nombres, " ");
108.             strcat(nombres,array[j]);
109.         }
110.         printf("nombre(s): %s\n", nombres); //depuracion
111.     }
112.
113.     //Cadena con los datos finales
114.     sprintf(buffer, "%s %s %s", apPat, apMat, nombres);
115.
116.
117.     // REDIRIGIENDO A FUNCION CORRESPONDIENTE SEGUN EL COMANDO
118.     if(strcmp(comando,"INSERT")==0 || strcmp(comando,"insert")==0) {
119.         insert_cmd();
120.     }
121.     else if(strcmp(comando,"SELECT")==0 || strcmp(comando,"select")==0){
122.         select_cmd();
123.     }
124.     else{
125.         printf("Syntax error\n");
126.         main();
127.     }
128.     return 0;
129. }

```

Ejecución:



The screenshot shows a terminal window titled "cristian@pc-cvb: ~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01". The user has entered the command `./database`. The output shows the command being tokenized into five parts: `INSERT`, `32123123`, `MIGUEL`, `MARTINEZ`, and `REYES`. These are then displayed as `comando: INSERT`, `num cuenta: 32123123`, `ap pat: MIGUEL`, `ap mat: MARTINEZ`, and `nombre(s): REYES`. Finally, the message `INSERT EXITOSO` is printed, indicating a successful insertion.

```

cristian@pc-cvb: ~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01
File Edit View Search Terminal Help
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$ ./database
INSERT 32123123 MIGUEL MARTINEZ REYES
Token[0]: INSERT
Token[1]: 32123123
Token[2]: MIGUEL
Token[3]: MARTINEZ
Token[4]: REYES

comando: INSERT
num cuenta: 32123123
ap pat: MIGUEL
ap mat: MARTINEZ
nombre(s): REYES

INSERT EXITOSO
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$

```

Insersión en la base de datos.

```
cristian@pc-cvb: ~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01
File Edit View Search Terminal Help
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$ cc databas
e.c -o database
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$ ./database
SELECT 32123123
Token[0]: SELECT
Token[1]: 32123123

comando: SELECT
num cuenta: 32123123

El contenido del archivo es:

MIGUEL MARTINEZ REYES
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$
```

Consulta a la base de datos.

```
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$ ./database
SELECT 300000
Token[0]: SELECT
Token[1]: 300000

comando: SELECT
num cuenta: 300000

No existen datos para el num. de cuenta.
```

Consulta a la base de datos de registro no existente.

```
cristian@pc-cvb:~/Desktop/APUNTES/Arquitect CS/proyecto01/proyecto01$ ./database
UPDATE 32123123
Token[0]: UPDATE
Token[1]: 32123123

comando: UPDATE
num cuenta: 32123123
Syntax error
```

Comando SQL diferente a los especificados.

Posteriormente, reutilizamos las funciones de la base de datos, ya probadas, dentro del servidor, para que éste se encargue de gestionarla.

Código serverstream.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <errno.h>
5. #include <string.h>
6. #include <sys/types.h>
7. #include <sys/socket.h>
8. #include <netinet/in.h>
9. #include <arpa/inet.h>
10. #include <sys/wait.h>
11. #include <signal.h>
12.
13. #define MAXDATASIZE 300
14. /* the port users will be connecting to */
15. #define MYPORT 3490
16. /* how many pending connections queue will hold */
17. #define BACKLOG 10
18.
19. /*
20.  * Variables de la base de datos
21.  */
22. char numcta[8]; //primary key
23. char apPat[20];
24. char apMat[20];
25. char nombres[40];
26. char *array[10];
27. char comando[6]; //INSERT o SELECT
28. char buffer[100];
29. char contenido[80];
30. char mensajeFinal[MAXDATASIZE];
31. char nuevoArchivo[14]; //Cadena que recibirá el nombre del archivo
32. char filename[13]; //SELECT
33. FILE *nuevo;
34. int i;
35.
36. /*
37.  * Funciones de la base de datos
38.  */
39. int insert_cmd()
40. {
41.     //El número de cuenta ingresado pasa como el nombre del nuevo archivo
42.     //Se abre el archivo en modo escritura
43.     nuevo = fopen(filename, "w"); //FILE * fopen (const char *filename, const
char *opentype);
44.
45.     //El contenido de la variable "buffer" se escribe en el archivo
46.     fputs (buffer, nuevo);
47.
```

```

48.         fclose(nuevo); //Cierre del archivo
49.         sprintf(mensajeFinal, "INSERT EXITOSO, el archivo %s fue creado\n",
    filename);
50.         //printf("%s", mensajeInsert);
51.     }
52.
53. int select_cmd()
54. {
55.         nuevo = fopen(filename,"r"); //Se abre el archivo en modo lectura
56.
57.         //Si el archivo no se encuentra
58.         if (nuevo == NULL)
59.             sprintf(mensajeFinal, "No existen datos para el num. de cuenta %s\n",
    numcta);
60.         //Si existe, se lee el archivo encontrado
61.         else{
62.             while (feof(nuevo) == 0){
63.                 fgets(contenido, 80, nuevo);
64.             }
65.             sprintf(mensajeFinal, "El contenido del archivo %s es:\n%s", filename,
    contenido);
66.         }
67.         fclose(nuevo); //Cierre del archivo
68.     }
69.
70. void sigchld_handler(int s){
71.     while(wait(NULL) > 0);
72. }
73.
74. int main(int argc, char *argv[ ])
75. {
76.     /* listen on sock_fd, new connection on new_fd
77.     * sockfd es el file descriptor 1, propio del servidor
78.     * new_fd es el file descriptor 2, tiene la información propia del cliente
79.     */
80.     int sockfd, new_fd;
81.     char entrada[MAXDATASIZE];
82.     int numbytes;
83.
84.     /* my address information */
85.     struct sockaddr_in my_addr;
86.
87.     /* connectors address information */
88.     struct sockaddr_in their_addr;
89.     int sin_size;
90.     struct sigaction sa;
91.     int yes = 1;
92.
93.     // sockfd se inicia con la llamada de la función socket
94.     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
95.         perror("Server-socket() error lol!");
96.         exit(1);
97.     }

```

```

98.     else
99.         printf("Server-socket() sockfd is OK...\n");
100.
101.         if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) == -1){
102.             perror("Server-setsockopt() error lol!");
103.             exit(1);
104.         }
105.         else
106.             printf("Server-setsockopt is OK...\n");
107.
108.         /* host byte order */
109.         my_addr.sin_family = AF_INET;
110.
111.         /* short, network byte order */
112.         my_addr.sin_port = htons(MYPORT);
113.
114.         /* automatically fill with my IP */
115.         my_addr.sin_addr.s_addr = INADDR_ANY;
116.         printf("Server-Using %s and port %d...\n", inet_ntoa(my_addr.sin_addr),
MYPORT);
117.
118.         /* zero the rest of the struct */memset(&(my_addr.sin_zero), '\0', 8);
119.         if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) ==
-1){
120.             perror("Server-bind() error");
121.             exit(1);
122.         }
123.         else
124.             printf("Server-bind() is OK...\n");
125.
126.         if(listen(sockfd, BACKLOG) == -1){
127.             perror("Server-listen() error");
128.             exit(1);
129.         }
130.
131.         printf("Server-listen() is OK...Listening...\n");
132.         /* clean all the dead processes */
133.         sa.sa_handler = sigchld_handler;
134.         sigemptyset(&sa.sa_mask);
135.         sa.sa_flags = SA_RESTART;
136.
137.         // sigaction elimina los procesos que puedan quedar
138.         if(sigaction(SIGCHLD, &sa, NULL) == -1){
139.             perror("Server-sigaction() error");
140.             exit(1);
141.         }
142.         else
143.             printf("Server-sigaction() is OK...\n");
144.
145.         // servidor entra en un ciclo infinito, en él se encuentra un accept()
146.         while(1){
147.             sin_size = sizeof(struct sockaddr_in);

```



```

148.         // cuando un cliente se conecta, es aceptado por accept() y se
        inicializa new_fd
149.         if((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size))
        == -1){
150.             perror("Server-accept() error");
151.             continue;
152.         }
153.         else
154.             printf("Server-accept() is OK...\n");
155.
156.             printf("Server-new socket, new_fd is OK...\n");
157.             printf("Server: Got connection from %s\n",
        inet_ntoa(their_addr.sin_addr));
158.
159.         // se conecta el cliente y el servidor crea un hijo con fork(), el hijo
        entra al if
160.         if(!fork())
161.         {
162.             // cierra el sockfd, no lo necesita.
163.             close(sockfd);
164.             // hijo se comunica con el cliente, envía mensaje (37 bytes) a
        través del new_fd
165.             if(send(new_fd, "Conectado al servidor exitosamente\n", 37, 0) ==
        -1)
166.                 perror("Server-send() error lol!");
167.             // padre entra al else, regresa al while a aceptar la conexión de
        otro cliente
168.         else
169.             printf("Server-send is OK...!\n");
170.
171.             printf("Conectado con el cliente exitosamente\n");
172.
173.             ////PETICION DEL CLIENTE
174.             printf("\nEsperando el mensaje del cliente:\n");
175.             if((numbytes = recv(new_fd, entrada, MAXDATASIZE-1, 0)) == -1){
176.                 perror("recv()");
177.                 exit(1);
178.             }
179.             else{
180.                 printf("Client-The recv() is OK...\n");
181.                 entrada[numbytes] = '\0';
182.                 printf("Server-Received: %s\n", entrada);
183.             }
184.             ////FIN DE PETICION
185.
186.             ////ANALISIS DEL MENSAJE
187.             entrada[strcspn(entrada, "\n")] = 0; // encuentra un salto de linea
        \n y lo elimina
188.             char *token = strtok(entrada, " ");
189.             if(token != NULL){
190.                 while(token != NULL){
191.                     array[i++]=token; //Guardando cada token en un arreglo
192.                     token = strtok(NULL, " ");

```

```

193.         }
194.     }
195.
196.     for (int j=0; j<i;j++)
197.         printf("Token[%i]: %s\n",j, array[j]);
198.
199.     strcpy(comando, array[0]);
200.     printf("\ncomando: %s\n", comando);
201.
202.     strcpy(numcta,array[1]);
203.     printf("num cuenta: %s\n", numcta);
204.
205.     if((i>2)&&(i<9)) //Para evitar segmentation fault
206.     {
207.         strcpy(apPat,array[2]);
208.         printf("ap pat: %s\n", apPat);
209.         strcpy(apMat,array[3]);
210.         printf("ap mat: %s\n", apMat);
211.         strcpy(nombres,array[4]);
212.         for (int j=5; j<i; j++){
213.             strcat(nombres, " ");
214.             strcat(nombres,array[j]);
215.         }
216.         printf("nombre(s): %s\n", nombres);
217.     }
218.     sprintf(buffer, "%s %s %s", apPat, apMat, nombres); // cadena con el
nombre completo
219.     sprintf(filename, "%s.txt", numcta); // cadena con el nombre de
archivo
220.
221.     // REDIRIGIENDO A FUNCION CORRESPONDIENTE SEGUN EL COMANDO
222.     if(strcmp(comando,"INSERT")==0) {
223.         insert_cmd();
224.     }
225.     else if(strcmp(comando,"SELECT")==0){
226.         select_cmd();
227.     }
228.     else{
229.         printf("Syntax error\n");
230.         exit(1);
231.     }
232.     ///FIN DEL ANALISIS
233.
234.     ///ENVIO DE RESULTADO
235.     if(send(new_fd, mensajeFinal, 80, 0) == -1)
236.         perror("Server-send() error lol!");
237.     else
238.         printf("\nRESULTADO ENVIADO\n");
239.     ///FIN DEL ENVIO
240.
241.     close(new_fd); // cierra el descriptor de archivo
242.     printf("\nServer-new socket, new_fd closed successfully...\n");
243.     printf("\nEsperando nueva conexion...\n");

```

```

244.             exit(0);    // termina
245.
246.         }
247.         /* parent doesnt need this */
248.         // el padre no se va a comunicar con el cliente, cierra el new_fd
249.         close(new_fd);
250.         printf("\nServer-new socket, new_fd closed successfully...\n");
251.         fflush(stdin);
252.     }
253.     return 0;
254. }

```

Código clientstream.c

```

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <errno.h>
5. #include <string.h>
6. #include <netdb.h>
7. #include <sys/types.h>
8. #include <netinet/in.h>
9. #include <sys/socket.h>
10.
11. // puerto 3490 aleatorio (se pueden elegir desde 1024 en adelante)
12. #define PORT 3490
13. // max number of bytes we can get at once
14. #define MAXDATASIZE 300
15.
16. int main(int argc, char *argv[])
17. {
18.     int sockfd, numbytes;
19.     char str[100];
20.     char buf[MAXDATASIZE];
21.     struct hostent *he;    // estructura "hostent", apuntador "he"
22.
23.     // connectors address information
24.     struct sockaddr_in their_addr;
25.     // if no command line argument supplied
26.     if(argc != 2){
27.         fprintf(stderr, "Client-Usage: %s host_servidor\n", argv[0]);
28.         // just exit
29.         exit(1);
30.     }
31.

```

```

32.     // si lo que devuelve gethostbyname es NULL, entra al if
33.     if((he=gethostbyname(argv[1])) == NULL){
34.         // argv[1] es el argumento desde linea de comandos (dominio)
35.         // se le asigna el valor de gesthostbyname a la variable "he", apuntador a
    estructura hostend
36.         // asignado el valor, se compara contra NULL, si es igual, entra al if
37.         perror("gethostbyname()");
38.         exit(1); // termina la ejecución con exit(1)
39.     }
40.     // si no es NULL, entra al else
41.     else
42.         printf("Client-The remote host is: %s\n", argv[1]);
43.
44.     if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
45.         perror("socket()");
46.         exit(1);
47.     }
48.     else
49.         printf("Client-The socket() sockfd is OK...\n");
50.
51.     // host byte order
52.     their_addr.sin_family = AF_INET;
53.     // short, network byte order
54.     printf("Server-Using %s and port %d...\n", argv[1], PORT);
55.     their_addr.sin_port = htons(PORT);
56.     their_addr.sin_addr = *((struct in_addr *)he->h_addr);
57.
58.     // zero the rest of the struct
59.     memset(&(their_addr.sin_zero), '\0', 8);
60.
61.
62.     // si la conexión a través del sockfd es fallida
63.     if(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) ==
-1){
64.         perror("connect()");
65.         exit(1);
66.     }
67.     // si la conexión a través del sockfd es exitosa
68.     else{
69.
70.         printf("Client-The connect() is OK...\n");
71.
72.         ////SALUDO SERVER
73.         if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1){
74.             perror("recv()");
75.             exit(1);
76.         }
77.         else
78.             printf("Client-The recv() is OK...\n");
79.
80.         buf[numbytes] = '\0';
81.         printf("Client-Received: %s\n", buf);
82.         ////FIN SALUDO

```

```

83.
84.     ///ENVIO DE PETICION
85.     printf("> ");
86.     fgets(str, 100, stdin);
87.     send(sockfd, str, strlen(str), 0);
88.     ///FIN DEL ENVIO DE PETICION
89.
90.     ///MENSAJE RESULTADO
91.     if((numbytes = recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1){
92.         perror("recv()");
93.         exit(1);
94.     }
95.     else
96.         printf("Client-The recv() is OK...\n");
97.
98.     buf[numbytes] = '\0';
99.     printf("\nClient-Received:\n%s\n", buf);
100.    ///FIN DEL MENSAJE
101.    }
102.    printf("\nClient-Closing sockfd\nDESCONECTADO DEL SERVIDOR!\n");
103.    close(sockfd);
104.    return 0;
105. }

```

Ejecución:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$ ./clientstream localhost
Client-The remote host is: localhost
Client-The socket() sockfd is OK...
Server-Using localhost and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: Conectado al servidor exitosamente
> 
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$ ./serverstream
Server-socket() sockfd is OK...
Server-setsockopt() is OK...
Server-Using 0.0.0.0 and port 3490...
Server-bind() is OK...
Server-listen() is OK...Listening...
Server-select() is OK...
Server-accept() is OK...
Server-new socket, new_fd is OK...
Server: Got connection from 127.0.0.1

Server-new socket, new_fd closed successfully...
Server-send is OK...!
Conectado con el cliente exitosamente

Esperando el mensaje del cliente:

```

Conexión cliente-servidor.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$ ./clientstream localhost
Client-The remote host is: localhost
Client-The socket() sockfd is OK...
Server-Using localhost and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: Conectado al servidor exitosamente

> INSERT 312227960 SALINAS GUTIERREZ GERARDO
Client-The recv() is OK...

Client-Received:
INSERT EXITOSO, el archivo 312227960.txt fue creado

Client-Closing sockfd
DESCONECTADO DEL SERVIDOR!
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$
```

```
Esperando el mensaje del cliente:
Client-The recv() is OK...
Server-Received: INSERT 312227960 SALINAS GUTIERREZ GERARDO

Token[0]: INSERT
Token[1]: 312227960
Token[2]: SALINAS
Token[3]: GUTIERREZ
Token[4]: GERARDO

comando: INSERT
num cuenta: 312227960
ap pat: SALINAS
ap mat: GUTIERREZ
nombre(s): GERARDO

RESULTADO ENVIADO

Server-new socket, new_fd closed successfully...

Esperando nueva conexion...
```

*INSERT correcto realizado por ejecución del servidor.
El cliente termina y el servidor sigue a la espera de la llegada de una nueva petición.*

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Client-Closing sockfd
DESCONECTADO DEL SERVIDOR!
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$ ./clientstream localhost
Client-The remote host is: localhost
Client-The socket() sockfd is OK...
Server-Using localhost and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: Conectado al servidor exitosamente

> INSERT 123456789 VALERIANO BARRIOS CRISTIAN
Client-The recv() is OK...

Client-Received:
INSERT EXITOSO, el archivo 123456789.txt fue creado

Client-Closing sockfd
DESCONECTADO DEL SERVIDOR!
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$
```

```
Esperando el mensaje del cliente:
Client-The recv() is OK...
Server-Received: INSERT 123456789 VALERIANO BARRIOS CRISTIAN

Token[0]: INSERT
Token[1]: 123456789
Token[2]: VALERIANO
Token[3]: BARRIOS
Token[4]: CRISTIAN

comando: INSERT
num cuenta: 123456789
ap pat: VALERIANO
ap mat: BARRIOS
nombre(s): CRISTIAN

RESULTADO ENVIADO

Server-new socket, new_fd closed successfully...

Esperando nueva conexion...
```

Se realiza un segundo INSERT sin problema.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Client-Closing sockfd
DESCONECTADO DEL SERVIDOR!
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$ ./clientstream localhost
Client-The remote host is: localhost
Client-The socket() sockfd is OK...
Server-Using localhost and port 3490...
Client-The connect() is OK...
Client-The recv() is OK...
Client-Received: Conectado al servidor exitosamente

> INSERT 123456789 VALERIANO BARRIOS CRISTIAN
Client-The recv() is OK...

Client-Received:
INSERT EXITOSO, el archivo 123456789.txt fue creado

Client-Closing sockfd
DESCONECTADO DEL SERVIDOR!
gerardo@pc-gsg:~/Documents/github folder/proyecto-ACS/proyecto01$

Esperando el mensaje del cliente:
Client-The recv() is OK...
Server-Received: INSERT 123456789 VALERIANO BARRIOS CRISTIAN

Token[0]: INSERT
Token[1]: 123456789
Token[2]: VALERIANO
Token[3]: BARRIOS
Token[4]: CRISTIAN

comando: INSERT
num cuenta: 123456789
ap pat: VALERIANO
ap mat: BARRIOS
nombre(s): CRISTIAN

RESULTADO ENVIADO

Server-new socket, new_fd closed successfully...

Esperando nueva conexion...
```

*SELECT correcto realizado por ejecución del servidor.
El cliente termina y el servidor sigue a la espera de la llegada de una nueva petición.*