

SDEP: SERVIDOR WEB

ALUMNOS:

BASTIDA PRADO JAIME ARMANDO

ORTÍZ RODRÍGUEZ SALVADOR ALEJANDRO

SÁNCHEZ CUEVAS ESTEBAN

PROFESOR: UKRANIO CORONILLA CONTRERAS

GRUPO: 4CM4

Mayo 2020

Índice

1. INTRODUCCIÓN	3
2. DESARROLLO DE LA PRÁCTICA	3
2.1. Probando UDP Broadcast	3
3. EJERCICIO 1	5
3.1. Desarrollo	5
3.2. Pruebas	10
3.2.1. Prueba con 1000	10
3.2.2. Prueba con 5000	15
4. CONCLUSIONES	19

1. INTRODUCCIÓN

En esta práctica vamos a agregar una interfaz web para que el administrador pueda consultar en “tiempo real” las estadísticas de los votos, así como el estado de los tres servidores de la práctica pasada.

Para determinar el estado de los tres servidores y saber cuáles se encuentran activos vamos a utilizar el mensaje UDP de broadcast.

Utilizando los mensajes de broadcast descubriremos los servicios en una subred de una manera sencilla y eficiente.

2. DESARROLLO DE LA PRÁCTICA

2.1. Probando UDP Broadcast

Implementamos la función “setBroadcast()” en la clase “SocketDatagrama” para poder hacer el envío de mensajes broadcast. Procedimos entonces a probar nuestro nuevo método imprimiendo en consola una lista de las IP’s correspondientes a los servidores activos (servidores del capítulo 9 del manual de “Programación de Sistemas Linux”), junto con las respuestas que estos mandan que es simplemente la suma de los dos enteros enviados por broadcast. Utilizamos hamachi así que la IP a la que mandamos los mensajes fue a la de la VPN en este caso la “25.255.255.255”

Salida del cliente:

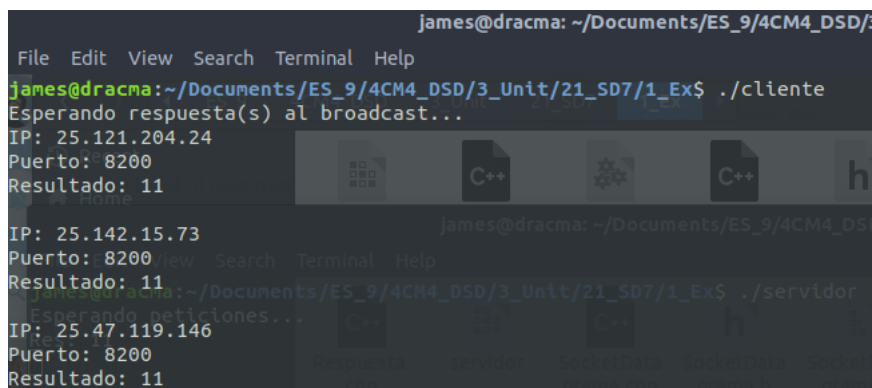


Figura 1: Cliente

Servidor 1:

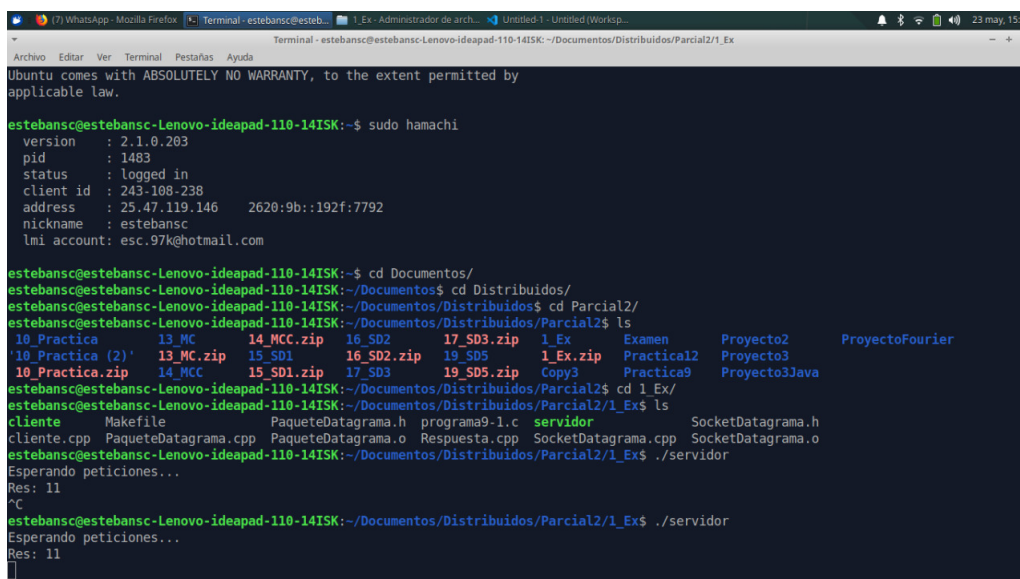
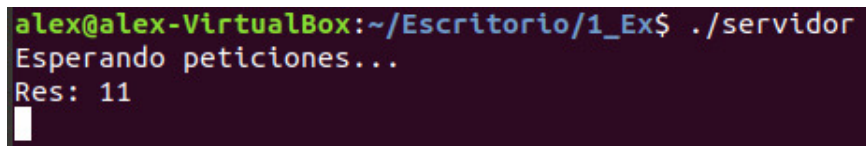


Figura 2: Servidor 1

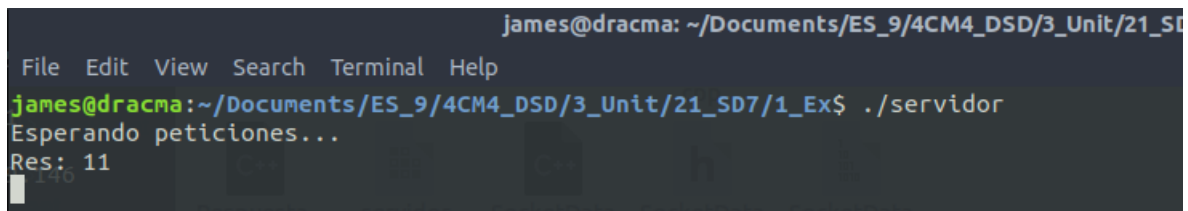
Servidor 2:

A terminal window with a dark purple background. The prompt is 'alex@alex-VirtualBox:~/Escritorio/1_Ex\$'. The command './servidor' has been executed, resulting in the output 'Esperando peticiones...' followed by 'Res: 11' on the next line. A white cursor is visible at the end of the second line.

```
alex@alex-VirtualBox:~/Escritorio/1_Ex$ ./servidor
Esperando peticiones...
Res: 11
```

Figura 3: Servidor 2

Servidor 3:

A terminal window with a dark blue background and a menu bar at the top containing 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The prompt is 'james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ex\$'. The command './servidor' has been executed, resulting in the output 'Esperando peticiones...' followed by 'Res: 11' on the next line. A white cursor is visible at the end of the second line.

```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ex$ ./servidor
Esperando peticiones...
Res: 11
```

Figura 4: Servidor 3

3. EJERCICIO 1

Ahora, ya con todos los conocimientos sobre Mongoose obtenidos por la práctica correspondiente, procedimos a implementar un servidor web embebido en cada uno de los tres servidores de nuestro sistema distribuido, de manera que cuando el administrador ingrese a la página web de cualquiera de los tres servidores, le sea mostrado; el número de votos registrados hasta el momento de los tres servidores de votos, una gráfica con la estadística de votos en "tiempo real", así como las IPs de los servidores activos.

La forma en la que lo realizamos fue la siguiente, construyendo sobre la última implementación (práctica de sincronización), codificamos el servidor web utilizando Mongoose embebido en el código que ya teníamos de los servidores de votos, pero lanzando este servidor web en un hilo por cada uno de los servidores para que no interrumpiera con la ejecución normal de los servidores de votos. Estos tres servidores web corren en los puertos 8000, 8001 y 8002, respectivamente, de acuerdo al número de servidor. Además cada servidor de votos corre en los puertos 7200, 7201 y 7202. Por otro lado, cada servidor de votos abre un socket en el puerto 9000 para la comunicación broadcast, pues cuando el administrador ingresa a la página web esta realiza una petición al servidor web correspondiente, para que haga una transmisión de mensaje broadcast hacia los servidores de votos y así obtener su IP, número de servidor y número de votos que lleva registrados.

Para el desarrollo de esta práctica creamos un sitio web utilizando jQuery para hacer uso de Ajax y para hacer la página dinámica, y para la vista utilizamos Materialize para poder crear una página agradable visualmente y responsiva. Además para la grafica con estadística de votos empleamos Chart.js.

3.1. Desarrollo

Dentro del código "servidor.c" tenemos definidas unas variables globales para poder modificar los puertos donde corre cada servidor de votos, cada servidor web y el puerto de comunicación broadcast, si así se desea:

```
33
34  #define SERVIDOR_BROAD_PORT 9000
35
36  #define SERVIDOR_PORT_1 7200
37  #define SERVIDOR_PORT_2 7201
38  #define SERVIDOR_PORT_3 7202
39
```

Figura 5: Puertos de servidores de votos y broadcast

```
59  //
60  static const char *s_http_port_1 = "8000";
61  static const char *s_http_port_2 = "8001";
62  static const char *s_http_port_3 = "8002";
63
```

Figura 6: Puertos de servidores de web

Como habíamos mencionado antes, cada servidor de votos crea un objeto de la clase SocketDatagrama para poder la comunicación broadcast con los servidores web. La forma en la que funciona esto, es que cuando el servidor web manda un mensaje de broadcast para pedir las estadísticas a los servidores de votos que estén corriendo, manda un mensaje con un entero con el número 1, este número es conocido como el código de operación y el 1 le indica al servidor de votos que conteste con su número de servidor, el número de votos que lleva registrados y si seguirá activo o no(explicado más adelante), la IP del servidor de votos va en el paquete enviado al servidor web. Para no perder mucho la rapidez, los servidores de votos solo esperan 1ms (cada iteración) al mensaje broadcast, antes de continuar con su operación normal.

```

264 // ===== PARA COMUNICACIÓN CON E
265 SocketDatagrama socket_broadcast(SERVIDOR_BROAD_PORT); // Cr
266 socket_broadcast.setBroadcast(); // Ha
267 PaqueteDatagrama paquete_recepcion_broad(sizeof(int)); // Cr
268 int info_a_enviar[3]; // In
269 int contador_votos = 0; // Co
270 int codigo_operacion = 0; // Co
271

```

Figura 7: Variables del servidor de votos para la comunicación broadcast con el(los) servidor(es) web

```

327 if(socket_broadcast.recibeTimeout(paquete_recepcion_broad, 0, 1000) > 0)
328 {
329     codigo_operacion = *(int *) paquete_recepcion_broad.obtieneDatos();
330     // printf("Codigo de operacion recibido: %d\n", codigo_operacion);
331     // SI EL CÓDIGO DE OPERACIÓN ES 1, EL SERVIDOR WEB NOS PIDE ENVIAR NUESTRO NÚM
332     if(codigo_operacion == 1)
333     {
334         info_a_enviar[0] = num_server;
335         info_a_enviar[1] = contador_votos;
336         info_a_enviar[2] = 1;
337         PaqueteDatagrama paquete_respuesta((char *) info_a_enviar, sizeof(int) * 3);
338         socket_broadcast.envia(paquete_respuesta);
339     }
340 }

```

Figura 8: Código del servidor de votos que maneja los mensajes broadcast

Cuando uno de los hilos del Cliente, envía un mensaje de "cerrando comunicación" (explicado en la práctica de sincronización), el servidor de votos espera a recibir un próximo mensaje broadcast para avisar al servidor web, con un "-1" el tercer campo de la variable "info_a_enviar", que dejará de estar atendiendo registros y así el servidor web lo marque como en estado "Offline".

```

485 if(socket_broadcast.recibe(paquete_recepcion_broad) > 0)
486 {
487     info_a_enviar[0] = num_server;
488     info_a_enviar[1] = contador_votos;
489     info_a_enviar[2] = -1;
490     PaqueteDatagrama paquete_respuesta((char *) info_a_enviar, sizeof(int) * 3);
491     socket_broadcast.envia(paquete_respuesta);
492 }

```

Figura 9: Código del servidor de votos que maneja el aviso de salida al servidor web

Como mencionamos al principio del ejercicio, cada servidor de votos cuenta con su servidor web embebido, este es implementado en un hilo aparte del hilo principal de ejecución del servidor de votos, para no interrumpir su ejecución normal y lograr así una mayor eficiencia.

```
316
317 // !!!!!!! = > IMPORTANTE <= !!!!!!!
318 // EN ESTE PUNTO SE ECHA A ANDAR EL HILO QUE EJECUTA EL SERVIDOR WEB
319 thread hilo_web(funcion_mgr);
320
321 // EL SERVIDOR DE VOTOS CORRE INDEFINIDAMENTE,
322 while(1)
323 {
```

Figura 10: Código del servidor de votos que crea el hilo del servidor web

Este servidor web corre en el puerto especificado según el número de servidor del que se trate y fuera de eso el código es muy similar al de cualquier servidor HTTP implementado con Mongoose:

```
208 // CREANDO CONEXIÓN DE ESCUCHA Y AÑADIÉNDOLA A LA FUNCIÓN
209 if(num_server == 1)
210     nc = mg_bind(&mgr, s_http_port_1, ev_handler);
211 else if(num_server == 2)
212     nc = mg_bind(&mgr, s_http_port_2, ev_handler);
213 else
214     nc = mg_bind(&mgr, s_http_port_3, ev_handler);
215
```

Figura 11: Código del servidor web que hace bind de acuerdo al número de servidor

El servidor web, cuenta con una función "ev_handler()" que nos permite personalizar como queremos responder a cada petición que llegue al servidor, en este caso tratamos con dos tipos de peticiones, la de servir algún recurso dentro del directorio del servidor y la de devolver las estadísticas de los servidores de votos, el cual es un endpoint RESTful llamado "/statistics":

```
184 case MG_EV_HTTP_REQUEST:
185     // SI SE TRATA DE UN EVENTO HTTP, DEBEMOS CHECAR SI ES
186     // O SI SE ESTÁ PIDIENDO DEVOLVER ALGÚN ARCHIVO DENTRO
187     if(mg_vcmp(&hm->uri, "/statistics") == 0)
188         handle_search_services(nc, hm);
189     else
190         mg_serve_http(nc, (struct http_message *) ev_data,
191
```

Figura 12: Código que maneja las peticiones HTTP

Cuando se trata de ese evento, el handler manda a llamar a la función "handle_search_services()" la cual se encarga de crear un SocketDatagrama habilitando los mensajes broadcast con la función "setBroadcast()". Esta función es la que se encarga de mandar los mensajes de broadcast a los servidores que estén dentro de la red, recibiendo sus respuestas (número de servidor, IP y número de votos), para almacenarlas en formato objeto JSON y enviarlas hacia la página web para que esta pueda mostrar los resultados de manera amigable hacia el administrador.

```

74     SocketDatagrama socket_datagrama(0);           // Abrimos un socket
75     socket_datagrama.setBroadcast();               // Habilitamos el env
76     int código_operacion = 1;                      // Mandamos un 1 por
77     char servidor_json_1[512] = {0};               // Para almacenar en
78     char servidor_json_2[512] = {0};               // Para almacenar en
79     char servidor_json_3[512] = {0};               // Para almacenar en
80     int *info_a_recibir;                           // Para almacenar temp
81     bool servidorRespondio1, servidorRespondio2, servidorRespondio3;
82     int num_servidores_respondieron = 0;           // Para ir contando e
83     bool disminuirNumServidoresAEsperar;           // Cuando un serv

```

Figura 13: Variables de la función que recibe las estadísticas de los servidores de votos

Esta tiene un while que se mantiene iterando mientras las dos siguientes condiciones se cumplan: que el número de servidores a esperar por una respuesta aún no haya respondido y que el timeout de la respuesta al broadcast esperada no haya excedido 5 segundos. Esto lo hacemos por un lado para asegurar que el servidor tenga tiempo de ejecutar sus operaciones de registro y sincronización con los demás servidores, pero también manejamos la variable de "num_servidores_a_esperar" para que cuando un servidor haya notificado que dejará el grupo, el servidor web no tenga que esperar ya 3 respuestas, sino 2 o 1, según el caso, y así no tener que esperar a que se agoten los timeouts para responder a la página web.

```

103     while(num_servidores_respondieron < num_servidores_a_esperar && socket_datag
104     {
105         // printf("-----\n");
106         // printf("IP: %s\n", paquete_recepcion.obtieneDireccion());
107         // printf("Puerto: %d\n", paquete_recepcion.obtienePuerto());
108         info_a_recibir = (int *) paquete_recepcion.obtieneDatos();
109         // printf("Votos que lleva el servidor numero: %d - %d\n", info_a_recibi
110
111         // SI EL SERVIDOR DEVUELVE UN "-1" SIGNIFICA QUE ESTA CERRANDO COMUNICAC
112         if(info_a_recibir[2] == -1)
113             disminuirNumServidoresAEsperar = true;
114
115         // CHECAMOS EL NÚMERO DE SERVIDOR QUE ENVÍO LA RESPUESTA Y DE ACUERDO A
116         if(info_a_recibir[0] == 1)
117         {
118             sprintf(servidor_json_1 + strlen(servidor_json_1), " \"ip\": \"%s\",
119             servidorRespondio1 = true;
120         }
121         else if(info_a_recibir[0] == 2)
122         {
123             sprintf(servidor_json_2 + strlen(servidor_json_2), " \"ip\": \"%s\",
124             servidorRespondio2 = true;
125         }
126         else if(info_a_recibir[0] == 3)
127         {

```

Figura 14: Código que maneja las peticiones HTTP

La página web, por otro lado, luce como se muestra en la imagen de abajo. En la parte superior tenemos una etiqueta donde se muestra el número total de votos registrados, abajo de ello hay tres etiquetas que es donde se muestra cada uno de los servidores; su estado, dirección y número de votos registrados, debajo de ello se encuentra la gráfica que nos permite ver el registro de los votos en el tiempo, en el eje y se muestran el número de votos totales registrados hasta el momento y en el eje x el tiempo (en minutos y segundos), transcurrido desde que el administrador ingresó a la página web.



Figura 15: Muestra de como luce la página web

La forma en la que la página web recibe las estadísticas es por medio de un mensaje que contiene tres objetos JSON cada uno con la información de los 3 servidores de votos correspondientes. Para esto, se programa a la página web para que cada intervalo de medio segundo haga una petición REST al servidor web y este responda con las estadísticas, después, manipulando la vista con Javascript logramos mostrar los resultados en la página:

```

66 // TIMER QUE LLAMA AL ENDPOINT RESTFUL /statistics EN EL SERVIDOR WEB QUE
67 setInterval(function()
68 {
69     // USANDO AJAX ENVIAMOS LA PETICIÓN REST
70     $.ajax(
71     {
72         type: "POST",           // UTILIZANDO EL MÉTODO HTTP POST
73         url: "/statistics",     // ENDPOINT RESTFUL
74         dataType: "json",       // INDICANDO QUE ESPERAMOS RECIBIR DE
75         success: function(respAx) // FUNCIÓN QUE SE EJECUTA AL SER ÉXITO
76     {
77         // MANIPULANDO LA VISTA CON JAVASCRIPT MOSTRAMOS LOS RESULTADOS

```

Figura 16: Código Javascript que hace la petición REST cada intervalo de medio segundo

3.2. Pruebas

Estamos usando la configuración de atención a los registros de la siguiente manera: el servidor 1 recibe los números con terminación 0,1,2,3 a su vez el servidor 2 recibe los que terminan en 4,5,6 y el servidor 3 atiende a 7,8,9.

Además, para realizar las pruebas, corrimos usando hamachi cada servidor en una computadora distinta, para ello se utilizó la dirección de broadcast para la VPN de "25.255.255.255" y se configuró de acuerdo a cada servidor la dirección de los otros, por ejemplo para el servidor 1:

```
20 // ----- VARIABLES G
21 #define SERVIDOR_IP_1 "127.0.0.1"
22 // #define SERVIDOR_IP_1 "192.168.100.99" // Cel
23 // #define SERVIDOR_IP_3 "25.121.204.24" // Jaime
24
25 #define SERVIDOR_IP_2 "25.142.15.73" // Alex
26 // #define SERVIDOR_IP_2 "127.0.0.1"
27 // #define SERVIDOR_IP_2 "25.121.204.24" // Jaime
28
29 #define SERVIDOR_IP_3 "25.47.119.146" // Esteban
30 // #define SERVIDOR_IP_3 "127.0.0.1"
31 // #define SERVIDOR_IP_3 "192.168.100.110" // Compu
32 // #define SERVIDOR_IP_3 "25.121.204.24" // Jaime
33
```

Figura 17: Configuración de IP's del servidor 1

3.2.1. Prueba con 1000

Cliente que manda los registros hacia los servidores de votos:

```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$ ./cliente 1000
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.
3: Ya no hay mas registros por leer. Terminando el programa...
2: Ya no hay mas registros por leer. Terminando el programa...
1: Ya no hay mas registros por leer. Terminando el programa...
```

Figura 18: Cliente

Podemos observar la manera en que se muestran los datos en la página web:

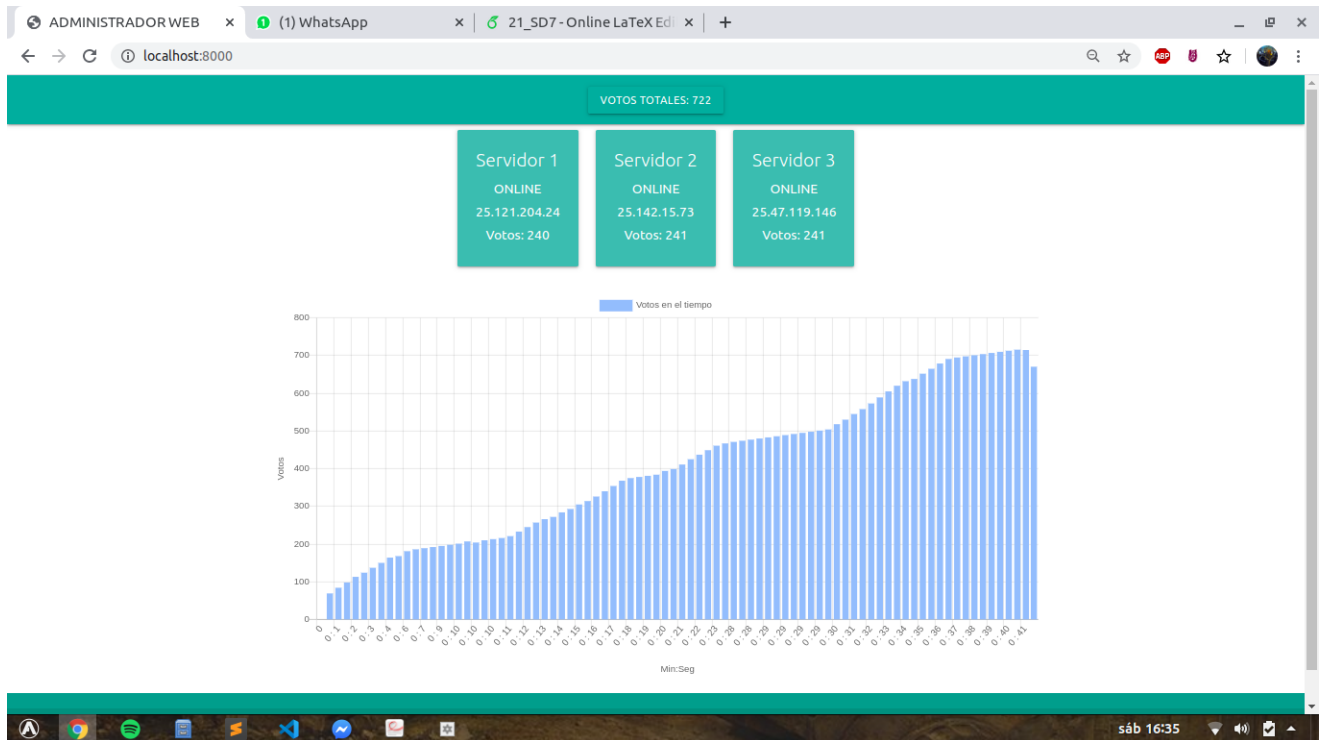


Figura 19: Página web

Vemos que para aproximadamente el momento en que se llega al registro número 4445, el servidor 3 terminó su operación y solo se quedan comunicando entre el servidor 1 y 2, además el servidor web ha recibido su mensaje de salida y lo muestra como Offline:

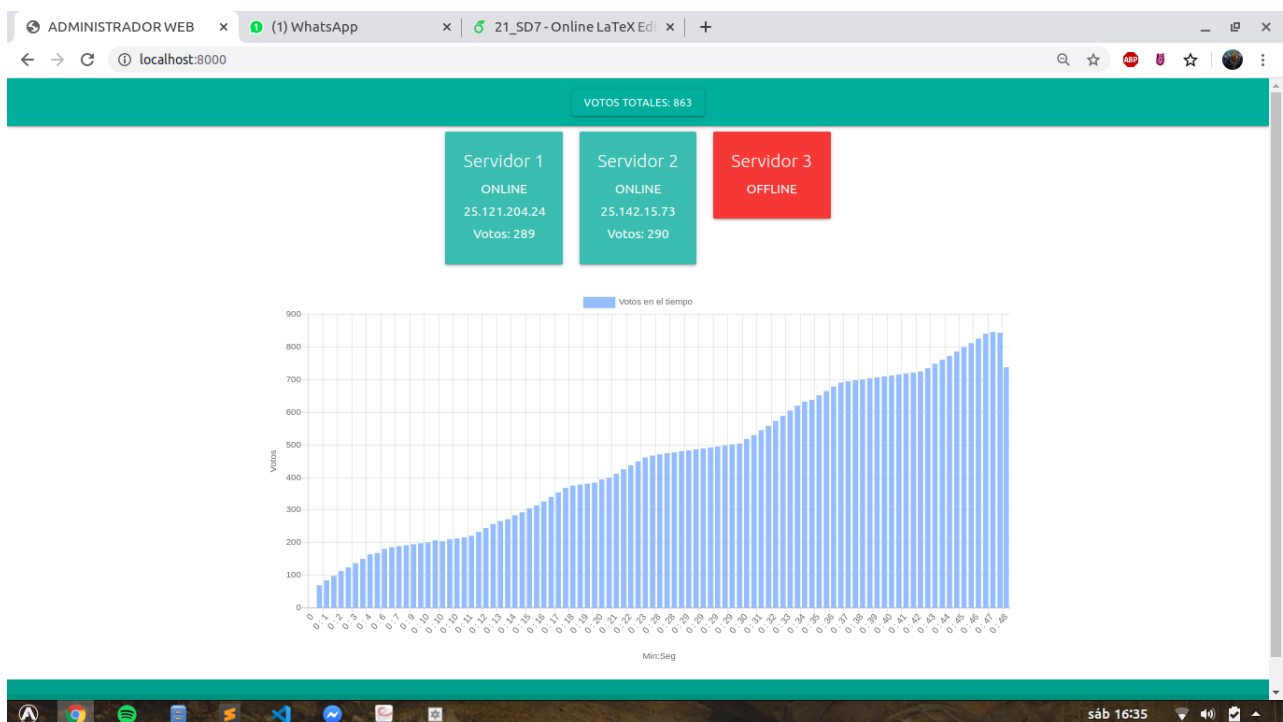


Figura 20: Página web

Después, terminó el servidor 3:

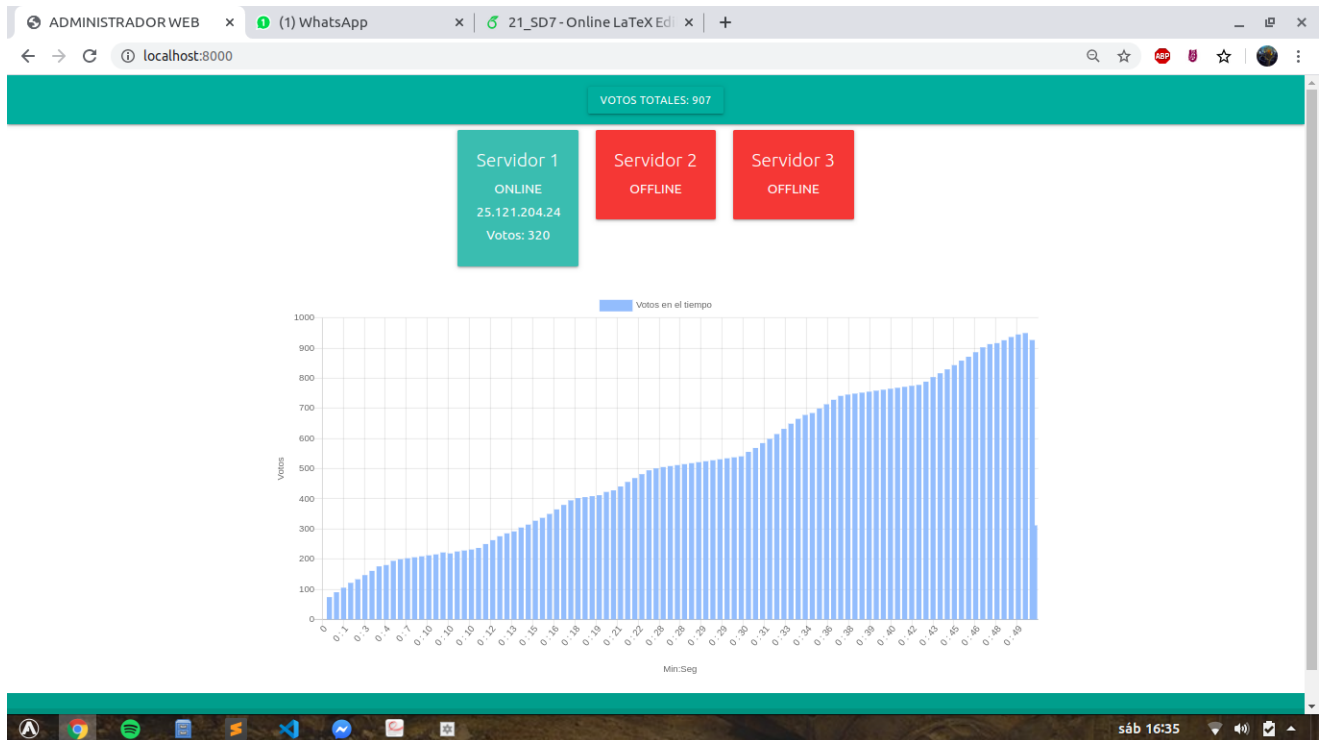


Figura 21: Página web

Y por último el servidor 1, y se atendieron los 1000 registros:

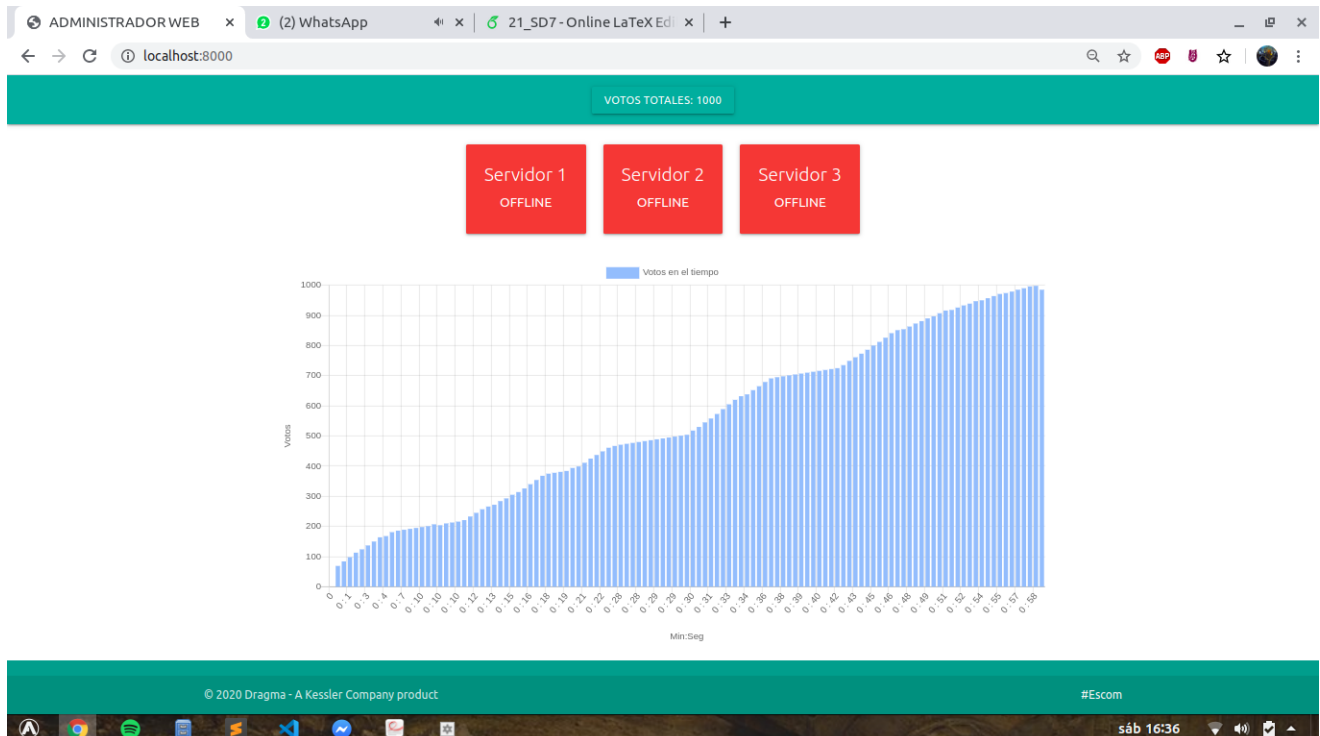
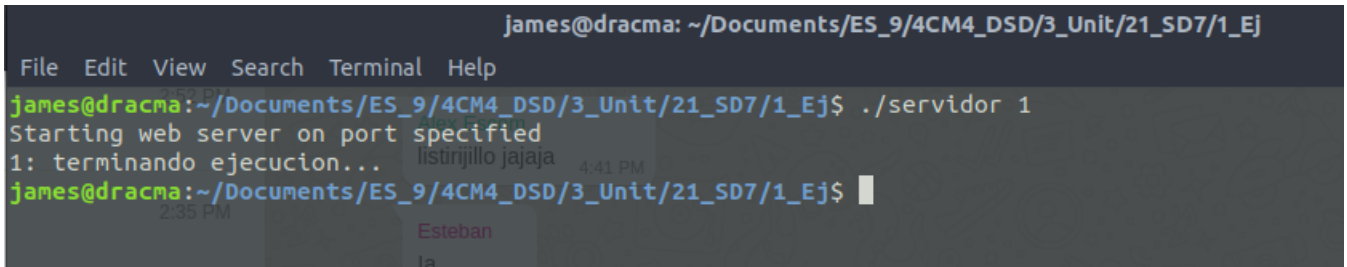


Figura 22: Página web

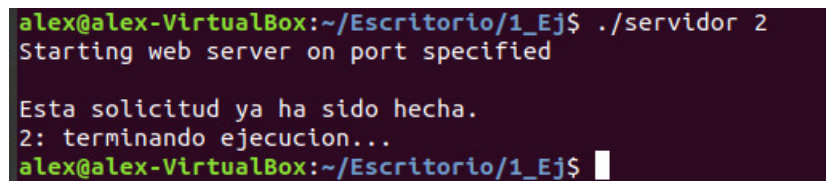
Salida del servidor 1:



```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$ ./servidor 1
Starting web server on port specified
1: terminando ejecucion...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$
```

Figura 23: Servidor 1

Salida del servidor 2:

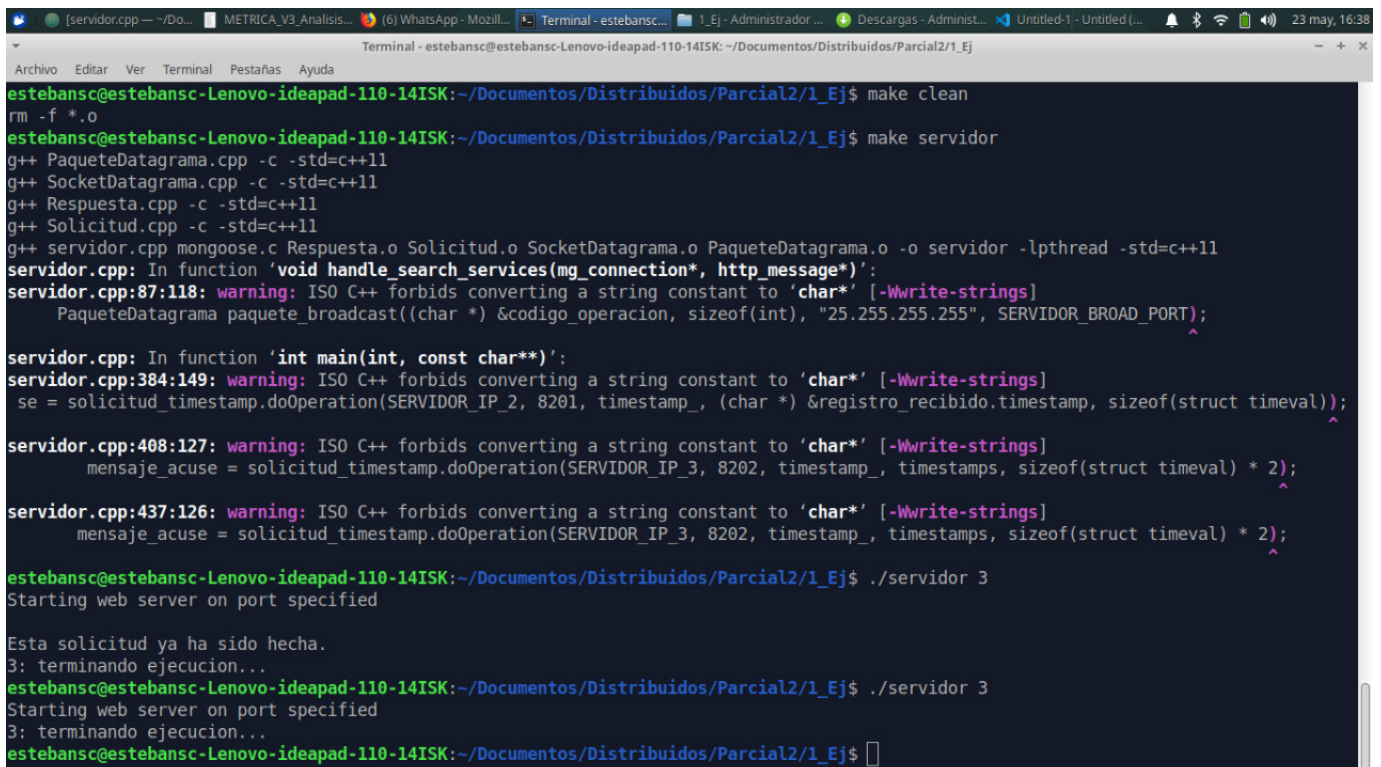


```
alex@alex-VirtualBox:~/Escritorio/1_Ej$ ./servidor 2
Starting web server on port specified

Esta solicitud ya ha sido hecha.
2: terminando ejecucion...
alex@alex-VirtualBox:~/Escritorio/1_Ej$
```

Figura 24: Servidor 2

Salida del servidor 3:



```
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ make clean
rm -f *.o
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ make servidor
g++ PaqueteDatagrama.cpp -c -std=c++11
g++ SocketDatagrama.cpp -c -std=c++11
g++ Respuesta.cpp -c -std=c++11
g++ Solicitud.cpp -c -std=c++11
g++ servidor.cpp mongoose.c Respuesta.o Solicitud.o SocketDatagrama.o PaqueteDatagrama.o -o servidor -lpthread -std=c++11
servidor.cpp: In function 'void handle_search_services(mg_connection*, http_message*)':
servidor.cpp:87:118: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    PaqueteDatagrama paquete_broadcast((char *) &codigo_operacion, sizeof(int), "25.255.255.255", SERVIDOR_BROAD_PORT);
                                                                                                     ^
servidor.cpp: In function 'int main(int, const char**)':
servidor.cpp:384:149: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    se = solicitud_timestamp.doOperation(SERVIDOR_IP_2, 8201, timestamp_, (char *) &registro_recibido.timestamp, sizeof(struct timeval));
                                                                                                     ^
servidor.cpp:408:127: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    mensaje_acuse = solicitud_timestamp.doOperation(SERVIDOR_IP_3, 8202, timestamp_, timestamps, sizeof(struct timeval) * 2);
                                                                                                     ^
servidor.cpp:437:126: warning: ISO C++ forbids converting a string constant to 'char*' [-Wwrite-strings]
    mensaje_acuse = solicitud_timestamp.doOperation(SERVIDOR_IP_3, 8202, timestamp_, timestamps, sizeof(struct timeval) * 2);
                                                                                                     ^
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ ./servidor 3
Starting web server on port specified

Esta solicitud ya ha sido hecha.
3: terminando ejecucion...
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ ./servidor 3
Starting web server on port specified
3: terminando ejecucion...
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$
```

Figura 25: Servidor 3

También se puede acceder a la página de cualquiera de los servidores para checar las estadísticas:

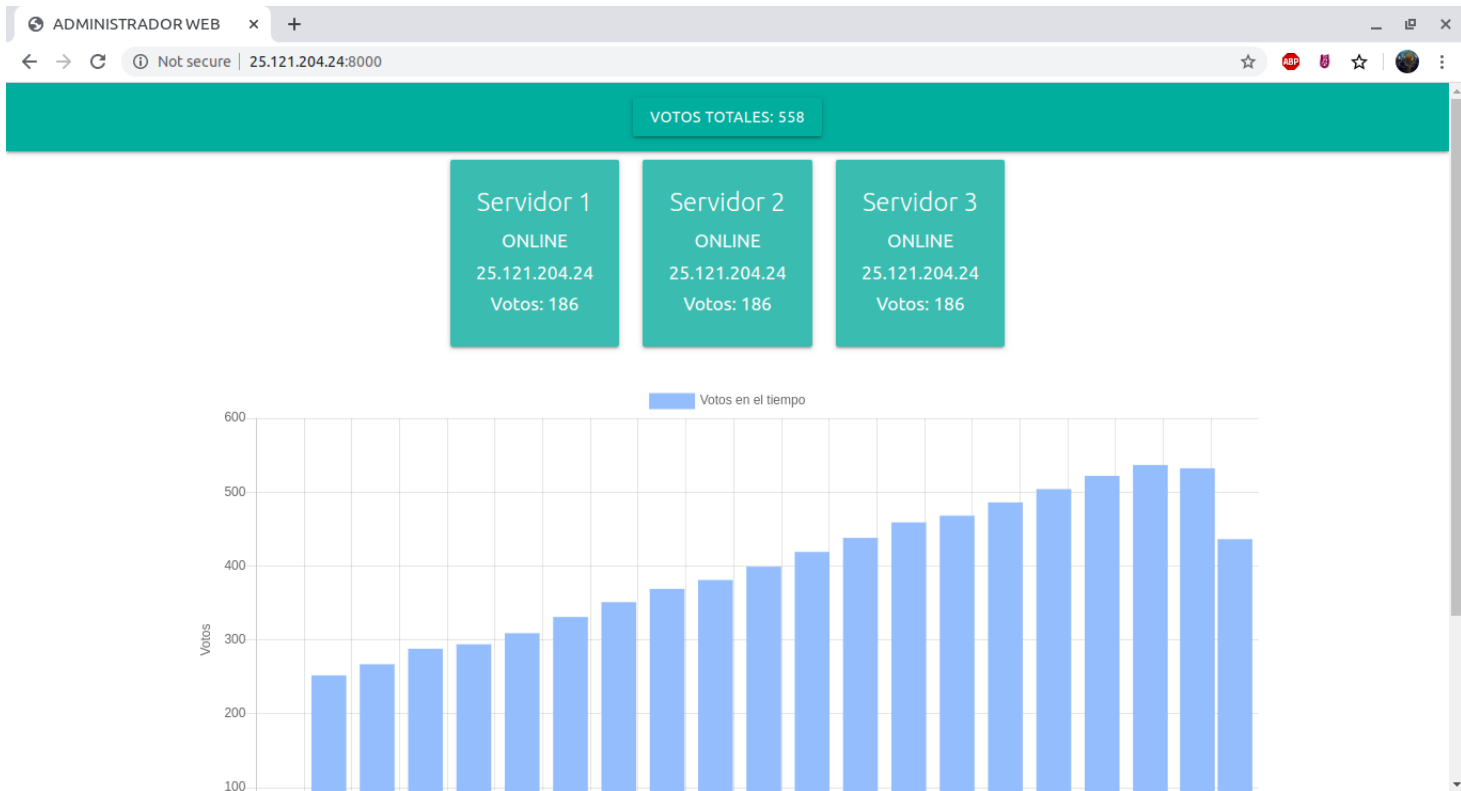


Figura 26: Checando las estadísticas desde otro server

3.2.2. Prueba con 5000

Cliente que manda los registros hacia los servidores de votos:

```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$ ./cliente 5000
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.3Ds
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.004
3: Mensaje del servidor: Registro ya había sido recibido exitosamente.f
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.004
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.3Ds
3: Mensaje del servidor: Registro ya había sido recibido exitosamente.0^
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.'
2: Ya no hay mas registros por leer. Terminando el programa...
3: Ya no hay mas registros por leer. Terminando el programa.png
1: Ya no hay mas registros por leer. Terminando el programa...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$
```

Figura 27: Cliente

Podemos observar la manera en que se muestran los datos en la página web:

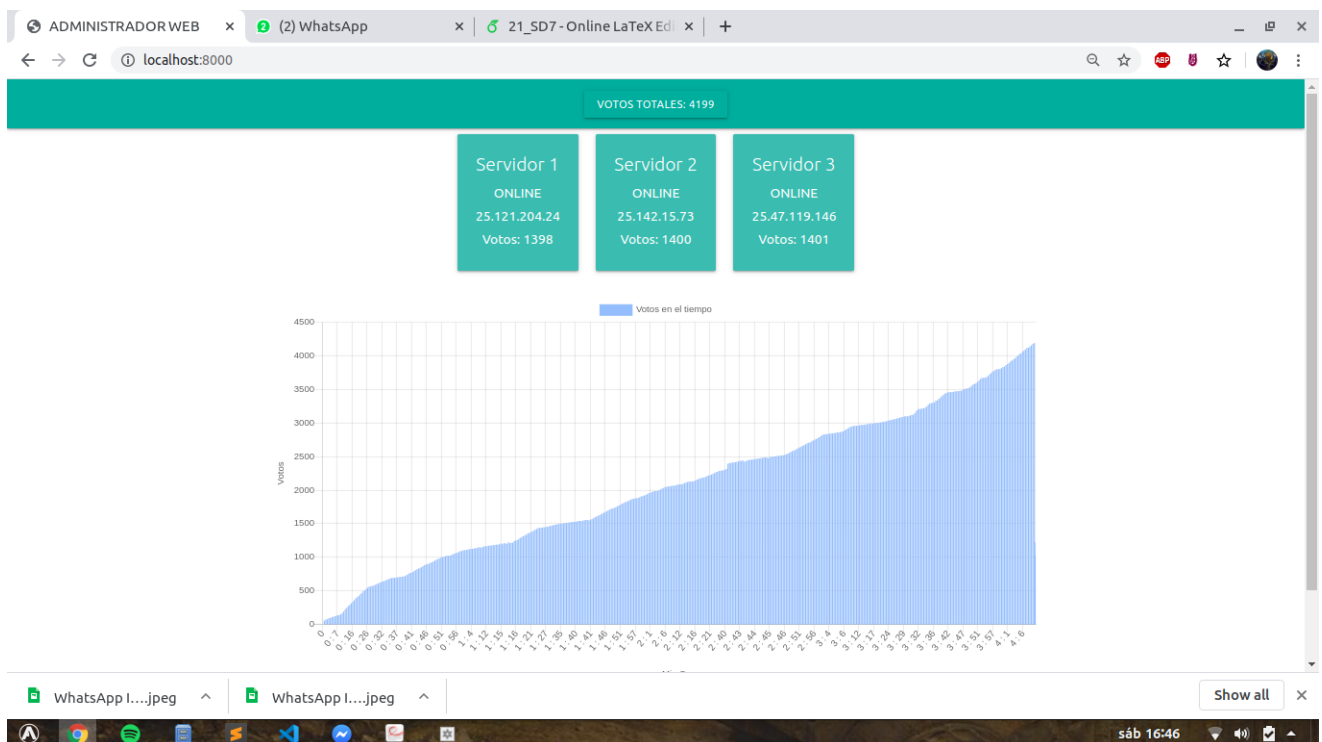


Figura 28: Página web

Vemos que para aproximadamente el momento en que se llega al registro número 4445, el servidor 2 terminó su operación, y solo se quedan comunicando entre el servidor 1 y 3, además el servidor web ha recibido su mensaje de salida y lo muestra como Offline:

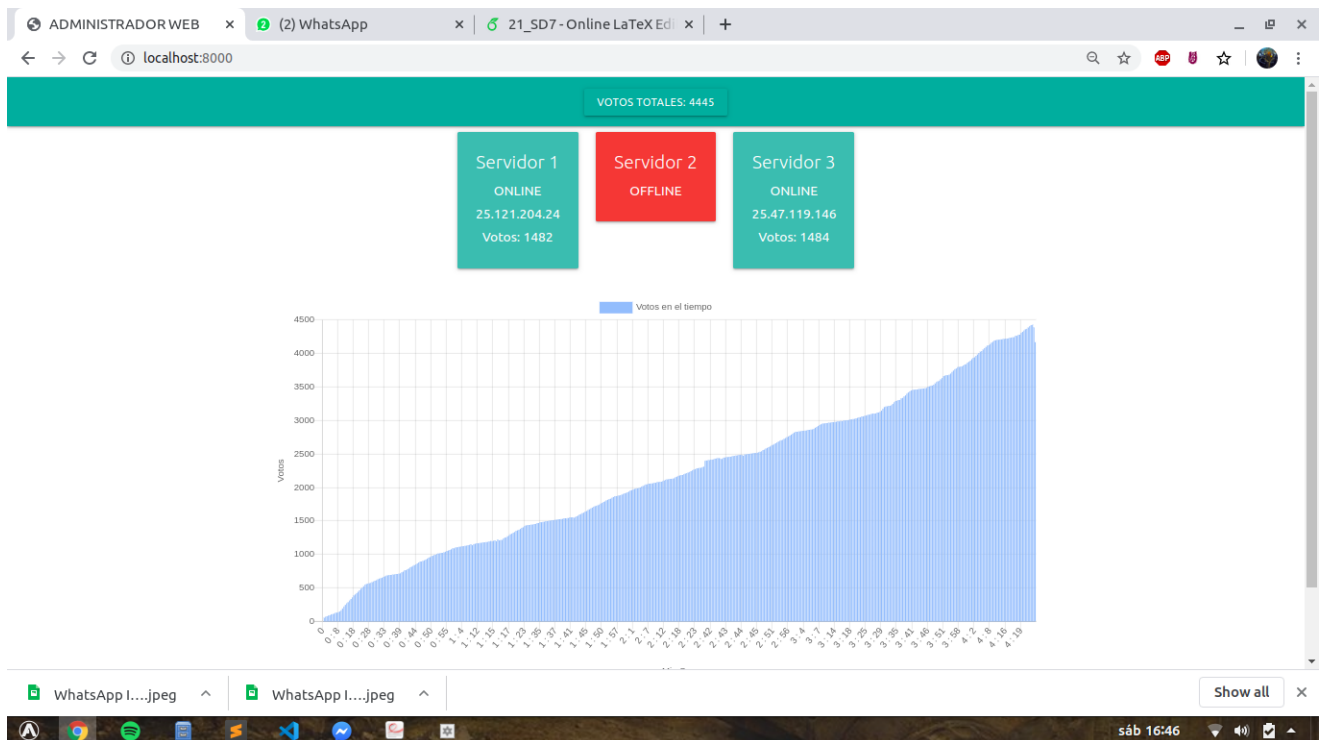


Figura 29: Página web

Después, terminó el servidor 3:

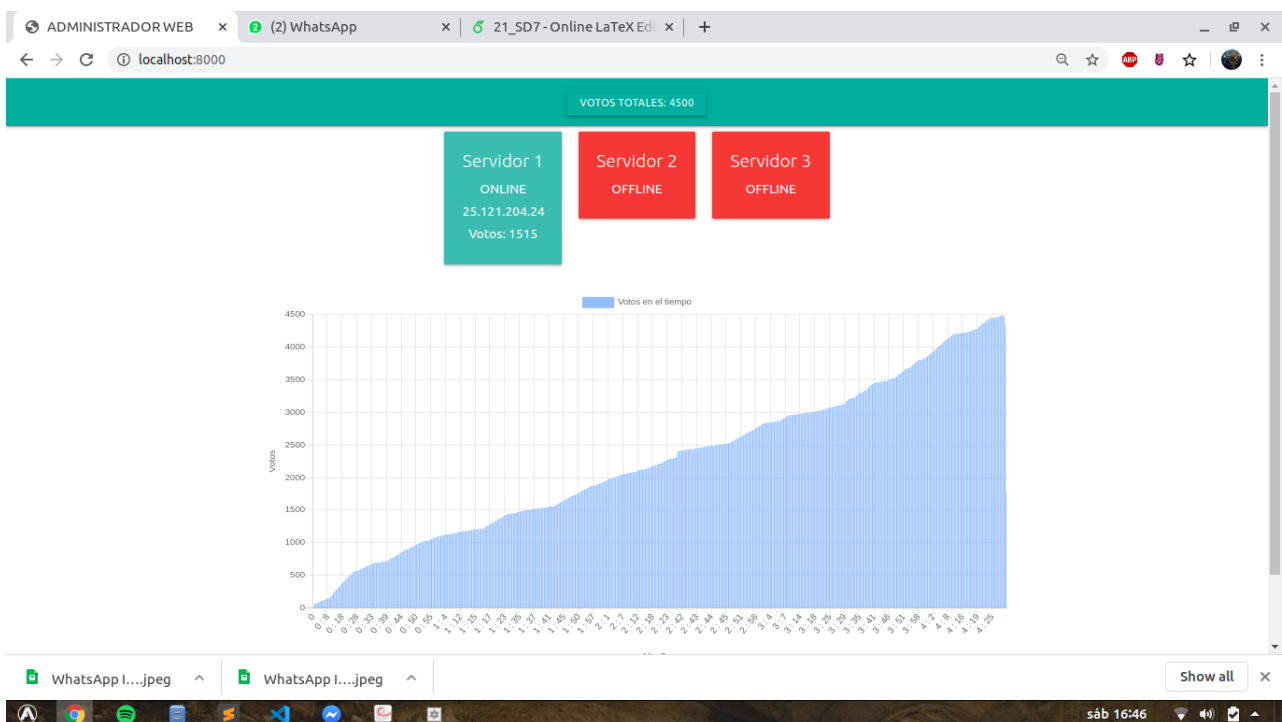


Figura 30: Página web

Y por último el servidor 1, y se atendieron 4998 registros, esto fue porque dos números celulares se repitieron, como veremos a la salida de los servidores 1 y 2:

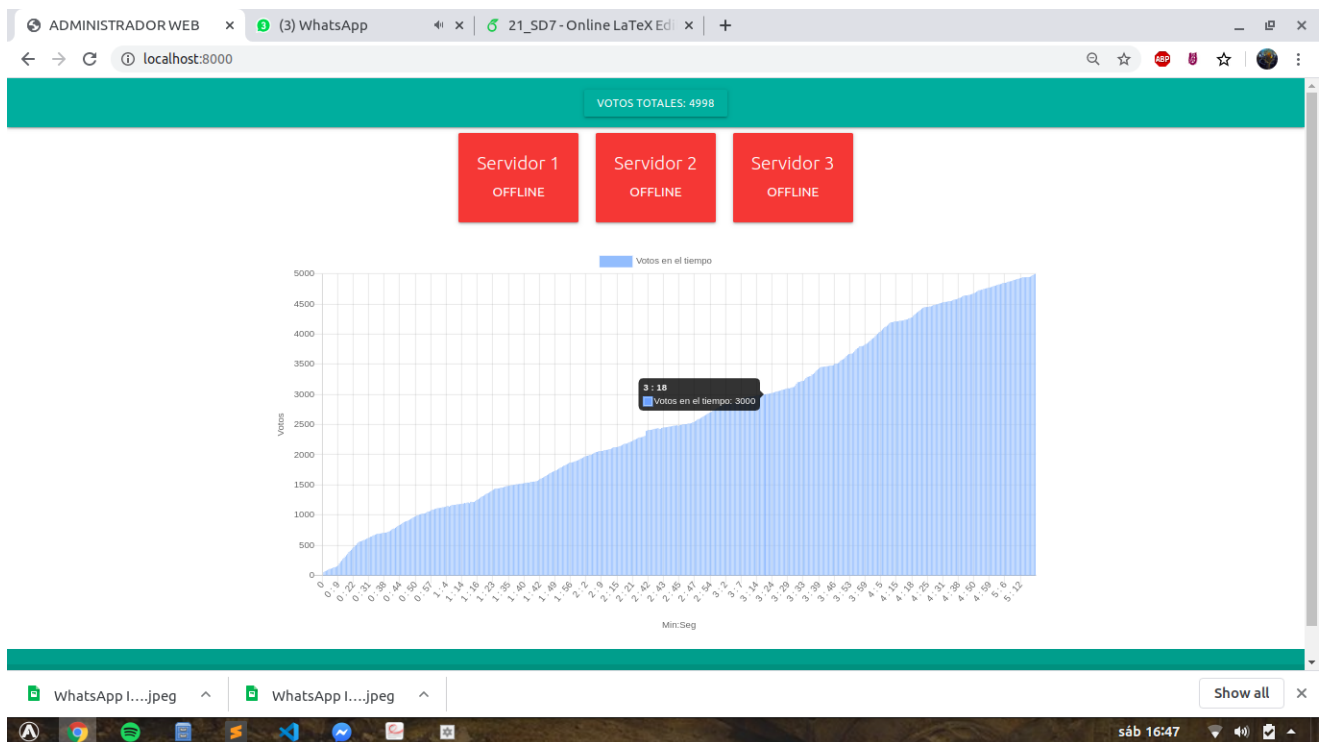


Figura 31: Página web

Salida del servidor 1:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$ ./servidor 1
Starting web server on port specified

Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Ya existia este celular: 5504293162
1: terminando ejecucion...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/1_Ej$
```

Figura 32: Servidor 1

Salida del servidor 2:

```
alex@alex-VirtualBox:~/Escritorio/1_Ej$ ./servidor 2; spd-say done
Starting web server on port specified

Esta solicitud ya ha sido hecha.

Esta solicitud ya ha sido hecha.
Ya existia este celular: 5504293136

Esta solicitud ya ha sido hecha.

Esta solicitud ya ha sido hecha.
2: terminando ejecucion...
alex@alex-VirtualBox:~/Escritorio/1_Ej$
```

Figura 33: Servidor 2

Salida del servidor 3:

```
[servidor.cpp -- ~/Do... METRICA_V3_Analisis... (7) WhatsApp - Mozill... Terminal - estebansc... 1_Ej - Administrador... Descargas - Administ... Untitled-1 - Untitled (... 23 may, 16:49
Terminal - estebansc@estebansc-Lenovo-ideapad-110-14ISK: ~/Documentos/Distribuidos/Parcial2/1_Ej
Archivo Editar Ver Terminal Pestañas Ayuda
Esta solicitud ya ha sido hecha.
3: terminando ejecucion...
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ ./servidor 3
Starting web server on port specified
3: terminando ejecucion...
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$ ./servidor 3
Starting web server on port specified

Esta solicitud ya ha sido hecha.

Esta solicitud ya ha sido hecha.

Esta solicitud ya ha sido hecha.
3: terminando ejecucion...
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$
estebansc@estebansc-Lenovo-ideapad-110-14ISK:~/Documentos/Distribuidos/Parcial2/1_Ej$
```

Figura 34: Servidor 3

4. CONCLUSIONES

Durante el desarrollo de la práctica pudimos notar los siguientes puntos:

El desarrollo de esta práctica a sido a su vez el más pesado pero con más aprendizaje hasta ahora. El haber estudiado la herramienta Mongoose junto con todo lo que se ha ido aprendiendo en el curso, nos dió una visión más clara de como es que se construye un servicio web desde lo más básico que es la programación de sockets hasta lograr programar un servicio RESTful con una interfaz amigable.

Sobre el sistema, podemos decir que estamos bastantes satisfechos con lo que se ha logrado, la implementación de cada una de las clases desde PaqueteDatagrama hasta las clases Solicitud y Respuesta, que implementan un protocolo confiable, han sido un trabajo laborioso pero que ha dado sus frutos y sobre el cual hemos construido un sistema distribuido bastante interesante con balanceo, sincronización y optimización, utilizando diferentes formas y algoritmos de comunicación entre procesos.

Sobre el desempeño de la aplicación, podemos observar de acuerdo a las pruebas un rendimiento favorable igual al que se obtenía en la práctica pasada (Sincronización) y esto es bueno pues el implementar los servidores web no supuso un lastre significativo en el tiempo de respuesta de los servidores de votos, gracias en mucha medida a que fue implementado usando hilos.