

SDEP: SINCRONIZACIÓN

ALUMNOS:

BASTIDA PRADO JAIME ARMANDO

ROJAS MONTAÑO MARCELL DOUGLAS

ORTÍZ RODRÍGUEZ SALVADOR ALEJANDRO

SÁNCHEZ CUEVAS ESTEBAN

PROFESOR: UKRANIO CORONILLA CONTRERAS

EQUIPO: 7

GRUPO: 4CM4

Mayo 2020

Índice

1. INTRODUCCIÓN	3
1.1. SISTEMA DISTRIBUIDO PARA ELECCIONES PRESIDENCIALES: SINCRONIZACIÓN	3
2. DESARROLLO DE LA PRÁCTICA	3
2.1. EJERCICIO 1	3
2.1.1. Respondiendo las preguntas	4
2.2. EJERCICIO 2	4
2.3. Haciendo las pruebas	7
2.3.1. PRUEBA CON 1000	7
2.3.2. PRUEBA CON 5000	10
2.3.3. PRUEBA CON 10000	11
3. CONCLUSIONES	13

1. INTRODUCCIÓN

1.1. SISTEMA DISTRIBUIDO PARA ELECCIONES PRESIDENCIALES: SINCRONIZACIÓN

Para el desarrollo de ésta práctica se busca implementar un mecanismo de sincronización entre los tres servidores para poder hacer el envío de timestamps únicos hacia el cliente sin que se repita ninguno, pues este es un problema que se llega a dar ocasionalmente cuando los servidores no se comunican entre sí porque, como veremos más adelante, cada servidor tiene su propio reloj el cual varía hasta por milisegundos de los demás, y esto puede provocar fallos a la hora de enviar los timestamps.

2. DESARROLLO DE LA PRÁCTICA

2.1. EJERCICIO 1

Elaboramos el código y procedimos a instalar un servidor NTP en ambas computadoras, creamos un servidor NTP en una utilizando el daemon ntp:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/1_Ej$ sudo systemctl status ntp
[sudo] password for james:
● ntp.service - Network Time Service
   Loaded: loaded (/lib/systemd/system/ntp.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-05-14 19:34:54 CDT; 1h 59min ago
     Docs: man:ntpd(8)
   Main PID: 15446 (ntpd)
      Tasks: 2 (limit: 4915)
    CGroup: /system.slice/ntp.service
            └─15446 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 122:127
```

Figura 1: Servidor NTP corriendo

Y en la otra máquina agregamos dicho servidor como el servidor ntp predeterminado para sincronizar sus relojes:

```
GNU nano 2.9.3 /etc/hosts
127.0.0.1    localhost
127.0.1.1    dracmaiv
192.168.100.107 NTP-james
```

Figura 2: Agregando servidor NTP desde el cliente

El nombre del servidor es "NTP-james", como se puede observar es el que esta siendo usado por la máquina cliente pues tiene el asterisco a un lado:

```
james@dracmaiv:~/Documents$ ntpq -p
remote           refid      st t when poll reach  delay  offset  jitter
=====
0.ubuntu.pool.n .POOL.     16 p -   64   0    0.000   0.000   0.000
1.ubuntu.pool.n .POOL.     16 p -   64   0    0.000   0.000   0.000
2.ubuntu.pool.n .POOL.     16 p -   64   0    0.000   0.000   0.000
3.ubuntu.pool.n .POOL.     16 p -   64   0    0.000   0.000   0.000
ntp.ubuntu.com  .POOL.     16 p -   64   0    0.000   0.000   0.000
*NTP-james      91.189.91.157 3 u 49  64  377 66.035   9.799 17.594
```

Figura 3: Comprobación del servidor NTP

Procedimos a hacer la prueba y grabamos el siguiente vídeo:

<https://photos.app.goo.gl/5ez9WK2fm1rXGCJc9>

2.1.1. Respondiendo las preguntas

La diferencia de tiempo entre ambas computadoras como se puede observar en el video es de unos pocos microsegundos, pero la hay, y eso puede provocar que en alguna ocasión se envíe un timestamp repetido.

Ejecutando el programa en dos terminales dentro de la misma computadora no hay diferencia.

2.2. EJERCICIO 2

Implementamos la solución asignando un número a cada servidor y haciendo que los servidores se comuniquen entre sí usando mensajes unicast a manera de una cadena de mensajes; es decir, el servidor designado como "servidor 1" se comunica con el servidor 2 enviándole su timestamp para que éste corrobore que no se repitan, posterior a ello, el servidor 2 se comunica con el servidor 3 enviándole el timestamp del servidor 1 y de él mismo, para que haga lo correspondiente.

Para lograr esto, al correr cada servidor, le indicamos el número de servidor correspondiente junto con el puerto en el que va a correr, por convención utilizamos el puerto 7200 para el servidor 1, el 7201 para el 2 y el 7202 para el 3. Internamente el servidor 2 utiliza el puerto 8201 para recibir los timestamps y el servidor 3 el 8202. Además, cada servidor conoce de antemano la dirección IP de los demás servidores, cosa que podemos modificar al principio del programa, así como con el cliente.

```
19
20 // #define SERVIDOR_IP_1 "127.0.0.1"
21 // #define SERVIDOR_IP_1 "192.168.100.99" // Cel
22 #define SERVIDOR_IP_1 "25.142.15.73" // Alex
23 // #define SERVIDOR_IP_3 "25.121.204.24" // Jaime
24
25 #define SERVIDOR_IP_2 "127.0.0.1"
26 // #define SERVIDOR_IP_2 "25.47.119.146" // Esteban
27 // #define SERVIDOR_IP_2 "25.121.204.24" // Jaime
28
29 #define SERVIDOR_IP_3 "127.0.0.1"
30 // #define SERVIDOR_IP_3 "192.168.100.110" // Compu
31 // #define SERVIDOR_IP_3 "25.78.221.110" // Douglas
32 // #define SERVIDOR_IP_3 "25.121.204.24" // Jaime
33
34
```

Figura 4: IPs de los servidores

Para hacer esta comunicación entre servidores se ocupa el protocolo Solicitud-Respuesta Confiable a través de los objetos Solicitud y Respuesta de las clases programadas previamente:

```
if(isServer2Alive)
{
    // ENVIAMOS NUESTRO TIMESTAMP AL SERVIDOR ||2|| PARA QUE VERIFIQUE, AN
    men_respuesta = solicitud.doOperation(SERVIDOR_IP_2, 8201, timestamp_,
    // printf("1: Server 2 dice: %s\n", men_respuesta->arguments);
}
```

Figura 5: Enviando timestamp(s)

Cada que un servidor recibe el timestamp del otro, responde con un acuse, que es simplemente una cadena:

```
// ENVIANDO ACUSE A SERVIDOR 1
respuesta1->sendReply(acuse, strlen(acuse));
```

Figura 6: Mandando acuse de recibido

Cuando un hilo del cliente termina de enviar sus registros al servidor que le corresponde, le envía un último mensaje avisando que va a cerrar la comunicación, esto lo logra mandando un mensaje con ID de operación cerrando_comunicacion que definimos en la cabecera mensaje.h:

```
// CERRANDO COMUNICACIÓN CON EL SERVIDOR CORRESPONDIENTE
char mensaje_adios[] = "Cerrando comunicacion";
if(num_hilo == 1)
    men_respuesta = solicitud.doOperation(SERVIDOR_IP_1, SERVIDOR_PUERTO_1, cerrando_comunicacion, t
else if(num_hilo == 2)
    men_respuesta = solicitud.doOperation(SERVIDOR_IP_2, SERVIDOR_PUERTO_2, cerrando_comunicacion, t
else if(num_hilo == 3)
    men_respuesta = solicitud.doOperation(SERVIDOR_IP_3, SERVIDOR_PUERTO_3, cerrando_comunicacion, t
```

Figura 7: Hilo de cliente cerrando comunicación

Esto le sirve al servidor correspondiente para avisar a los demás que "se sale del grupo", y así ya no se tengan que comunicar con él para comparar timestamps. Cuando queda un solo servidor, este ya no necesita comparar sus timestamps con ningún otro servidor así que solo se dedica a procesar los registros que le llegan con normalidad.

```
else if(mensaje_rcv->operationId == cerrando_comunicacion)
{
    if(num_server == 2)
    {
        // AVISANDO A SERVIDOR 1 QUE SERVIDOR 2 ESTA FUERA
        mensaje_rcv1 = respuesta1->getRequest();
        if(isServer3Alive)
            respuesta1->sendReply(mensaje_salida, strlen(mensaje_salida));
        else
            respuesta1->sendReply(mensaje_only_one, strlen(mensaje_only_one));
    }
    else if(num_server == 3)
    {
        // AVISANDO A SERVIDOR 2 QUE SERVIDOR 3 ESTA FUERA
        mensaje_rcv2 = respuesta2->getRequest();
        respuesta2->sendReply(mensaje_salida, strlen(mensaje_salida));
    }
}
```

Figura 8: Servidor cerrando comunicación y avisando a los demás

Cuando un servidor recibe el mensaje de salida del otro, por medio de banderas, cambia su configuración para comunicarse con el o los que siguen activos:

```
if(isServer2Alive)
{
    // ENVIAMOS NUESTRO TIMESTAMP AL SERVIDOR ||2|| PARA QUE VERIFIQUE, ANTES
    men_respuesta = solicitud.doOperation(SERVIDOR_IP_2, 8201, timestamp_, 0);
    // printf("1: Server 2 dice: %s\n", men_respuesta->arguments);
    if(men_respuesta->arguments[0] == 'S')
    {
        printf("1: Cambiando a server 3\n");
        isServer2Alive = false;
        solicitud.requestId--;
    }
    else if(men_respuesta->arguments[0] == '0')
    {
        isServer2Alive = false;
        isServer3Alive = false;
        printf("1: Servidor 2 y 3 fuera\n");
    }
}
```

Figura 9: Servidor recibiendo mensaje de salida y cambiando bandera

2.3. Haciendo las pruebas

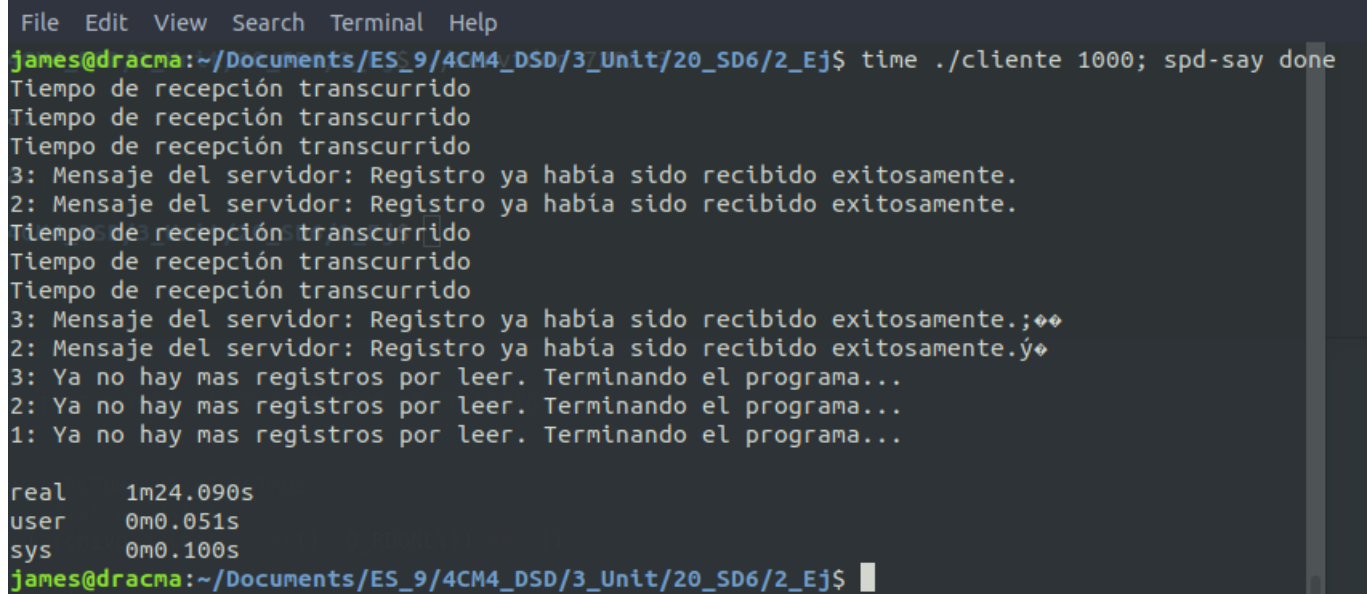
Hicimos pruebas con 1000, 5000 y 10000 votos:

2.3.1. PRUEBA CON 1000

En esta prueba el servidor 3 fue el primero en terminar, avisando al servidor 2, quien imprime el mensaje Cambiando solo a servidor 1; así solo se mantienen checando los timestamps entre el servidor 1 y 2, después terminó el servidor 2 avisando al servidor 1 que es el único servidor que queda, quien entonces imprime el mensaje Servidor 2 y 3 fuera y continua su ejecución, ya sin comparar su timestamp con nadie.

Como siempre, podemos observar mensajes de solicitudes repetidas, esto se debe a la ocasional pérdida de mensajes que hace a los hilos del cliente volver a enviar un registro, pero cuando el servidor detecta que un registro ya fue almacenado, regresa el mensaje de Registro ya había sido recibido exitosamente gracias a la implementación del protocolo Solicitud-Respuesta.

Cliente:

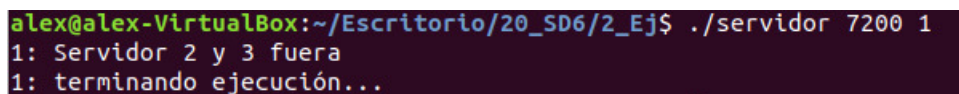


```
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ time ./cliente 1000; spd-say done
Tiempo de recepción transcurrido
Tiempo de recepción transcurrido
Tiempo de recepción transcurrido
3: Mensaje del servidor: Registro ya había sido recibido exitosamente.
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.
Tiempo de recepción transcurrido
Tiempo de recepción transcurrido
Tiempo de recepción transcurrido
3: Mensaje del servidor: Registro ya había sido recibido exitosamente.;♦♦
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...

real    1m24.090s
user    0m0.051s
sys     0m0.100s
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 10: Corriendo el cliente con 1000 registros

Servidor 1:



```
alex@alex-VirtualBox:~/Escritorio/20_SD6/2_Ej$ ./servidor 7200 1
1: Servidor 2 y 3 fuera
1: terminando ejecución...
```

Figura 11: Salida del servidor 1

Servidor 2:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 7201 2
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
192.168.100.99" // Cel
Esta solicitud ya ha sido hecha.
2: Cambiando solo a servidor 1
2: terminando ejecucion...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 12: Salida del servidor 2

Servidor 3:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 7202 3
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
3: terminando ejecucion...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 13: Salida del servidor 3

Para verificar la integridad de los registros enviados, la suma del peso de los archivos (BDs) de cada uno de los servidores es igual a 56,000 bytes, pues fueron enviados 1000 registros cada uno con un peso de 56 bytes, como podemos ver a continuación esto fue correcto:

Archivo del servidor 1:

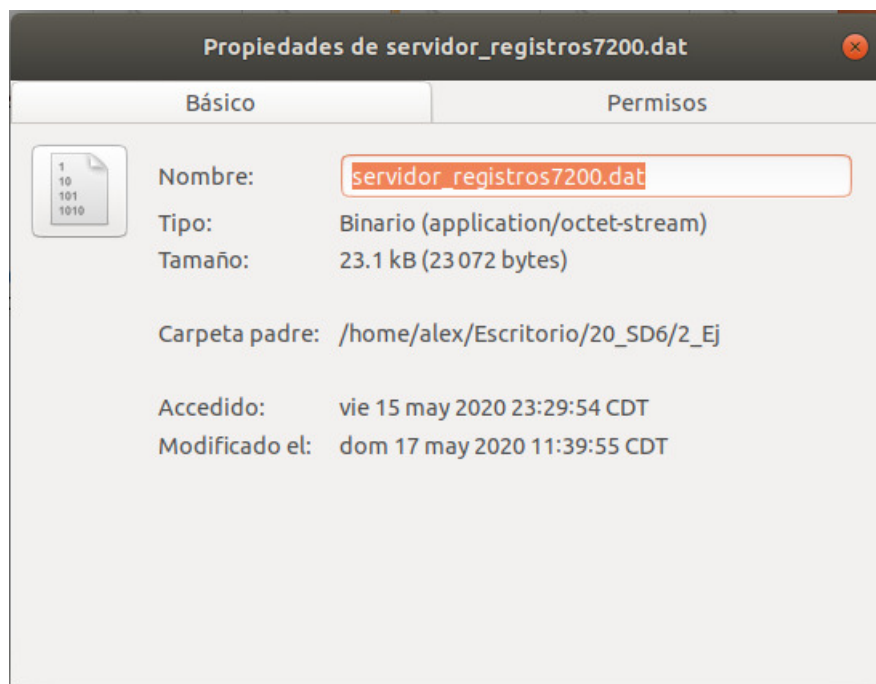


Figura 14: Archivo generado en el servidor 1

Archivo del servidor 2:

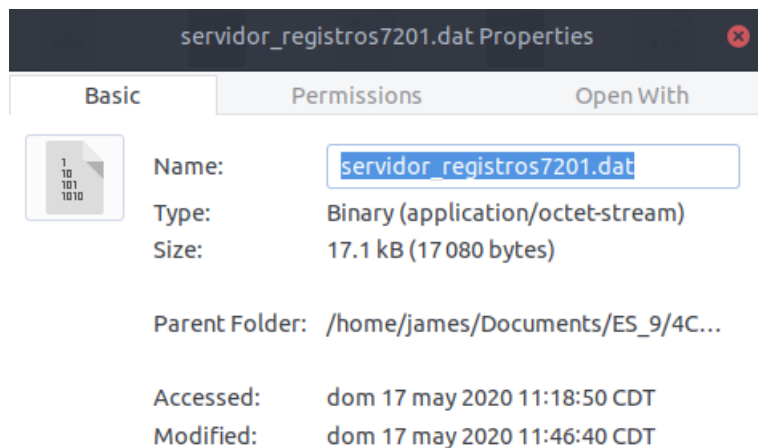


Figura 15: Archivo generado en el servidor 2

Archivo del servidor 3:

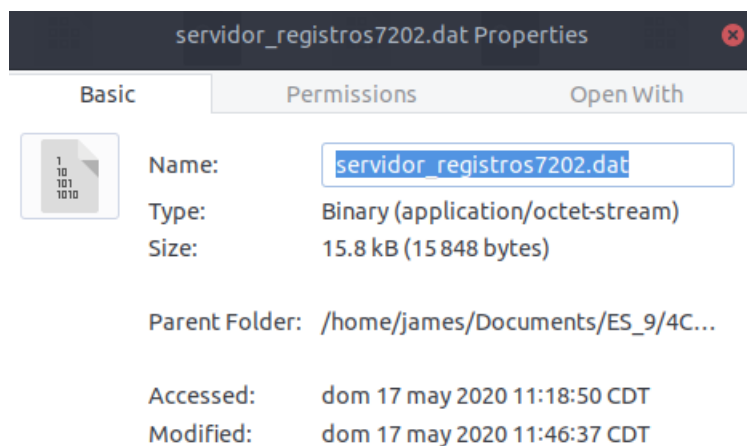


Figura 16: Archivo generado en el servidor 3

2.3.2. PRUEBA CON 5000

En esta prueba el servidor 2 fue el primero en terminar, avisando al servidor 1, quien imprime el mensaje Cambiando a servidor 3; así solo se mantienen checando los timestamps entre el servidor 1 y 3, después terminó el servidor 3 avisando al servidor 1 que es el único servidor que queda, quien entonces imprime el mensaje Servidor 2 y 3 fuera; continua su ejecución, ya sin comparar su timestamp con nadie.

Cliente:

```
Tiempo de recepción transcurrido
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.
Tiempo de recepción transcurrido
Tiempo de recepción transcurrido
2: Mensaje del servidor: Registro ya había sido recibido exitosamente.
3: 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...
1: Ya no hay mas registros por leer. Terminando el programa...

real    5m53.454s
user    0m0.126s
sys     0m0.526s
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 17: Corriendo el cliente con 5000 registros

Servidor 1:

```
alex@alex-VirtualBox:~/Escritorio/20_SD6/2_Ej$ ./servidor 7200 1

Esta solicitud ya ha sido hecha.
1: Cambiando a server 3
1: Servidor 2 y 3 fuera
1: terminando ejecución...
```

Figura 18: Salida del servidor 1

Servidor 2:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 7201 2

Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
2: terminando ejecución...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 19: Salida del servidor 2

Servidor 3:

```
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 3 2020 3 20_SD6
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
3: terminando ejecucion...
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 20: Salida del servidor 3

2.3.3. PRUEBA CON 10000

Cliente:

```
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.
3: Mensaje del servidor: Registro ya había sido recibido exitosamente.
2: 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...
Tiempo de recepción transcurrido
1: Mensaje del servidor: Registro ya había sido recibido exitosamente.
1: Ya no hay mas registros por leer. Terminando el programa...
real_0m11.310s
user      0m0.273s
sys       0m1.119s
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 21: Corriendo el cliente con 5000 registros

Servidor 1:

```
alex@alex-VirtualBox:~/Escritorio/20_SD6/2_Ej$ ./servidor 7200 1
Tiempo de recepción transcurrido

Esta solicitud ya ha sido hecha.
Tiempo de recepción transcurrido

Esta solicitud ya ha sido hecha.

Esta solicitud ya ha sido hecha.
1: Cambiando a server 3
1: Servidor 2 y 3 fuera

Esta solicitud ya ha sido hecha.
1: terminando ejecución...
```

Figura 22: Salida del servidor 1

Servidor 2:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 7201 2
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
3: Mensaje del servidor: Registro ya habia sido recibido exitosamente.
Esta solicitud ya ha sido hecha.
Tiempo de recepción transcurrido
Esta solicitud ya ha sido hecha.
2: terminando ejecución..
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 23: Salida del servidor 2

Servidor 3:

```
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$ ./servidor 7202 3
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
Esta solicitud ya ha sido hecha.
3: terminando ejecución..
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/20_SD6/2_Ej$
```

Figura 24: Salida del servidor 3

3. CONCLUSIONES

El desarrollo de esta práctica fue de los más difíciles hasta ahora, pues si implementar el balanceo fue un reto la sincronización lo fue aún más, afortunadamente se consiguieron resultados muy buenos y estamos satisfechos.

Podemos notar que el rendimiento de los tres servidores bajó, cosa que es normal debido a que ahora se deben sincronizar, sin embargo, se ha mantenido una velocidad de respuesta mejor que cuando solo se tenía un servidor y ahora con un mecanismo de balanceo y sincronización.

También se pudo haber implementado la sincronización utilizando multicast, sin embargo hemos tenido problemas al intentar esta comunicación por medio de hamachi, y usando Termux hasta donde nuestra investigación ha llegado solo es posible por medio de programación en Android quitar los seguros que impiden la llegada de mensajes multicast a teléfonos celulares. Sumado a esto también hemos tenido problemas para que los cuatro compañeros coincidamos para poder realizar las pruebas con los 3 servidores remotamente, ya sea por trabajo o incluso un compañero enfermó, además de que este programa no logramos correrlo en Termux, intentamos diferentes cosas todo un día y no nos fue posible utilizar algún celular como servidor.