

LABORATORIO 2

SISTEMAS DISTRIBUIDOS 2020

Barja, María Marta

1. Se implementaron las interfaces remotas `InterfaceRemota` que define los métodos suma y resta, e `InterfaceRemotaMD` que define multiplicación y división. Luego se implementaron las clases `ObjetoRemoto` y `ObjetoRemotoMD` que extienden de `UnicastRemoteObject` e implementan las interfaces remotas definidas. Luego en el servidor se instancian los 2 objetos remotos y se publican en la `rmiregistry`.

En el cliente se obtienen las referencias de los objetos remotos y se guardan en 2 variables. Luego se implementaron los 4 métodos solicitados (suma, resta, multiplicación y división) llamando a los métodos remotos de los 2 objetos. Por último, en el método `Main` se solicita al cliente que ingrese una operación y luego 2 números, según la operación ingresada ejecutará el método correspondiente.

- a. Se ejecutó el comando `netstat -a | find "LISTENING"` antes de lanzar y se obtuvo el siguiente resultado:

TCP	0.0.0.0:80	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:135	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:445	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:1099	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:2968	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5040	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5357	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5432	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5433	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:8733	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:13148	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49664	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49665	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49666	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49667	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49668	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49669	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49687	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5354	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5939	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12025	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12110	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12119	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12143	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12465	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12563	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12993	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12995	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:27275	LAPTOP-4ULBFG0N:0	LISTENING
TCP	192.168.0.104:139	LAPTOP-4ULBFG0N:0	LISTENING

Luego de lanzar el servidor se ejecutó nuevamente el comando y se observa que el puerto del servidor remoto es el 55872

TCP	0.0.0.0:80	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:135	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:445	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:1099	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:2968	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5040	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5357	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5432	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5433	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:8733	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:13148	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49664	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49665	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49666	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49667	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49668	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49669	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49687	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:55872	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5354	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5939	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12025	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12110	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12119	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12143	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12465	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12563	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12993	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12995	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:27275	LAPTOP-4ULBFG0N:0	LISTENING
TCP	192.168.0.104:139	LAPTOP-4ULBFG0N:0	LISTENING

Luego se detuvo el servidor y se lanzo nuevamente para verificar si el puerto en el que escucha cambio. Se ejecuto el comando nuevamente y se observa que el puerto del servidor es el 55947

TCP	0.0.0.0:80	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:135	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:445	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:1099	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:2968	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5040	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5357	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5432	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:5433	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:8733	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:13148	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49664	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49665	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49666	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49667	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49668	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49669	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:49687	LAPTOP-4ULBFG0N:0	LISTENING
TCP	0.0.0.0:55947	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5354	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:5939	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12025	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12110	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12119	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12143	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12465	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12563	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12993	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:12995	LAPTOP-4ULBFG0N:0	LISTENING
TCP	127.0.0.1:27275	LAPTOP-4ULBFG0N:0	LISTENING
TCP	192.168.0.104:139	LAPTOP-4ULBFG0N:0	LISTENING

- Como se puede observar en las tablas del punto anterior, en el cuarto lugar se encuentra el RMIRRegistry que atiende en el puerto 1099.
- Se aplica un filtro sobre el puerto 50076 que es el puerto del servidor, y al realizar una suma desde el cliente se identificaron los siguientes mensajes:

Capturing from Npcap Loopback Adapter (port 50076)

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

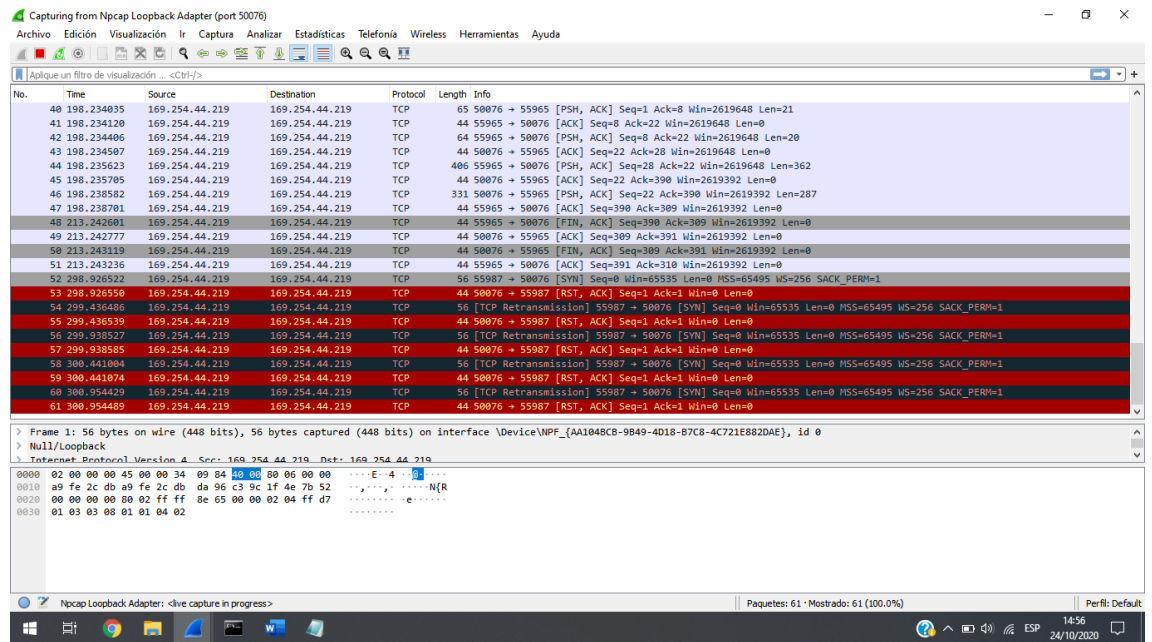
Aplique un filtro de visualización: <ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	169.254.44.219	169.254.44.219	TCP	56	55958 → 50076 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000100	169.254.44.219	169.254.44.219	TCP	56	50076 → 55958 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000303	169.254.44.219	169.254.44.219	TCP	44	55958 → 50076 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.000498	169.254.44.219	169.254.44.219	TCP	51	55958 → 50076 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=7
5	0.000528	169.254.44.219	169.254.44.219	TCP	44	50076 → 55958 [ACK] Seq=1 Ack=8 Win=2619648 Len=0
6	0.001457	169.254.44.219	169.254.44.219	TCP	65	50076 → 55958 [PSH, ACK] Seq=1 Ack=8 Win=2619648 Len=21
7	0.001494	169.254.44.219	169.254.44.219	TCP	44	55958 → 50076 [ACK] Seq=8 Ack=22 Win=2619648 Len=0
8	0.001630	169.254.44.219	169.254.44.219	TCP	64	55958 → 50076 [PSH, ACK] Seq=8 Ack=22 Win=2619648 Len=20
9	0.001661	169.254.44.219	169.254.44.219	TCP	44	50076 → 55958 [ACK] Seq=22 Ack=28 Win=2619648 Len=0
10	0.001792	169.254.44.219	169.254.44.219	TCP	93	55958 → 50076 [PSH, ACK] Seq=28 Ack=22 Win=2619648 Len=49
11	0.001820	169.254.44.219	169.254.44.219	TCP	44	50076 → 55958 [ACK] Seq=22 Ack=77 Win=2619648 Len=0
12	0.003274	169.254.44.219	169.254.44.219	TCP	70	50076 → 55958 [PSH, ACK] Seq=22 Ack=77 Win=2619648 Len=26
13	0.003330	169.254.44.219	169.254.44.219	TCP	44	55958 → 50076 [ACK] Seq=77 Ack=48 Win=2619648 Len=0

Npcap Loopback Adapter: <live capture in progress> Paquetes: 13 · Mostrado: 13 (100.0%) Perfil: Default

14:50 24/10/2020

Luego se detuvo la ejecución del servidor y se intento realizar otra suma desde el cliente. Estos fueron los paquetes identificados.



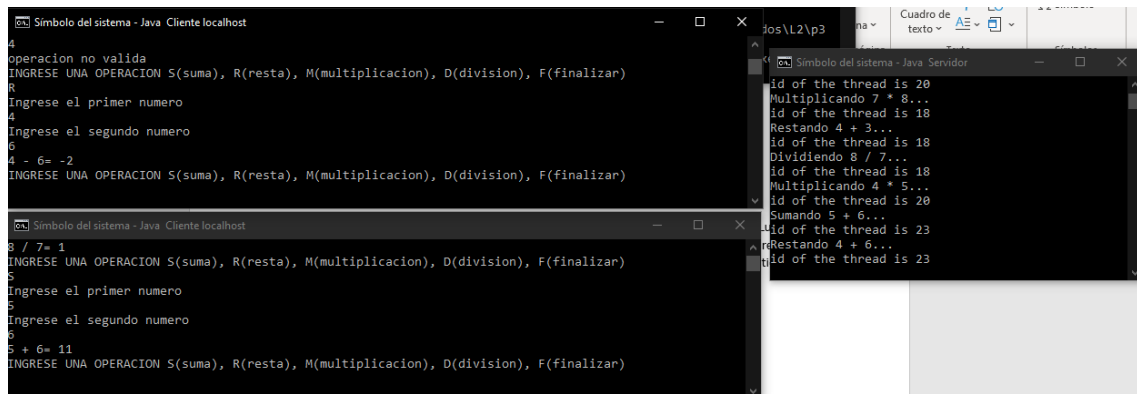
d. Como se observa en la imagen anterior, los mensajes se valen de TCP

2. Servidores concurrentes

- a. Para el correcto funcionamiento de un servidor concurrente, se deben poder gestionar por separado todas las peticiones de los clientes, por cada vez que se conecta un nuevo cliente al servidor, y además el servidor debe seguir esperando las peticiones de otros clientes. Para resolver esto normalmente se abre un socket y se inicia un bucle infinito en el cual, cada vez que se recibe una conexión entrante de un cliente a través del socket, se instancia un hilo que se encarga de procesar las peticiones de ese cliente. Luego el servidor continúa escuchando y repite este procedimiento por cada cliente que inicia una conexión.
- b. Java RMI se encarga automáticamente de desplegar los threads requeridos para dotar de concurrencia a un servicio implementado usando esta tecnología.
- c. Tanto RPC como RMI ofrecen interfaces que definen la funcionalidad remota para el empleo de los clientes (interfaz de servicio en RPC e interfaz remota para RMI), podría decirse que cuentan con el mismo nivel de abstracción ya que la implementación de la funcionalidad remota es transparente para los clientes en ambos casos. Podría agregarse también que hay una mayor abstracción con RMI, debido a la explotación de objetos, referencias, herencia, polimorfismo y excepciones, ya que la tecnología está integrada en el lenguaje.
- d. Similitudes entre RPC y RMI:
 - Ambos soportan programación con interfases, con los beneficios que surgen de este enfoque
 - Ambos se construyen sobre protocolos de petición/respuesta
 - Ambos ofrecen un nivel similar de transparencia, las llamadas locales y remotas emplean la misma sintaxis pero la interfaz remota típicamente expone la naturaleza distribuida de la llamada subyacente, por ejemplo soportando excepciones remotas

Diferencias entre RPC y RMI:

- RPC solo admite programación estructurada mientras que RMI admite programación orientada a objetos.
 - Los parámetros pasados a la llamada a procedimientos remotos consisten en estructuras de datos ordinarias. Por otro lado, los parámetros pasados al método remoto consisten en objetos.
- e. Se agregó un print del id del hilo a los métodos de los objetos concurrentes. Luego se lanzaron 2 clientes y se observó que los id de los hilos que ejecutaron los métodos remotos pueden ser diferentes para un mismo cliente. Por ello se concluye que Java RMI tiene un thread por requerimiento.



The image shows three overlapping Windows command prompt windows. The top-left window, titled 'Símbolo del sistema - Java Cliente localhost', displays a menu of operations (S, R, M, D, F) and user input for a subtraction operation (4 - 6 = -2). The bottom-left window, also titled 'Símbolo del sistema - Java Cliente localhost', shows a division operation (8 / 7 = 1) and user input for a subtraction operation (5 - 6 = 11). The rightmost window, titled 'Símbolo del sistema - Java Servidor', shows the server's output, including thread IDs (20, 18, 23) and the results of the operations performed by the clients: multiplication (7 * 8...), subtraction (4 + 3...), division (8 / 7...), multiplication (4 * 5...), and addition (5 + 6...).

```
Símbolo del sistema - Java Cliente localhost
operacion no valida
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
R
Ingrese el primer numero
4
Ingrese el segundo numero
6
4 - 6= -2
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)

Símbolo del sistema - Java Cliente localhost
8 / 7= 1
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
S
Ingrese el primer numero
5
Ingrese el segundo numero
6
5 - 6= 11
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)

Símbolo del sistema - Java Servidor
id of the thread is 20
Multiplicando 7 * 8...
id of the thread is 18
Restando 4 + 3...
id of the thread is 18
Dividiendo 8 / 7...
id of the thread is 18
Multiplicando 4 * 5...
id of the thread is 20
Sumando 5 + 6...
id of the thread is 23
Restando 4 + 6...
id of the thread is 23
```

3.

- a. Se agrego un delay de 10 segundos a todas las operaciones de ambos objetos remotos.

```
public int resta(int a, int b)
{
    System.out.println ("Restando " + a + " + " + b + "...");
    try{
        Thread.sleep(10000);
    }catch(InterruptedException ex){
        System.out.println(ex.getMessage());
    }
    return a-b;
}
```

Desde un cliente se ejecutó una operación de suma, y desde otro cliente una operación de multiplicación. Se verificó que el servidor recibe las 2 solicitudes de ambos clientes.

En la consola donde se ejecuta el servidor se observa que se imprimen ambos mensajes previos a la ejecución de los métodos.

```
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Servidor
Multiplicando 3 * 4...
Sumando 2 + 3...
```

Cliente 1

```
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Cliente localhost
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
S
Ingrese el primer numero
2
Ingrese el segundo numero
3
2 + 3= 5
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
```

Cliente 2

```
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Cliente localhost
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
M
Ingrese el primer numero
3
Ingrese el segundo numero
4
3 * 4= 12
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
```

- b. En el cliente, se establecieron las siguientes propiedades

```
System.setProperty("sun.rmi.transport.tcp.responseTimeout", "4000");
System.setProperty("sun.rmi.transport.connectionTimeout", "4000");
System.setProperty("sun.rmi.transport.tcp.readTimeout", "4000");
```


Al ejecutar una operación de suma, teniendo en cuenta que tenían definido un delay de 10 segundos, se observa este mensaje:

```
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Cliente localhost
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
S
Ingrese el primer numero
3
Ingrese el segundo numero
6
3 + 6= OCURRIO UNA EXCEPCION EN sumar
Error unmarshaling return header; nested exception is:
    java.net.SocketTimeoutException: Read timed out
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
```

- c. Se lanzaron dos clientes en simultáneo y se ejecuto una suma diferente en cada uno. Se verifica que los resultados son correctos, por ende las variables no fueron contaminadas

```

Simbolo del sistema - Java Cliente localhost
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Cliente localhost
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
S
Ingrese el primer numero
8
Ingrese el segundo numero
1
8 + 1= 9
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)

Simbolo del sistema - Java Cliente localhost
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Cliente localhost
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)
S
Ingrese el primer numero
3
Ingrese el segundo numero
5
3 + 5= 8
INGRESE UNA OPERACION S(suma), R(resta), M(multiplicacion), D(division), F(finalizar)

Simbolo del sistema - Java Servidor
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Servidor
Sumando 3 + 4...
Restando 2 + 5...
Sumando 3 + 6...
Sumando 5 + 7...
Sumando 6 + 7...
^C
C:\Users\Lala\Documents\SD2020\SistemasDistribuidos\L2\p3>Java Servidor
Sumando 8 + 1...
Sumando 3 + 5...

```

- d.

4. Para este ejercicio se implementaron las siguientes clases:

- Servidor.java

Esta clase extiende de Thread. Su constructor recibirá un número de puerto y creará un socket. En su método main instancia un objeto de Servidor con el número de puerto(1333), y lo inicia.

En el método run, dentro de un bucle infinito, que acepta y procesa las solicitudes de los clientes. Cuando un cliente envía una solicitud, setea una variable clienteActivo en verdadero, para que atienda todas las solicitudes del cliente hasta que el mismo le indique que se desconecta. El cliente le envía un entero con un código, 1 para solicitar la hora y 2 para indicarle que finalizó su consulta. Luego, dentro del bloque que esta procesando las solicitudes de hora del cliente, se agregó un sleep con un número random para simular diferentes retardos del servidor, ya que al estar cliente y servidor en la misma maquina los retardos eran siempre iguales y casi nulos. Para resumir, cuando se conecta un cliente, el servidor leerá el código que le envía el mismo, si es 1, hace un sleep y luego envía la hora. Así sucesivamente hasta que el cliente envíe un código distinto de 1.

```
public void run(){
    while (true){
        try {

            Socket server = serverSocket.accept();
            clienteConectado=true;
            DataInputStream in = new DataInputStream(server.getInputStream());
            DataOutputStream out = new DataOutputStream(server.getOutputStream());
            while(clienteConectado) {
                //lee el código que envía el cliente.
                int i=in.readInt();
                //si es 1, duerme y envía su hora.
                if(i==1)
                {
                    Thread.sleep((long)(100+new Random().nextInt(51)));
                    out.writeLong(System.currentTimeMillis());
                }
                //si es distinto de 1, el cliente finaliza las consultas
                else clienteConectado=false;
            }
            server.close();
        } catch (UnknownHostException ex) {
            System.out.println(ex.getMessage());
        } catch (IOException | InterruptedException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

- MuestraRelojServidor.java

Esta clase es para guardar las muestras que se obtienen de cada consulta que se realiza al servidor, cada instancia tendrá la hora del cliente, el retardo y la hora del servidor.

- Reloj.java

Esta clase representa a la interfaz del reloj, se crea en su constructor, y luego tiene un método actualizar hora que recibe un valor en milisegundos y actualiza la hora en pantalla.

- Cliente.java

Esta clase tiene definidas 2 clases estáticas: RelojDigital, que será el hilo que represente el reloj del cliente, y además instancia la interfaz de reloj y la actualiza en cada incremento de la hora. Y la clase ActualizarHora que se encarga de obtener las muestras del servidor, obtener la de menor retardo y analizar en base a esta muestra y la hora del cliente, si es necesario agrandar o achicar la deriva.

```
public static class RelojCliente extends Thread {
    Reloj relojDigital;

    public RelojCliente(long derivaIngresada)
    {
        Deriva=derivaIngresada;
        //Seteo hora actual - 10 minutos
        Timer = System.currentTimeMillis()-600000;
        relojDigital = new Reloj(Timer);
    }

    public void run(){
        while(true){
            try
            {
                Thread.sleep(Deriva);
                Timer+=1000;
                relojDigital.ActualizarHora(Timer);
                System.out.println(SDF.format(Timer));
            } catch (InterruptedException ex)
            {
                System.out.println(ex.getMessage());
            }
        }
    }
}
```

Como se observa en la imagen de la clase RelojDigital, la deriva inicial la recibe por parámetros, se verá mas adelante que el usuario debe definir este valor al iniciar el cliente. Además, se inicia el timer con la hora actual al que se lo retrasa 10 minutos para ver mas claramente la funcionalidad.

```

public static class ActualizarHora
{
    public void actualizar() throws IOException
    {
        System.out.println("ActualizarHora");
        ArrayList<MuestraRelojServidor> muestrasDelServidor = new ArrayList<MuestraRelojServidor>();
        long tsEnvio;
        long tsRecepcion;
        Socket client;
        client = new Socket(serverName, serverPort);
        OutputStream outToServer = client.getOutputStream();
        DataOutputStream out = new DataOutputStream(outToServer);
        InputStream inFromServer = client.getInputStream();
        DataInputStream in = new DataInputStream(inFromServer);
        for(int j=0;j<5;j++)
        {
            tsEnvio=System.currentTimeMillis();
            out.writeInt(1);
            long horaServer = in.readLong();
            tsRecepcion=System.currentTimeMillis();
            long retardo = (tsRecepcion-tsEnvio);
            muestrasDelServidor.add(new MuestraRelojServidor(Timer,retardo, horaServer));
        }
        out.writeInt(2);
        client.close();
        ObtenerDiferenciaYActualizarDeriva(muestrasDelServidor);
    }
}

```

En la imagen se ve el método actualizar de la clase actualizarHora, mas adelante se verá que el usuario debe indicar cada cuántos milisegundos desea que se ejecute. Este método se conecta con el servidor, y envía un código 1 para indicarle que le debe enviar la hora, se registra el timestamp inicial antes de la solicitud, y el timestamp final cuando recibe la respuesta, a partir de estas 2 variables obtiene el retardo. Luego instancia un objeto de la clase MuestraRelojServidor con la hora actual del cliente (tomada del timer de RelojCliente), el retardo que calculó, y la hora del servidor, este objeto muestra lo añade a un arreglo de muestras. Este proceso lo realiza 5 veces, luego le envía un código 2 al servidor y finaliza la conexión.

Luego llama al método ObtenerDiferenciaYActualizarDeriva con las 5 muestras del servidor obtenidas.

```

private void ObtenerDiferenciaYActualizarDeriva(ArrayList<MuestraRelojServidor> muestrasDelServidor) {
    MuestraRelojServidor muestraDeMenorRetardo = ObtenerMuestraDeMenorRetardo(muestrasDelServidor);
    long horaDelServidor = muestraDeMenorRetardo.ObtenerHoraRealDelServidor();

    System.out.println("horaCliente: "+muestraDeMenorRetardo.getHoraCliente());
    System.out.println("horaServidor: "+horaDelServidor);
    //Si la hora del cliente es mayor a la del servidor, debo agrandar la deriva
    System.out.println(Deriva);
    if(muestraDeMenorRetardo.getHoraCliente()>horaDelServidor) {
        Deriva*=2;
    }
    //Si la hora del cliente es menor a la del servidor, debo achicar la deriva
    if(muestraDeMenorRetardo.getHoraCliente()<horaDelServidor) {
        Deriva/=2;
    }
    //Si la hora del cliente es igual a la del servidor, dejo la deriva en 1 segundo
    if(muestraDeMenorRetardo.getHoraCliente()==horaDelServidor) {
        Deriva=1000;
    }
    System.out.println(Deriva);
}
}

```

En este método lo primero que se debe hacer es definir qué muestra utilizará para comparar con la hora local, la muestra de menor retardo se obtiene a

partir del método `ObtenerMuestraDeMenorRetardo` que recorre el arreglo de muestras comparándolas y devuelve la muestra que tenga el valor de retardo menor. Luego, para obtener la hora real del servidor que tiene la muestra, se llama al método `HoraRealDelServidor()` definido dentro de la clase `MuestraRelojServidor`, es decir que el objeto muestra obtiene su propia hora real y la devuelve.

```
public long ObtenerHoraRealDelServidor(){
    return this.horaServidor+(this.retardo/2);
}
```

Finalmente se compara la hora del cliente con la hora real del servidor, y en base a esta comparación la deriva se duplica o se reduce a la mitad.

Se observó que dividirla o multiplicarla por dos es muy extremo, sería mas adecuado tal vez reducirla o ampliarla en un 30% o algún valor mas cercano a la deriva actual. Se definió de esta manera para ver más clara la ejecución.

```
public static void main(String[] args) throws InterruptedException, IOException
{
    Scanner sc = new Scanner(System.in);
    long deriva;
    System.out.println("Ingrese la deriva inicial (ms)");
    deriva = sc.nextLong();
    String serverName = "localhost";
    int serverPort = 1333;
    Cliente client = new Cliente(serverName, serverPort);
    Cliente.RelojCliente relojCliente = new RelojCliente(deriva);
    Cliente.ActualizarHora actualizarHora = new ActualizarHora();

    System.out.println("Cada cuanto se deberá actualizar el reloj (ms)?");
    long CC=sc.nextLong();
    relojCliente.start();
    while(true)
    {
        Thread.sleep(CC);
        actualizarHora.actualizar();
    }
}
```

En la imagen del método `main` del cliente, se ve como se solicitan los parámetros de deriva e intervalo de actualización.

5. Para este ejercicio se utilizaron las clases SNTPCClient y NTPMessage. Se implementó una clase Tabla que representa la interfaz de la tabla. Además, recibe los valores de retardo, offset y fecha, los procesa y los añade como filas.

También se implementó una clase Cliente, que tiene una un objeto tabla que instancia en su constructor. Luego en el método main se llama al constructor y se solicita al usuario ingresar el servidor NTP que se desea consultar, se realizan 8 llamadas al método que consulta al servidor NTP ingresado. Se agregó un delay de 5 segundos entre consultas al servidor para que se vea como se va llenando la tabla.

```
public static void main(String[] args) throws IOException
{
    Cliente cliente = new Cliente();
    System.out.println("Cliente Iniciado");
    while(true){
        System.out.println("*****");
        System.out.println("Ingrese el servidor NTP a consultar");
        Scanner sc = new Scanner(System.in);
        String servidorNTP = sc.nextLine();
        for(int j=1;j<9;j++)
        {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("*****");
            System.out.println("Consulta "+j);
            cliente.ConsultarAlServidor(servidorNTP);
        }
    }
}
```

El método ConsultarAlServidor es igual al método main de la clase SNTPCClient.java con la única diferencia que cuando finaliza la consulta, llama al método agregarNuevaFila de la clase Tabla con los valores de roundTripDelay/2, localClockOffset, y la fecha y hora del mensaje.

Se realizaron pruebas con los servidores south-america.pool.ntp.org, jp.pool.ntp.org (servidor NTP de Japón) y au.pool.ntp.org (servidor NTP de Australia). Se observó que los retardos obtenidos con el servidor de América del sur son menores a los obtenidos respecto del servidor de Japón, y estos últimos a su vez son menores (casi la mitad) de los retardos obtenidos al consultar al servidor de Australia.

RoundTrip Delay	Offset	Offset estimado	Fecha y hora	Servidor NTP
30,97	18,83	-12,14 <= O <= 49,80	2020/10/24 12:08:35,017800	south-america.pool.ntp.org
23,91	26,26	2,35 <= O <= 50,17	2020/10/24 12:08:40,146167	south-america.pool.ntp.org
29,97	19,09	-10,87 <= O <= 49,06	2020/10/24 12:08:45,203064	south-america.pool.ntp.org
25,97	20,24	-5,73 <= O <= 46,21	2020/10/24 12:08:50,281212	south-america.pool.ntp.org
27,97	22,31	-5,66 <= O <= 50,27	2020/10/24 12:08:55,353275	south-america.pool.ntp.org
40,47	23,27	-17,20 <= O <= 63,75	2020/10/24 12:09:00,430745	south-america.pool.ntp.org
24,96	28,47	3,51 <= O <= 53,42	2020/10/24 12:09:05,515423	south-america.pool.ntp.org
55,97	55,42	-0,55 <= O <= 111,38	2020/10/24 12:09:10,646383	south-america.pool.ntp.org
103,97	31,04	-72,93 <= O <= 135,02	2020/10/24 12:09:33,047017	jp.pool.ntp.org
105,98	27,09	-78,89 <= O <= 133,07	2020/10/24 12:09:38,262074	jp.pool.ntp.org
122,98	6,90	-116,08 <= O <= 129,87	2020/10/24 12:09:43,485874	jp.pool.ntp.org
114,44	10,14	-104,30 <= O <= 124,58	2020/10/24 12:09:48,745581	jp.pool.ntp.org
116,96	3,74	-113,21 <= O <= 120,70	2020/10/24 12:09:53,992704	jp.pool.ntp.org
103,48	22,59	-80,88 <= O <= 126,07	2020/10/24 12:09:59,254066	jp.pool.ntp.org
103,48	15,10	-88,38 <= O <= 118,58	2020/10/24 12:10:04,479580	jp.pool.ntp.org
114,47	18,21	-96,26 <= O <= 132,68	2020/10/24 12:10:09,717678	jp.pool.ntp.org
204,47	30,86	-173,62 <= O <= 235,33	2020/10/24 12:10:24,211331	au.pool.ntp.org
208,98	35,08	-173,89 <= O <= 244,06	2020/10/24 12:10:29,635060	au.pool.ntp.org
200,97	28,52	-172,46 <= O <= 229,49	2020/10/24 12:10:35,058494	au.pool.ntp.org
199,47	32,64	-166,83 <= O <= 232,11	2020/10/24 12:10:40,482105	au.pool.ntp.org
201,97	24,23	-177,74 <= O <= 226,20	2020/10/24 12:10:45,883203	au.pool.ntp.org
214,44	12,14	-202,30 <= O <= 226,57	2020/10/24 12:10:51,301574	au.pool.ntp.org
213,97	24,32	-189,64 <= O <= 238,29	2020/10/24 12:10:56,760289	au.pool.ntp.org
192,44	27,93	-164,50 <= O <= 220,37	2020/10/24 12:11:02,183373	au.pool.ntp.org

6. Protocolo de 2 fases:

Fase 1: cada participante vota para que la transacción sea consumada o abortada. Una vez que un participante ha votado por la consumación de la transacción, no se le permite que la aborte. Por tanto, antes de que un participante vote para que se consume la transacción, debe asegurarse de que será capaz de llevar a cabo su parte del protocolo de consumación, incluso si falla y es reemplazado en su transcurso. Se dice que un participante en una transacción en estado preparado para una transacción si será finalmente capaz de consumarla. Para estar seguro de esto, cada participante guarda un dispositivo de almacenamiento permanente todos los objetos que haya alterado durante la transacción, junto con su estado (preparado).

Fase 2: todo participante en la transacción lleva a cabo la decisión conjunta. Si alguno de los participantes vota por abortar, entonces la decisión ha de ser de abortar la transacción. Si todos los participantes votan consumir, entonces la decisión es de consumir la transacción.

Protocolo de 2 fases para transacciones anidadas:

En esta aproximación, el protocolo de consumación en dos fases se convierte en un protocolo anidado multi-nivel. El coordinador de la transacción de nivel superior se comunica con los coordinadores de las subtransacciones para las que es su madre inmediata. Envía mensajes consultando si pueden comprometer a cada una de estas últimas, las cuales, a su vez, los pasan a los coordinadores de sus transacciones hijas (y así sucesivamente en el árbol). Cada participante recoge las respuestas de sus descendientes antes de responder sus madres.

Los argumentos necesarios del mensaje puedeComprometer son el id de la transacción de nivel superior que será usado cuando se preparen los datos. El segundo argumento es el id de la transacción del participante que hace la llamada a puedeConsumar. El participante que recibe la llamada busca en su lista de transacciones cualquier transacción o subtransacción consumada provisionalmente que concuerde con el id de transacción del segundo argumento.

Si un participante encuentra cualquier subtransacción que se ajuste al segundo argumento, prepara los objetos y responde con un voto SI. Si no consigue encontrar alguna, entonces debe haberse caído desde que realizó la subtransacción y le responde con un voto NO.

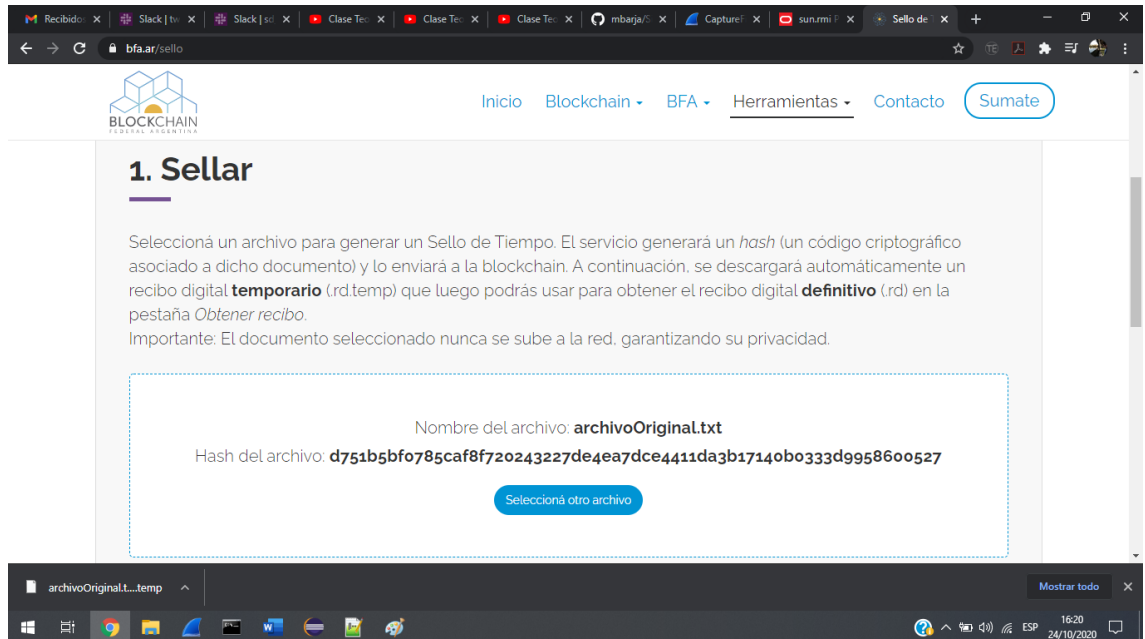
El protocolo de consumación en 2 fases para el caso de transacciones anidadas puede ocasionar que se retrasen el coordinador o un participante en los mismos pasos que la versión no anidada. Aparece un cuarto paso en el cual la subtransacción puede retrasarse. En el caso de las subtransacciones consumadas provisionalmente y que son hijas de subtransacciones abortadas, puede que no sean informadas del resultado de la votación de la transacción. Para abordar estas situaciones, cualquier subtransacción que no haya recibido un mensaje puedeConsumar indagará pasado un periodo de tiempo limite. Para hacer posibles tales preguntas, los coordinadores de las subtransacciones

abortadas han de sobrevivir un cierto período de tiempo. Si una subtransacción huérfana no puede contactar con su madre, al final, abortara.

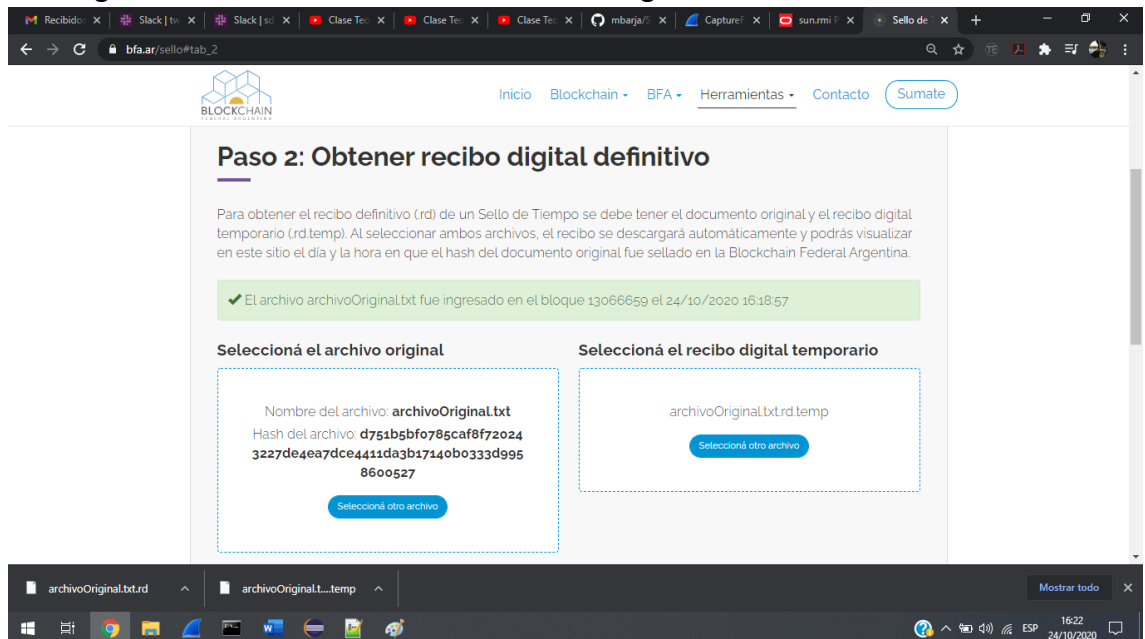
7. Se creo un archivo .txt

Bfa.ar

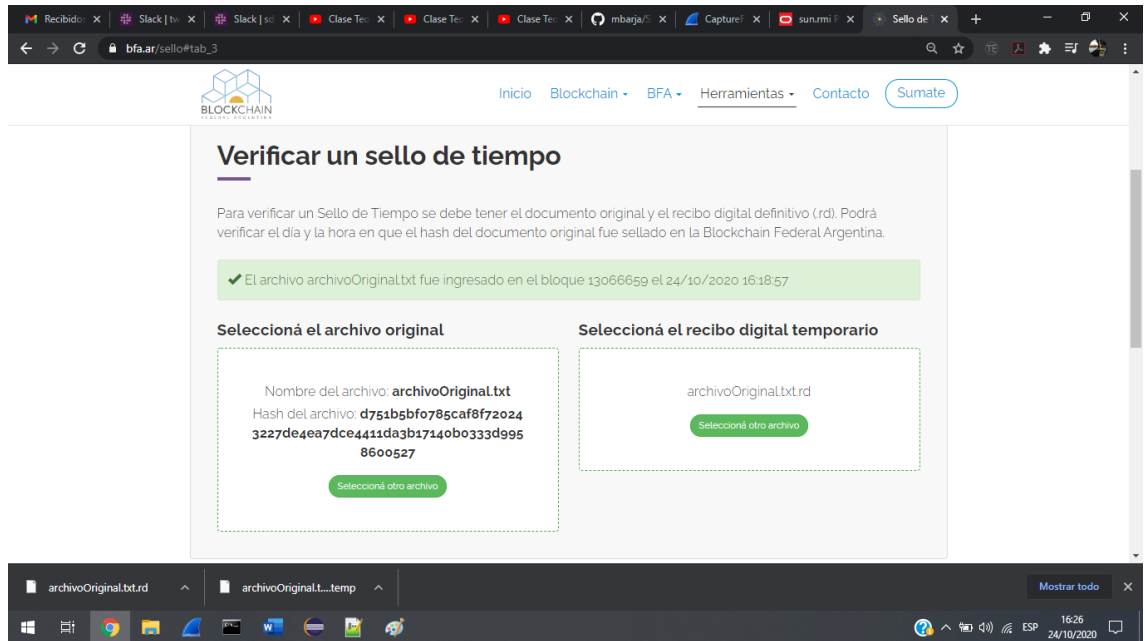
Se importa el archivo archivoOriginal.txt, y automáticamente se descargó el recibo digital archivoOriginal.txt.rd.temp y se generó el hash.



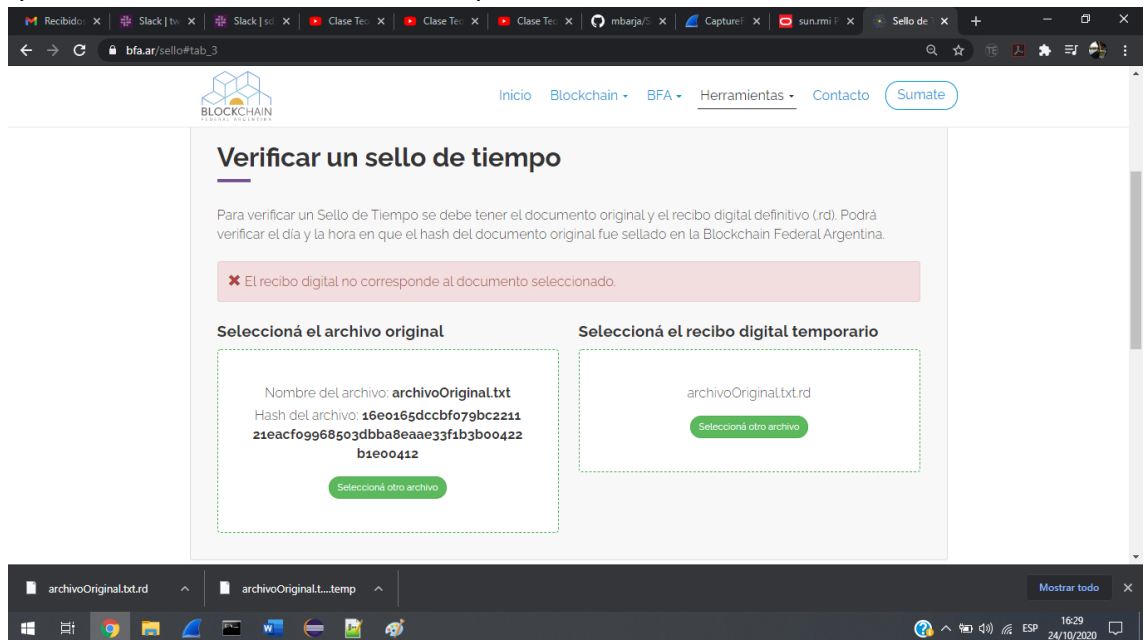
Luego en la pestaña de Obtener recibo, se importaron ambos archivos y se descarga automáticamente el archivo archivoOriginal.txt.rd



Finalmente, en la pestaña verificar, se ingreso el archivo original y el .rd y se verifica el sello de tiempo del archivo.



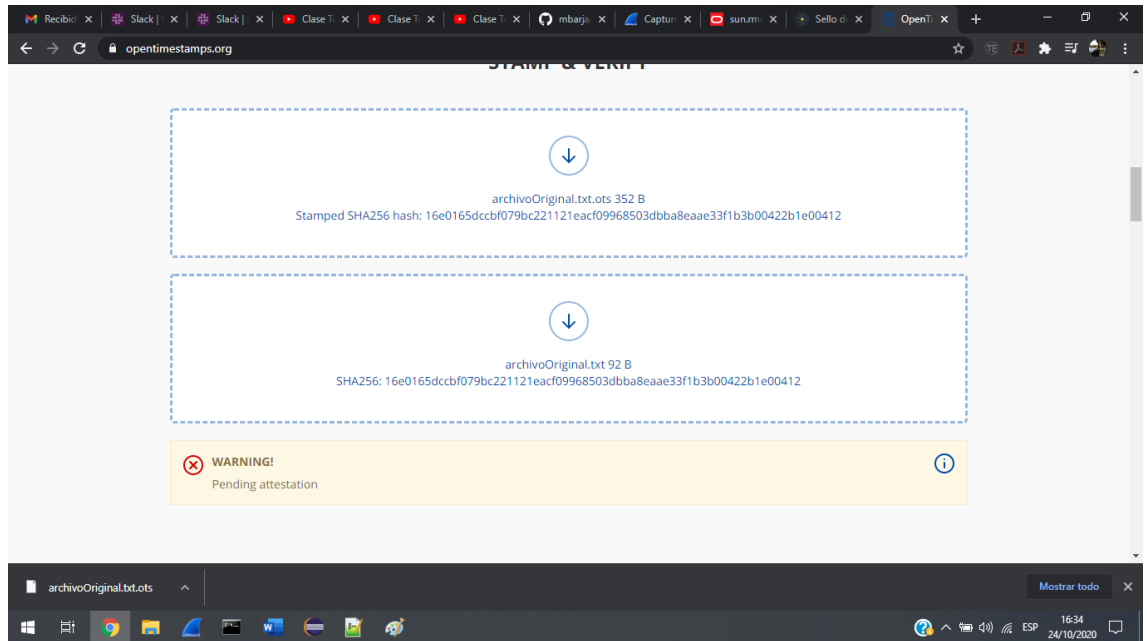
Luego se alteró el archivo original, y se intento verificar nuevamente. Se observa que no se valida el archivo una vez que fue modificado.



<https://opentimestamps.org/>

Se inserto el mismo archivoOriginal.txt y automáticamente se descarga un archivo con extensión .ots

Luego se importaron el archivo original y su extensión .ts, y se observa que lo verifica, pero muestra una advertencia por qué aun no fue incluido en ningún bloque.



Se modificó el archivo original y se intentó verificar nuevamente y se observó que no fue verificado.

