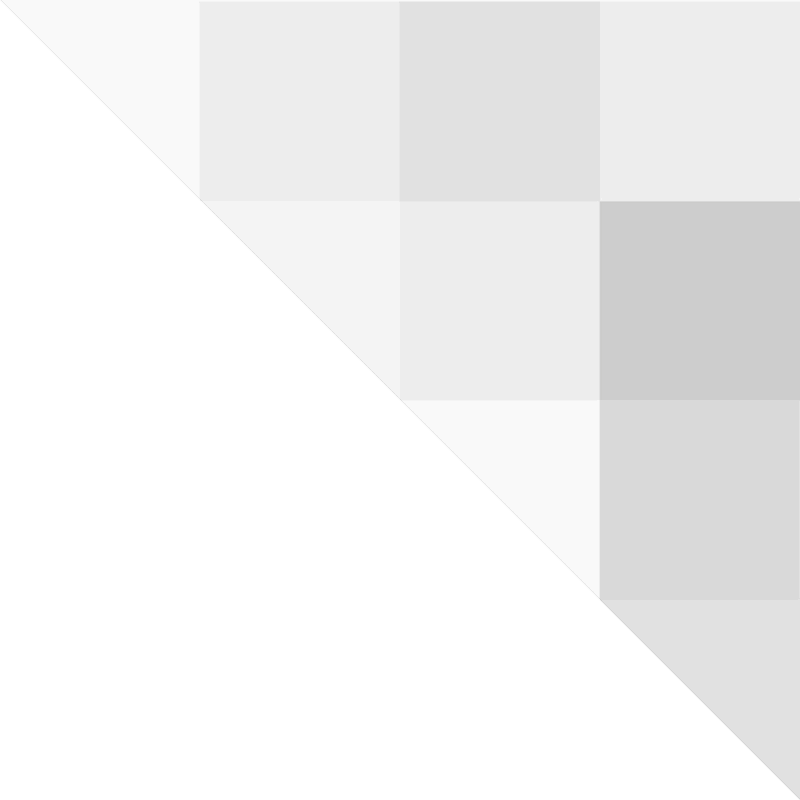
Sistemas Distribuidos

# Trabajo de Laboratorio 2



# Docentes

* Parisse Cristian
* Konstantinoff Pedro

# Alumnos

* Garibaldi Anele
* Ducid Matias

**1)** Realizamos la implementación del código fuente necesario para realizar las operaciones de suma, resta, multiplicación y división haciendo uso de RMI (Remote Method Invocation).

1. **En cuáles puertos atiende cada uno de los objetos remotos de los servidores en una determinada ejecución? Cambian si se vuelve a lanzar?.**

El objeto remoto utilizado para la suma y resta fue atendido en el puerto 46740.  
EL objeto remoto utilizado para la multiplicacion y division fue atendido en el puerto 46736.

Al volver a correr los objetos remotos se obtuvieron nuevos puertos, estos fueron, para el objeto remoto de suma y resta se observó que se le otorgó el puerto 46754 y para el objeto remoto de multiplicacion y division el puerto otorgado fué 46758.

Respondiendo a la pregunta, al volver a lanzar los objetos remotos cambian los puertos que se les dió.

A continuación mostraremos los 4 handshakes realizados por los objetos remotos al momento de instanciarse.

Handshake del primer objeto remoto de suma y resta:



Handshake del primer objeto remoto de multiplicacion y division:



Handshake del segundo objeto remoto de suma y resta:



Handshake del segundo objeto remoto de multiplicacion y division:

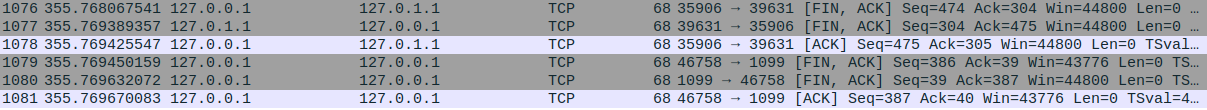
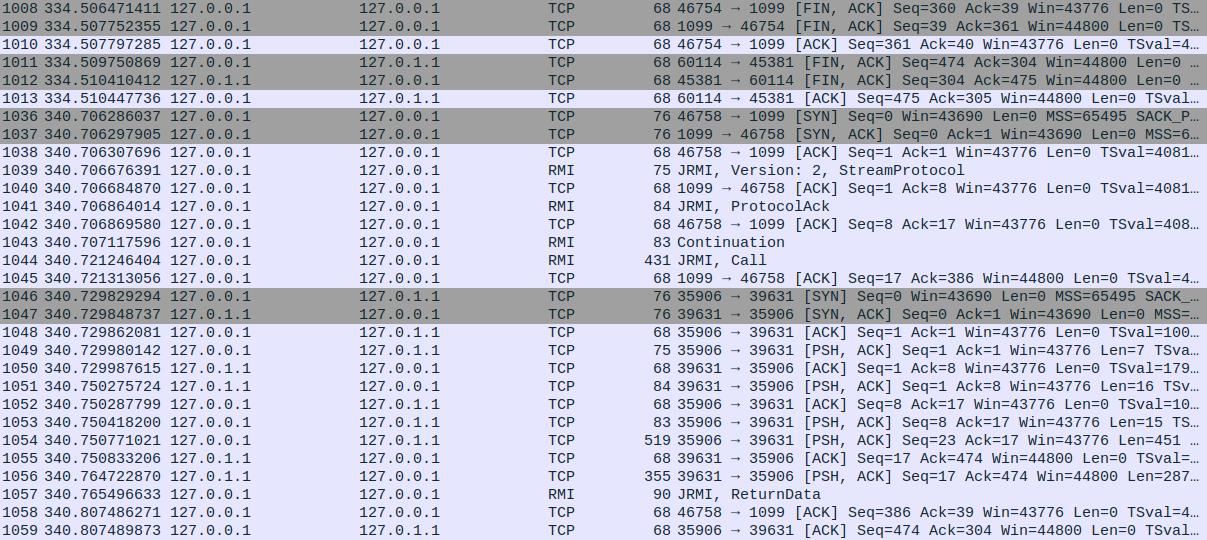
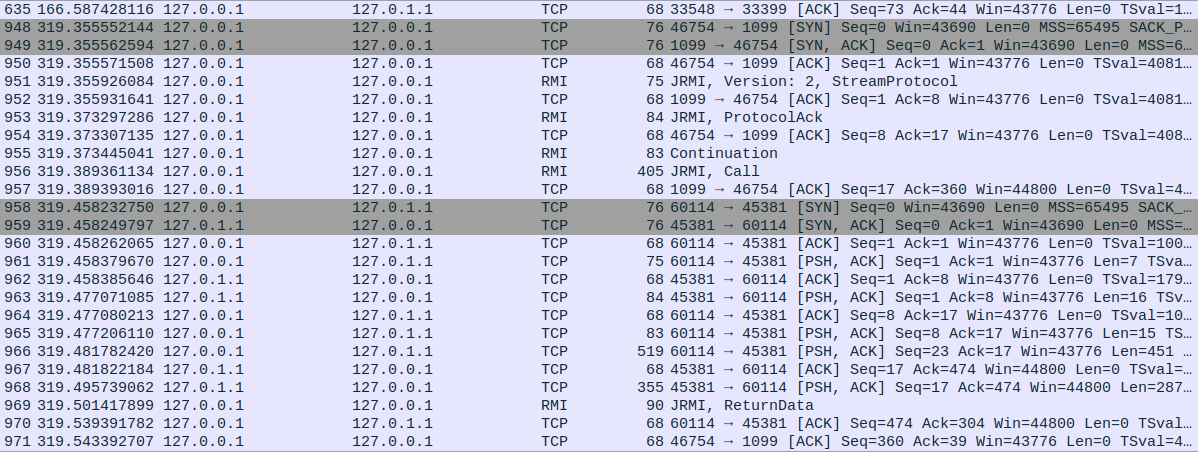
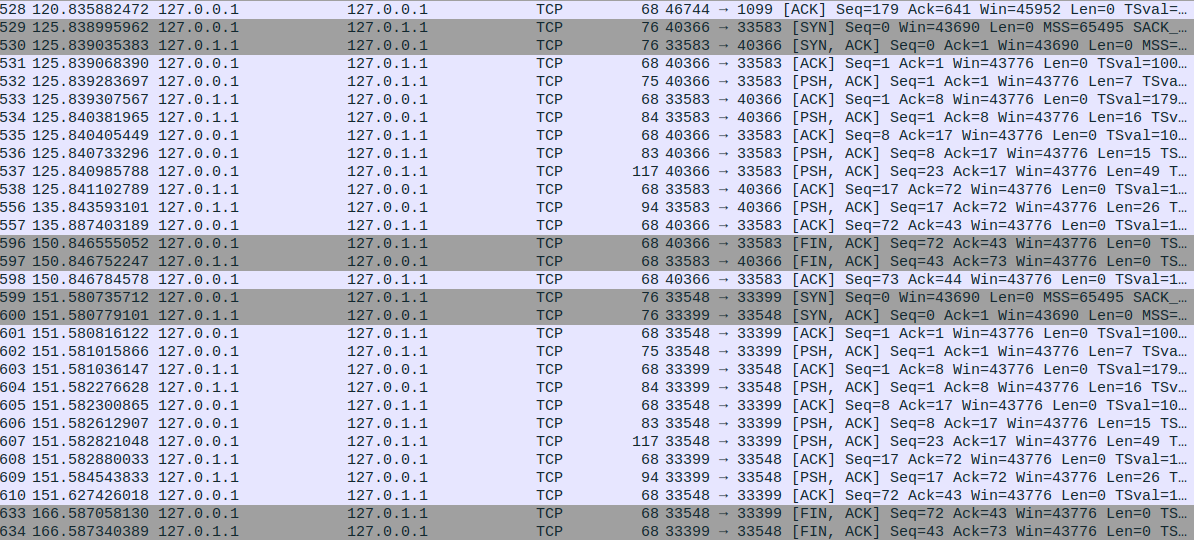
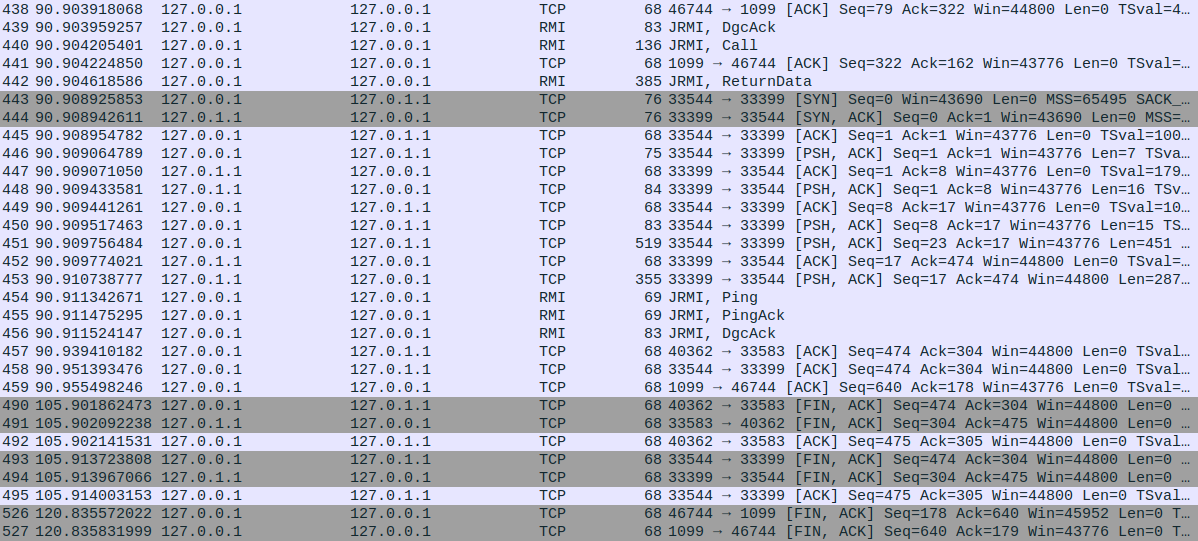
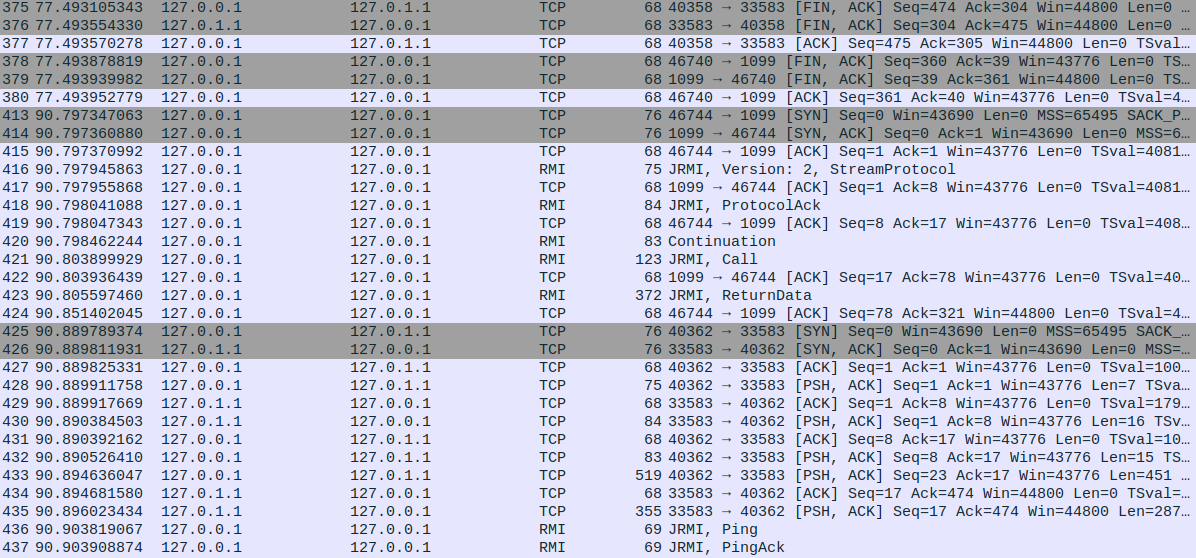
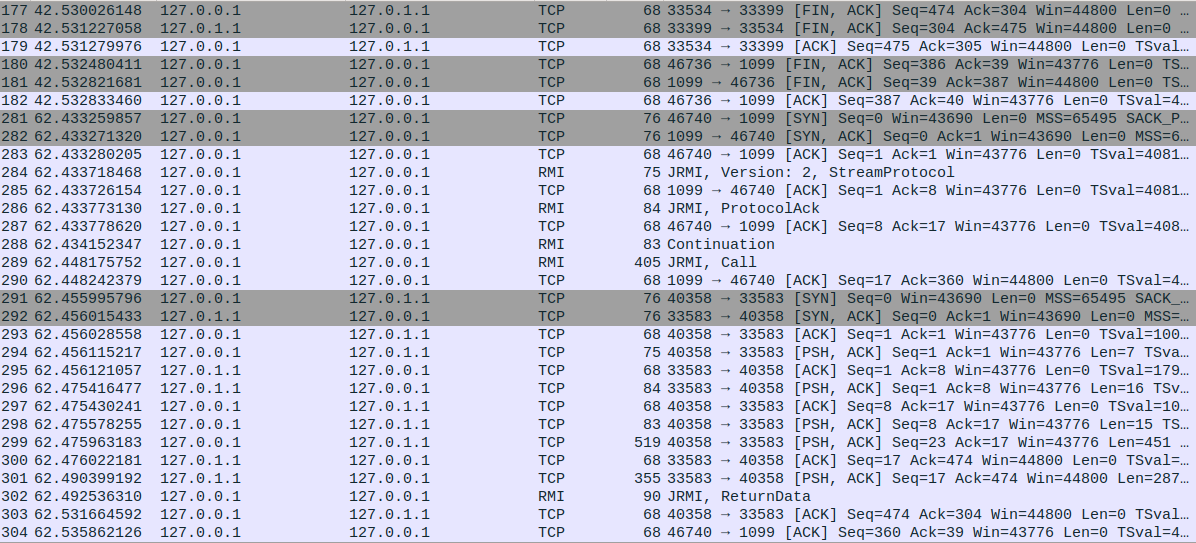
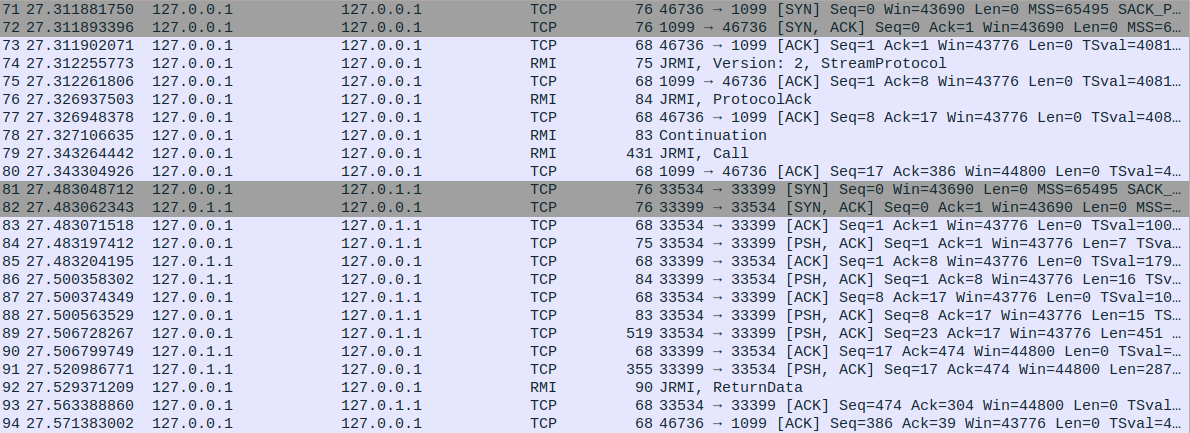


1. **Cuál es el puerto en el que atiende RMIRegistry?.**

RMI Registry es atendido en el puerto 1099.

1. **Identificar el/los mensajes que transfiere el cliente con RMIRegistry.**

Lo que hicimos fue separar los paquetes enviados al puerto 1099 (RMI Registry) usando el filtro *tcp.port == 1099* de Wireshark.  
Como resultado se tienen todos los paquetes transmitidos desde y hacia el puerto 1099.

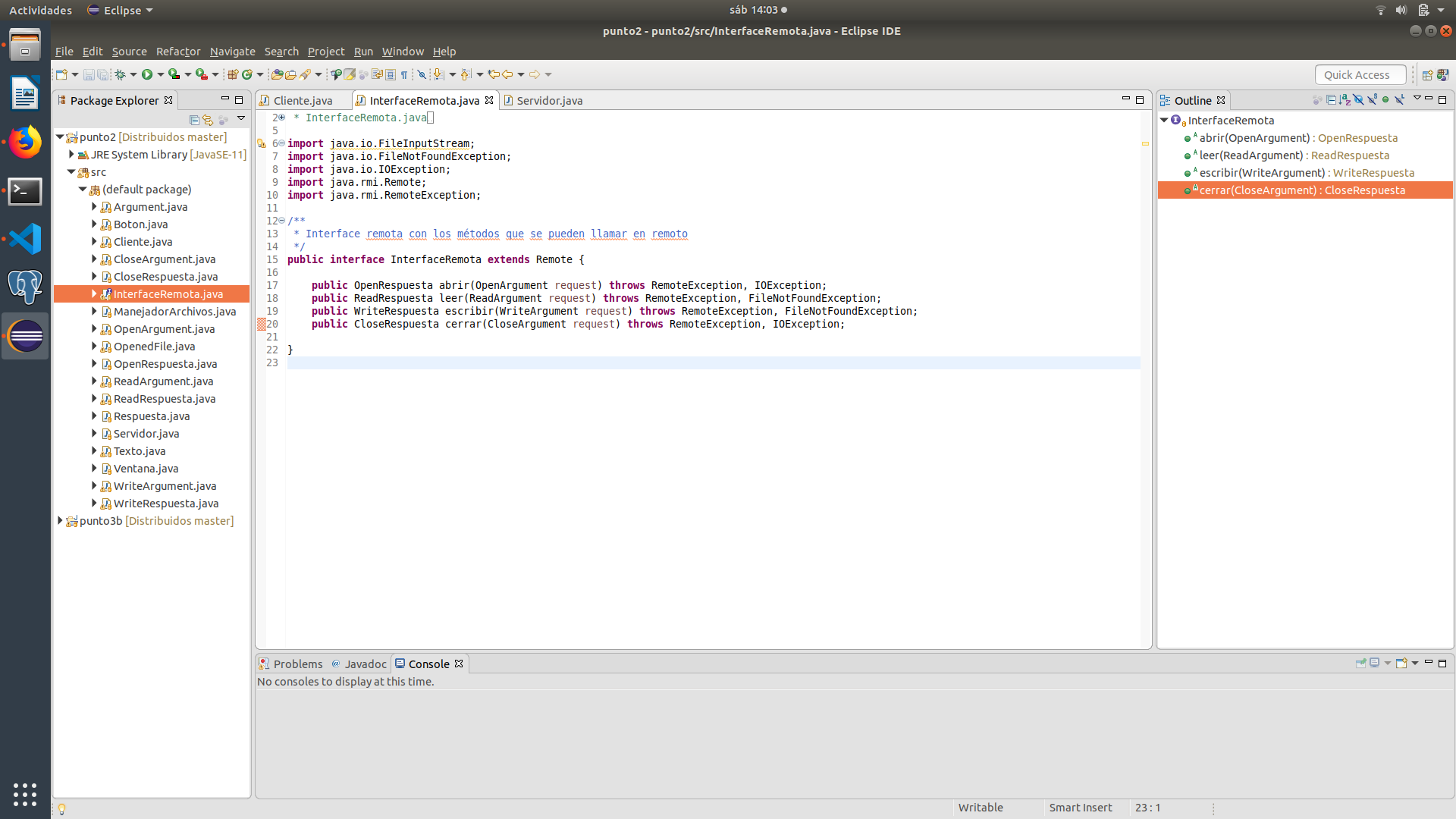


1. **Los mensajes hacia RMI Registry se valen de TCP o UDP.?**

Los mensajes que se envían hacia el RMI REgistry se valen de TCP.

**2) Implementación en código.**

1. La especificación de interfaz solamente está dada cuando se implementa el código *utilizando RMI.* En este caso particular definimos una interfaz como la siguiente:



Esta interfaz es conocida por cliente y servidor y resulta ser un acuerdo entre ambas clases conociendo cuáles métodos se pueden llamar remotamente desde el cliente.

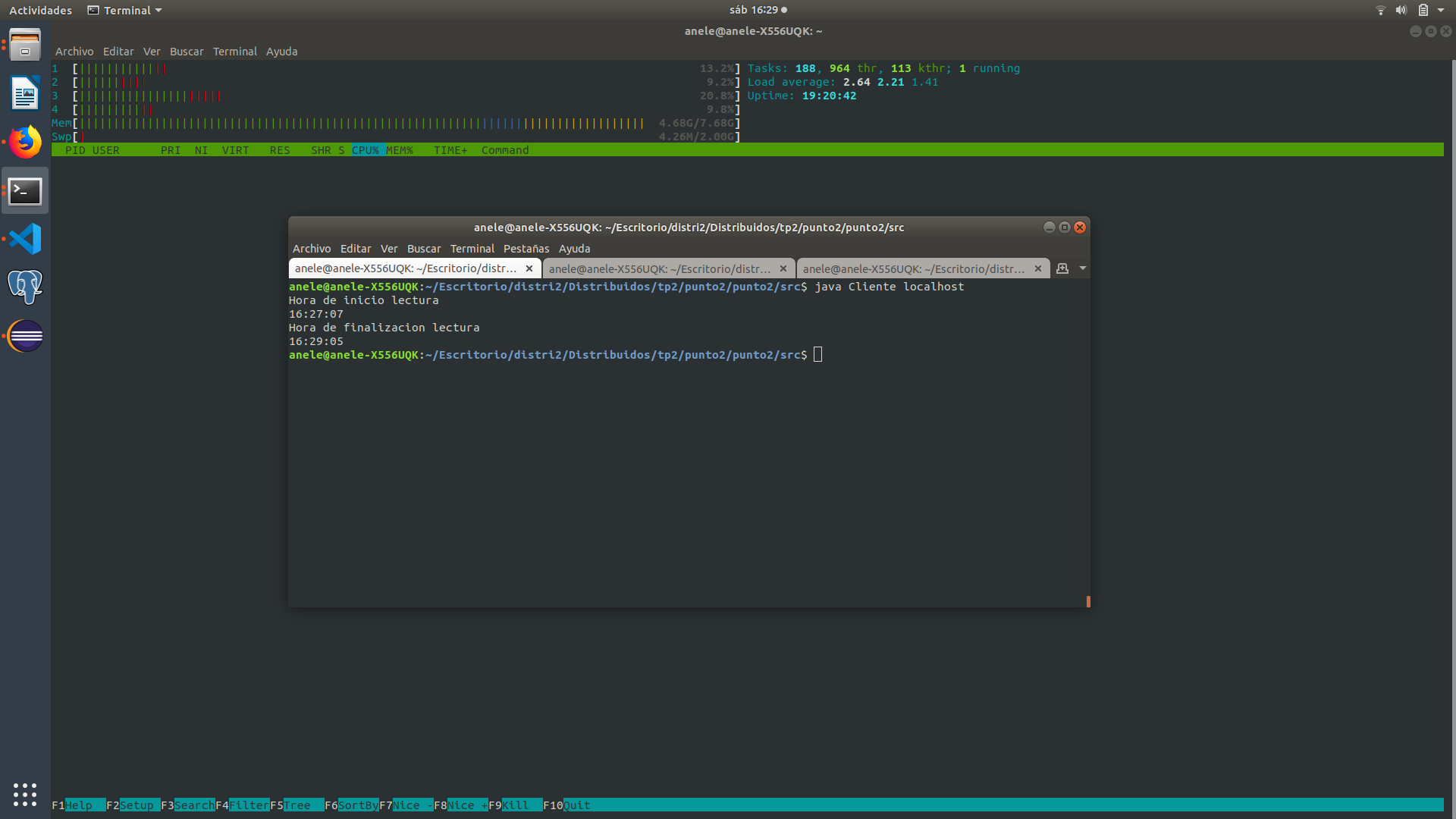
En el caso de la implementación *sin* *RMI* no se implementó una interfaz común para cliente y servidor sino que la elaboración es más “artesanal” es decir, la comunicación se realiza definiendo un socket el cual recibe y responde argumentos conocidos tanto por cliente como servidor pero el conocimiento de los métodos que puede ejecutar el servidor queda a cargo de quien elabora también el código en el cliente.

1. La implementación con Java Sockets desde nuestro punto de vista mantiene el código del cliente un poco más “limpio” en cuanto a que separa el código propio del cliente con su stub, de la misma forma en el servidor con su stub.

En el caso de la implementación con RMI gran parte del código perteneciente al stub del cliente y el stub del servidor tuvo que mudarse a las clases de Servidor y Cliente debido a que, aunque RMI abstrae la capa de comunicación entre cliente y servidor, aun existe el problema de trabajar con lecturas y escrituras en archivos, por lo que aún es necesario el trabajo que realizaban, en Java Sockets, las capas de Stub del cliente y el servidor para facilitar el manejo de archivos abiertos y posteriores lecturas, escrituras y cierres.

1. En cuanto a la transferencia de un archivo pesado (40MB) con la implementación de Java Sockets el tiempo de lectura supera ampliamente la hora de trabajo.

En este caso particular realizando la implementación con RMI la lectura del mismo archivo arroja un tiempo aproximado de 2 minutos.



3)

1. Problemas de un servidor concurrente:

* El problema es que todo cliente tiene que esperar su turno para ser atendido.
* Si uno de ellos pide un recurso muy grande los demás tienen que esperar.
* La mayor parte de la espera es debido a operaciones de IO, hay capacidad de CPU ociosa.
* Si la implementacion es con procesos es costoza su creacion y tambien su administracion.

1. Mecanismos Java RMI:

* Permite que los objetos invoquen métodos en objetos remotos utilizando la misma sintaxis que para las invocaciones locales. Además, la verificación de tipos se aplica tanto a las invocaciones remotas como a las locales.
* Java está diseñado para permitir que las clases se descarguen de una máquina virtual a otra. Esto es particularmente relevante para los objetos distribuidos que se comunican mediante invocación remota. Si el destinatario aún no posee la clase de un objeto pasado por valor, su código se descarga automáticamente. Del mismo modo, si el destinatario de una referencia a un objeto remoto no posee la clase para un proxy, su código se descarga automáticamente.  
  Esto provee dos ventajas:

1. No es necesario que cada usuario mantenga el mismo conjunto de clases en su entorno de trabajo.
2. Los programas de cliente y servidor pueden hacer un uso transparente de las instancias de nuevas clases cada vez que se agregan.

* Las implementaciones de middleware de RMI proporcionan componentes (incluidos servidores proxy, esqueletos y despachadores) que ocultan los detalles de la organización, el envío de mensajes y la localización de objetos remotos de los programadores de clientes y servidores.

1. Nosotros consideramos que el nivel de abstracción que provee RMI es mayor al nivel de abstracción que provee RPC, primero en principal porque RMI está basado en objetos y RPC en C ( la programación orientada a objetos es considerada de mayor nivel de abstracción que la programación imperativa).

También consideramos que RMI es más abstracto porque no vemos el código generado usado para la transmisión de datos (la implementación de sockets) y RPC nos genera los stubs, a los cuales si tenemos acceso y podemos ver cómo se transmite la información.

1. La principal diferencia entre la RPC y RMI es que **RMI implica *objetos***. En lugar de llamar a los procedimientos de forma remota mediante el uso de un proxy *de la función*, que en lugar de utilizar un proxy *objeto*.  
   Una similitud encontrada es que ambos proveen el servicio de abstracción en el uso de la red, es decir, ambos se encargan de la comunicación dándole sólo el trabajo de la lógica del programa al programador, dejando que este no se preocupe por cómo se transmite la información (sea con objetos o funciones).  
   Otra diferencia es que para usar RMI por la consola debemos correr el RMIRegistry; con RPC no hace falta correr algo similar.

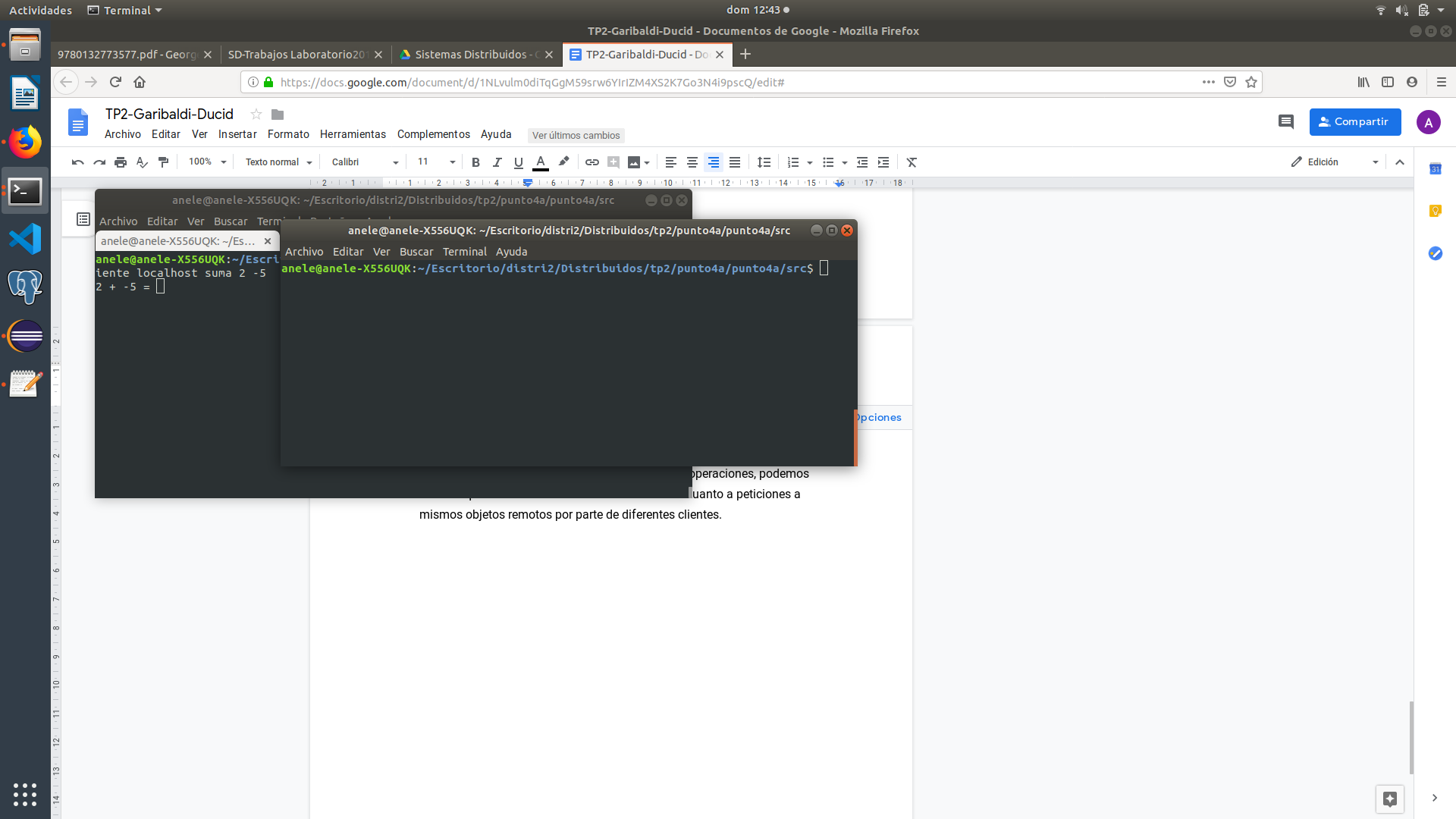
Ambos utilizan una herramienta para mapeo de puertos.

1. El servidor tiene un thread por requerimiento.

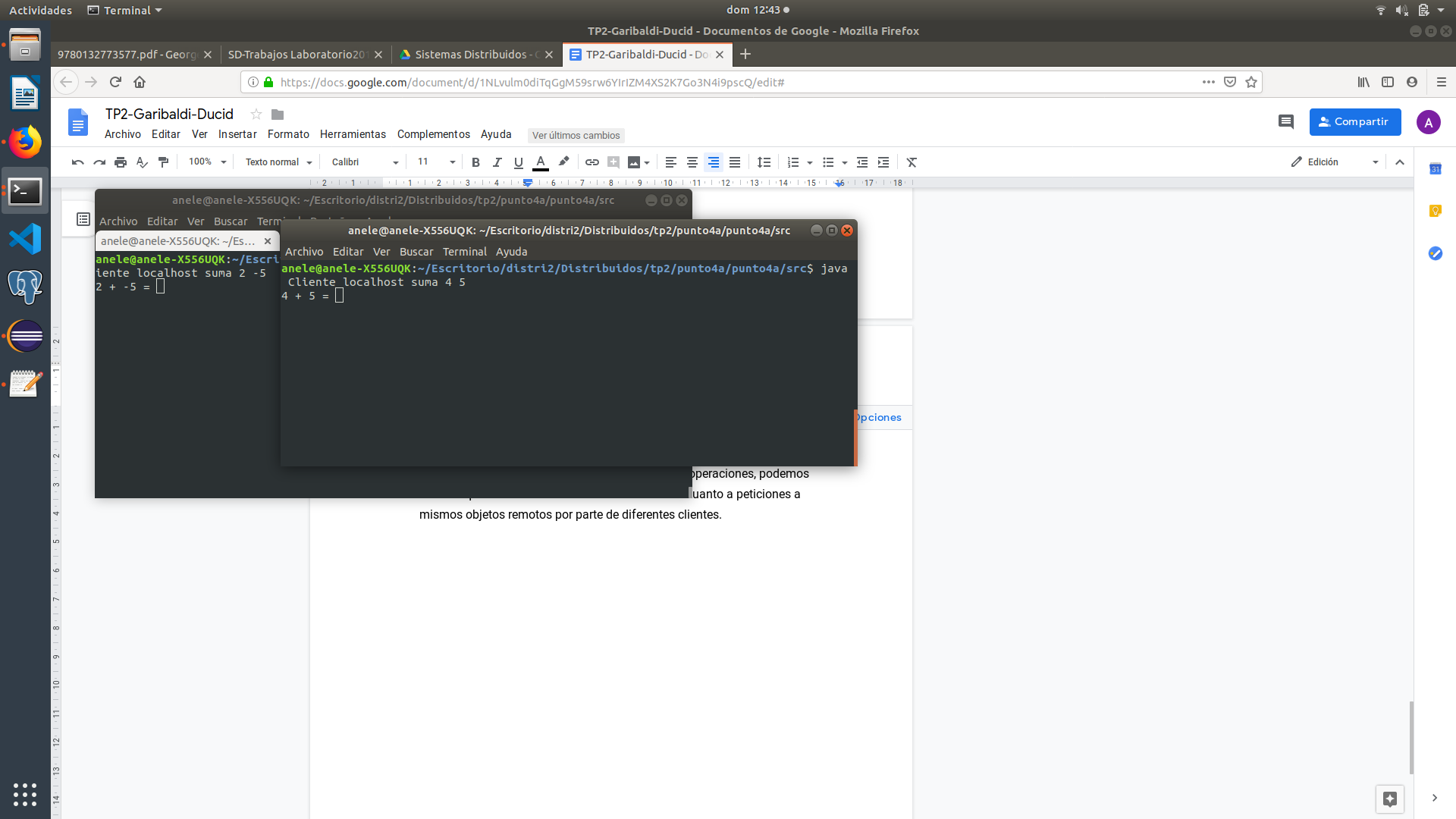
**4)**

1. **Utilizando la función TimeUnit.SECONDS.sleep(1) de Java que colocamos en los objetos remotos justo antes de retornar de sus operaciones, podemos verificar que si existe concurrencia automática en cuanto a peticiones a mismos objetos remotos por parte de diferentes clientes.**

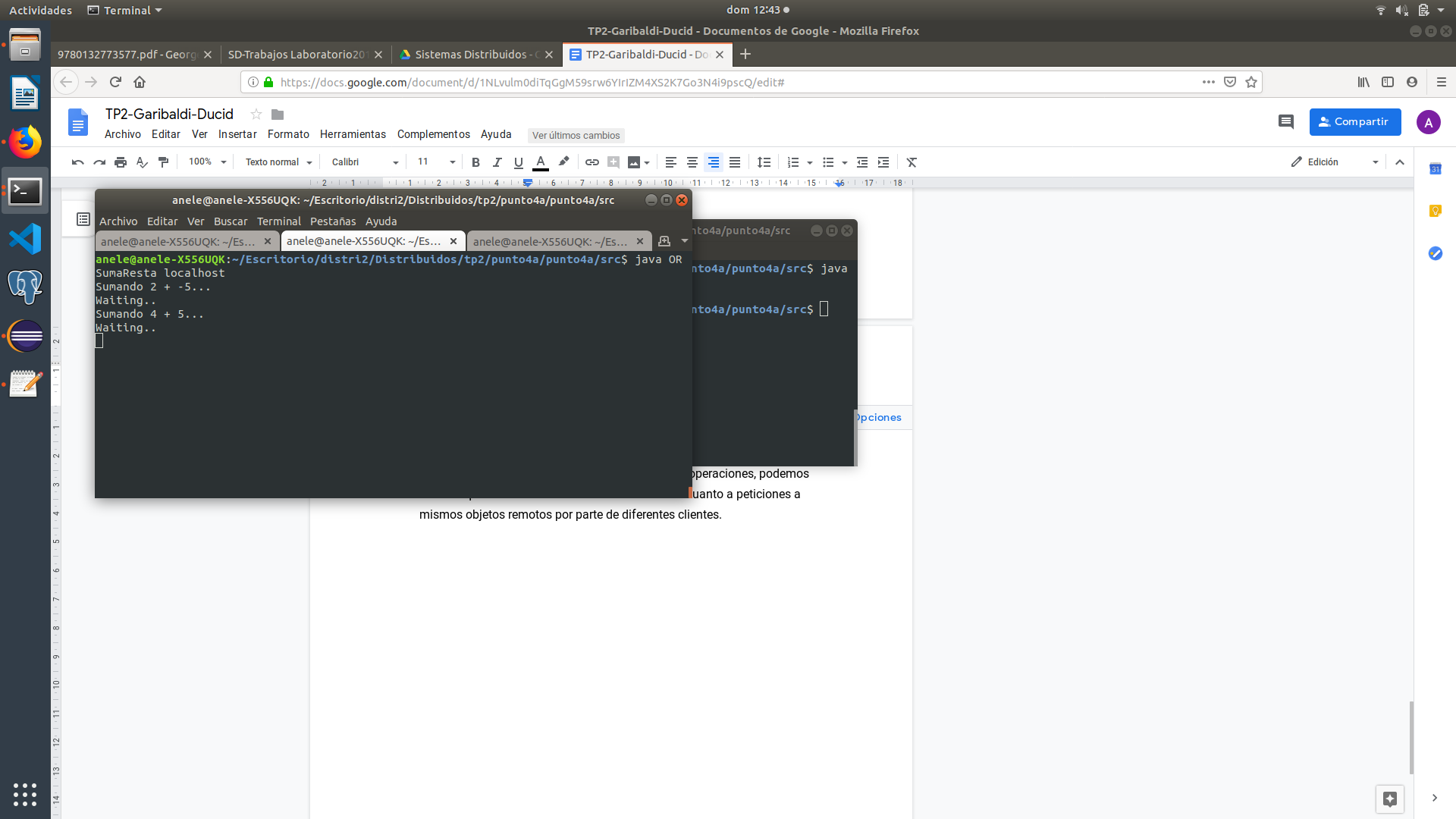
Inicialmente llamamos con un cliente:



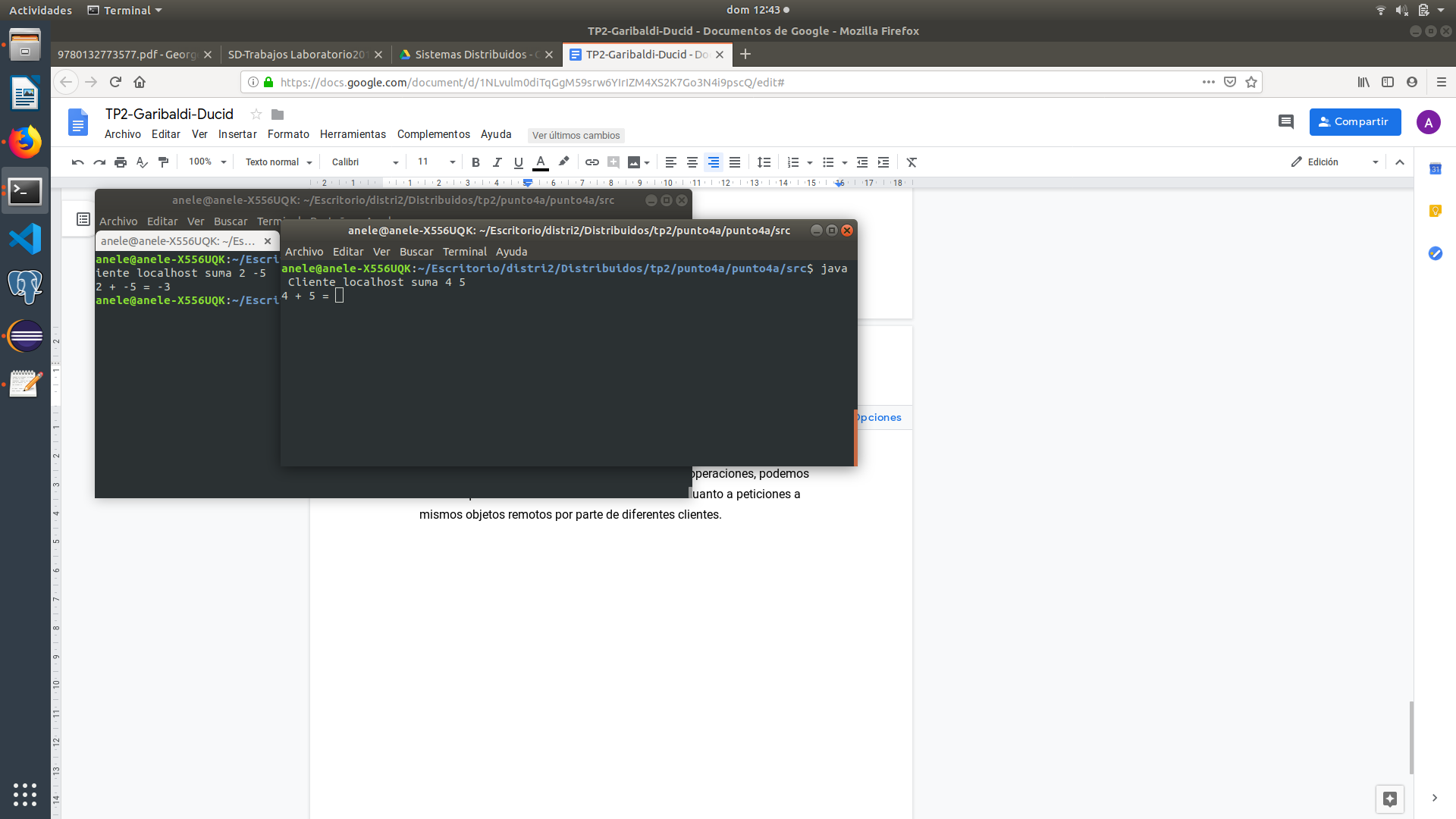
Luego de unos segundos realizamos una segunda llamada:



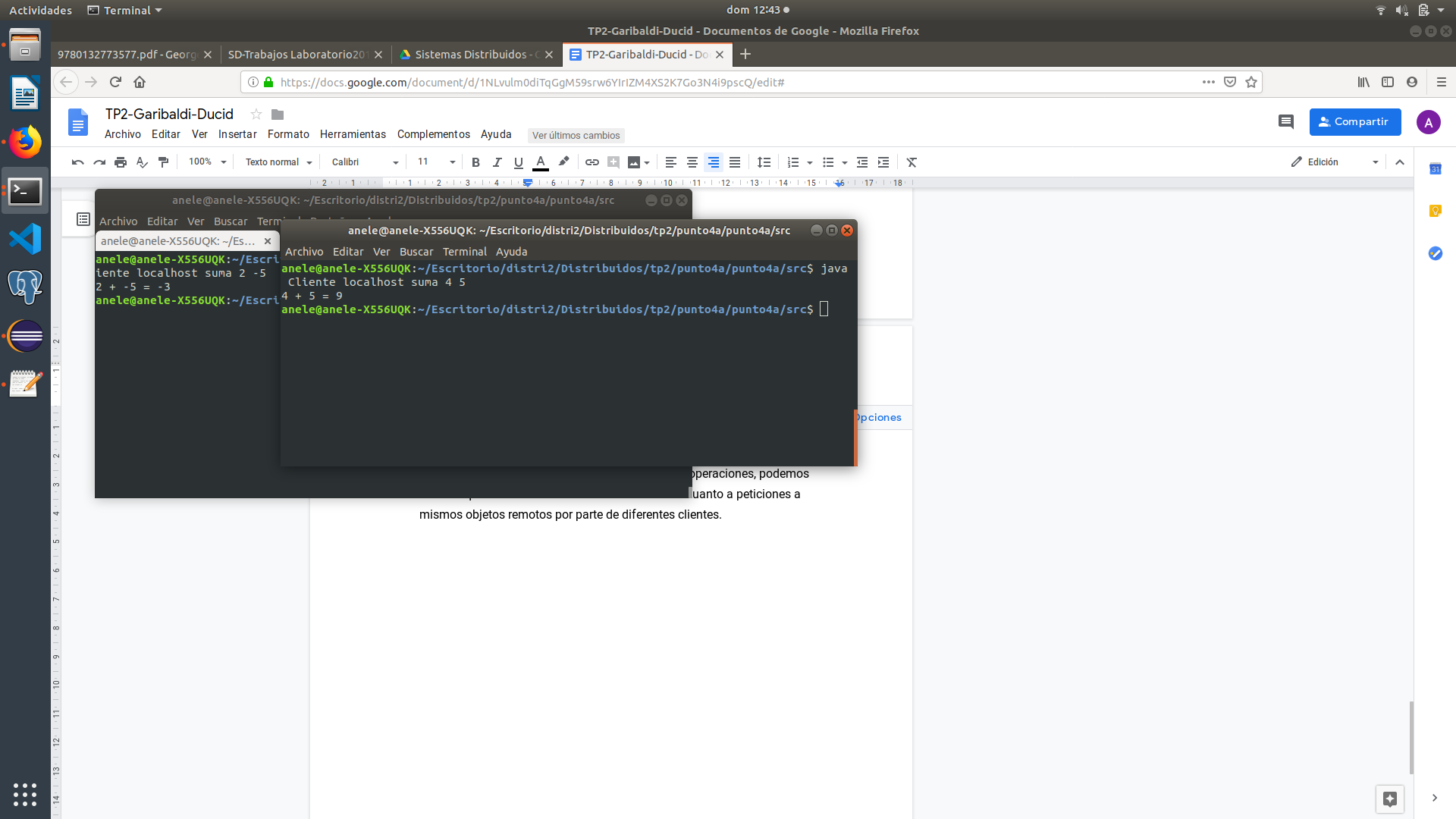
Se puede observar que el servidor recibe ambas peticiones:



Por ultimo pasados los 10 segundos de espera el servidor responde la primer petición mientras el segundo está esperando aun:

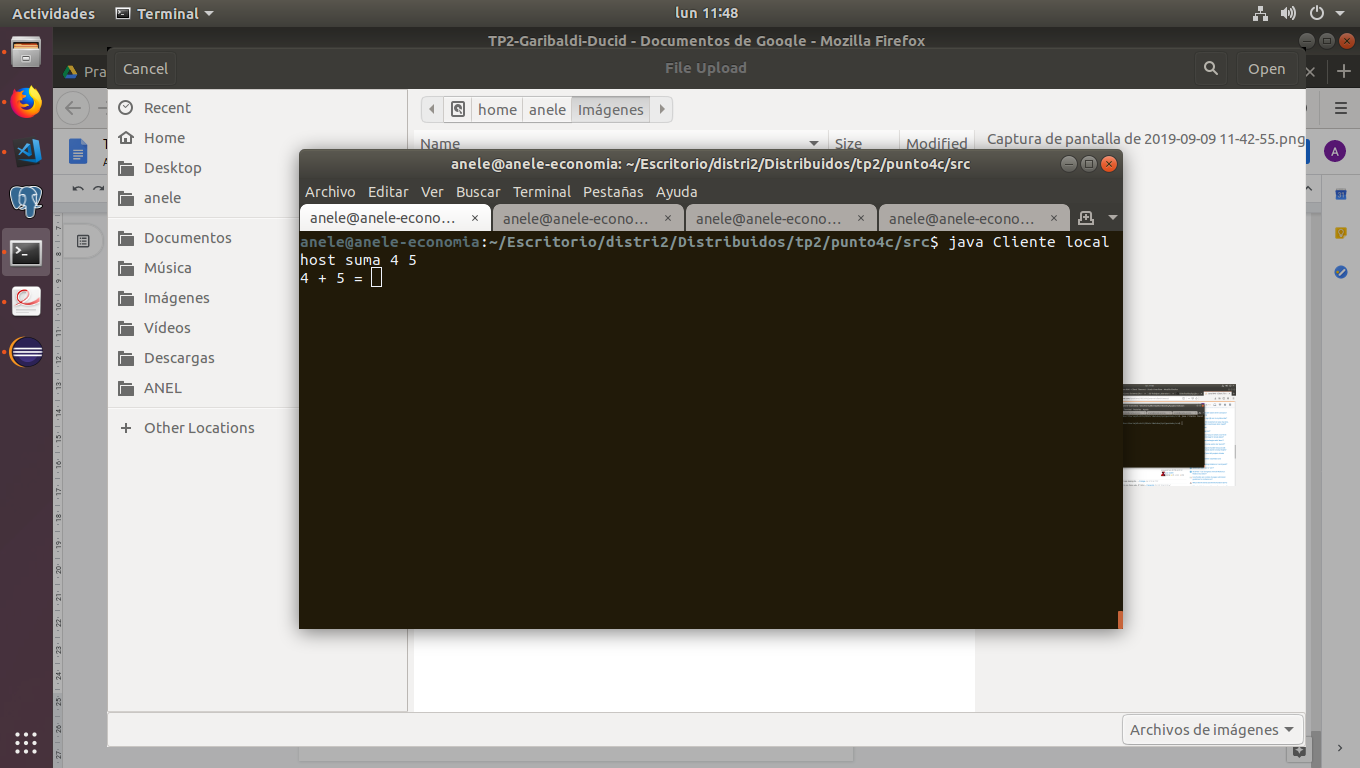


Por ultimo el segundo cliente recibe su respuesta:

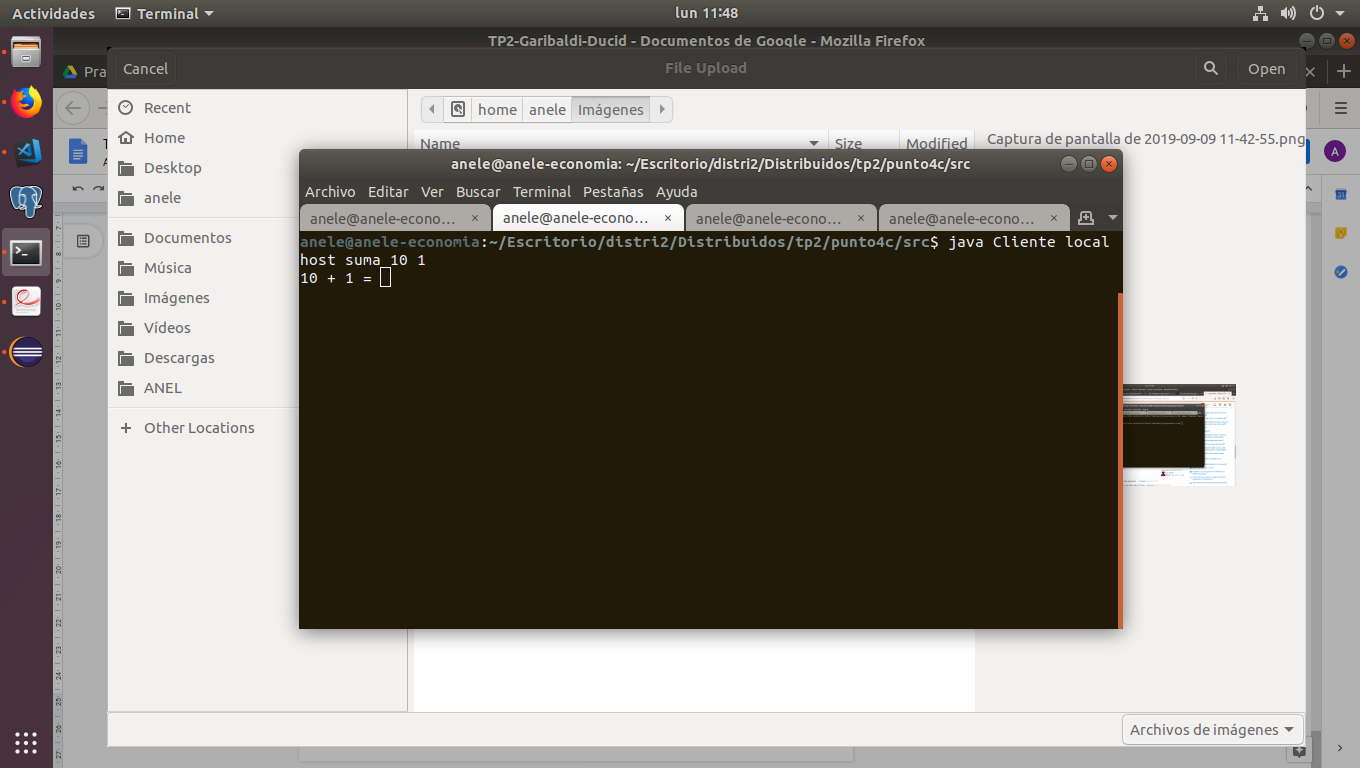


1. **Implementación en código.**
2. **En este caso verificamos que las variables de instancia de un objeto remoto si se contaminan cuando se acceden por distintos clientes en cuyo caso el único dato válido que queda guardado el el objeto es el del cliente que accede “ultimo” a esa variable.**

Primero accedemos desde un cliente utilizando la suma (valiéndonos de un delay en el objeto remoto)



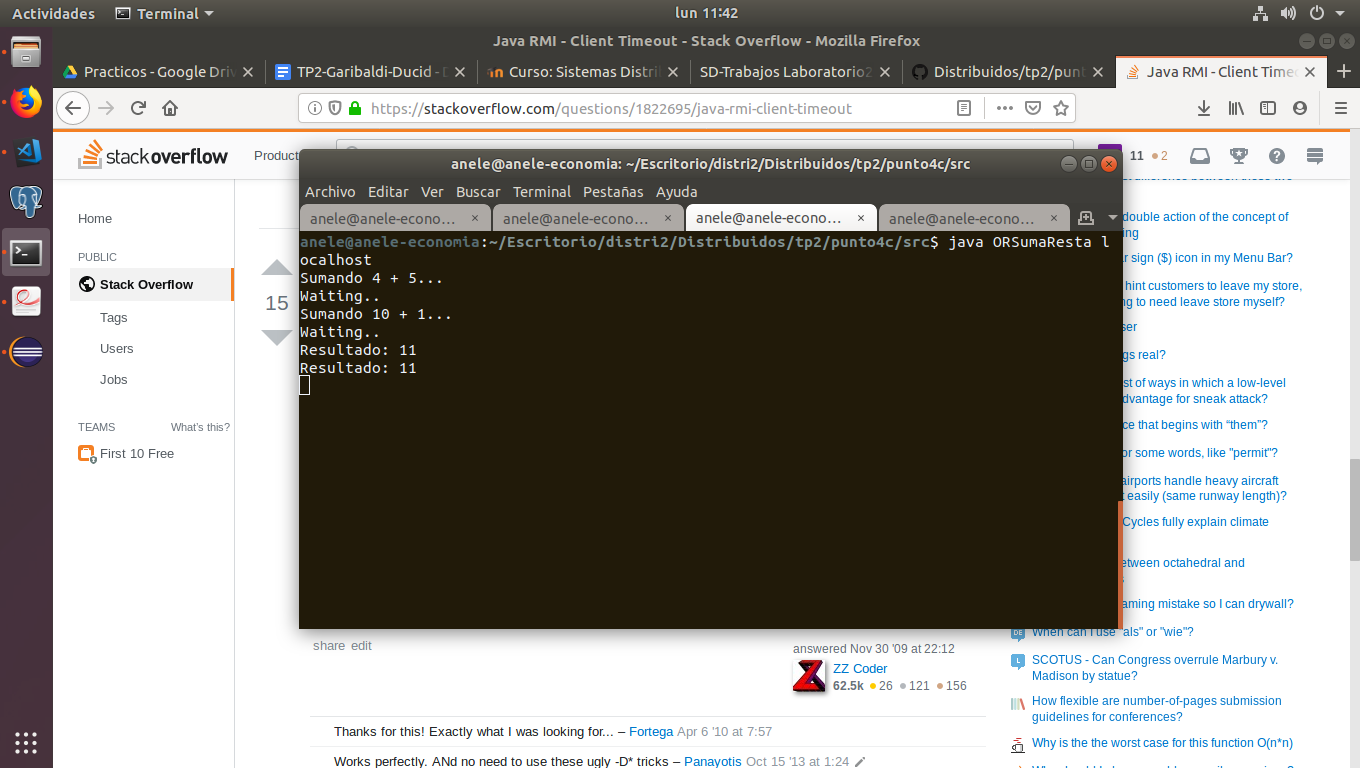
Luego instanciamos otro cliente y realizamos una suma:



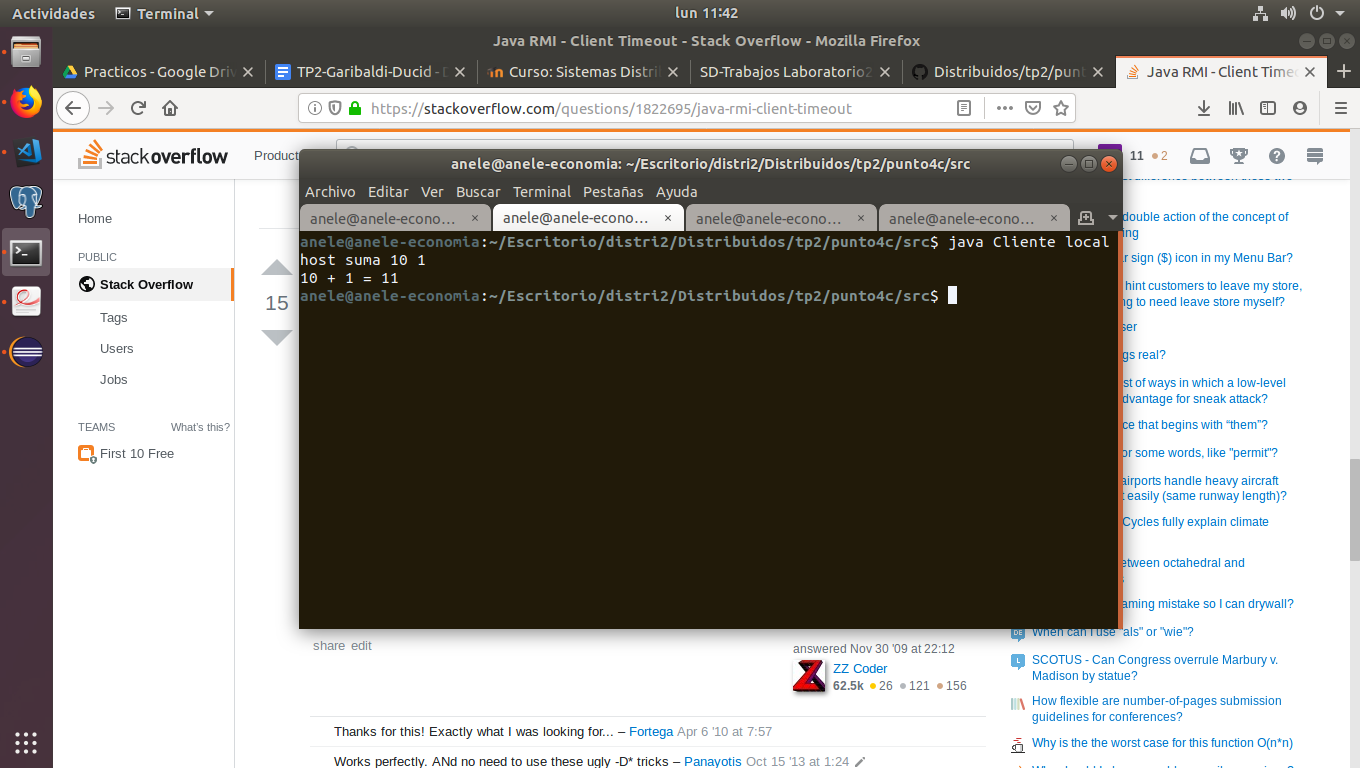
En el servidor podemos observar, gracias al delay, que llegan ambas peticiones. Cuando el servidor realiza la operación almacena el resultado de la misma en una variable de instancia llamada *resultado.*

Si bien el resultado de la primer operación (4+5) es 9 el servidor luego de calcularla la almacena en su variable de instancia *resultado* y queda en estado Waiting.

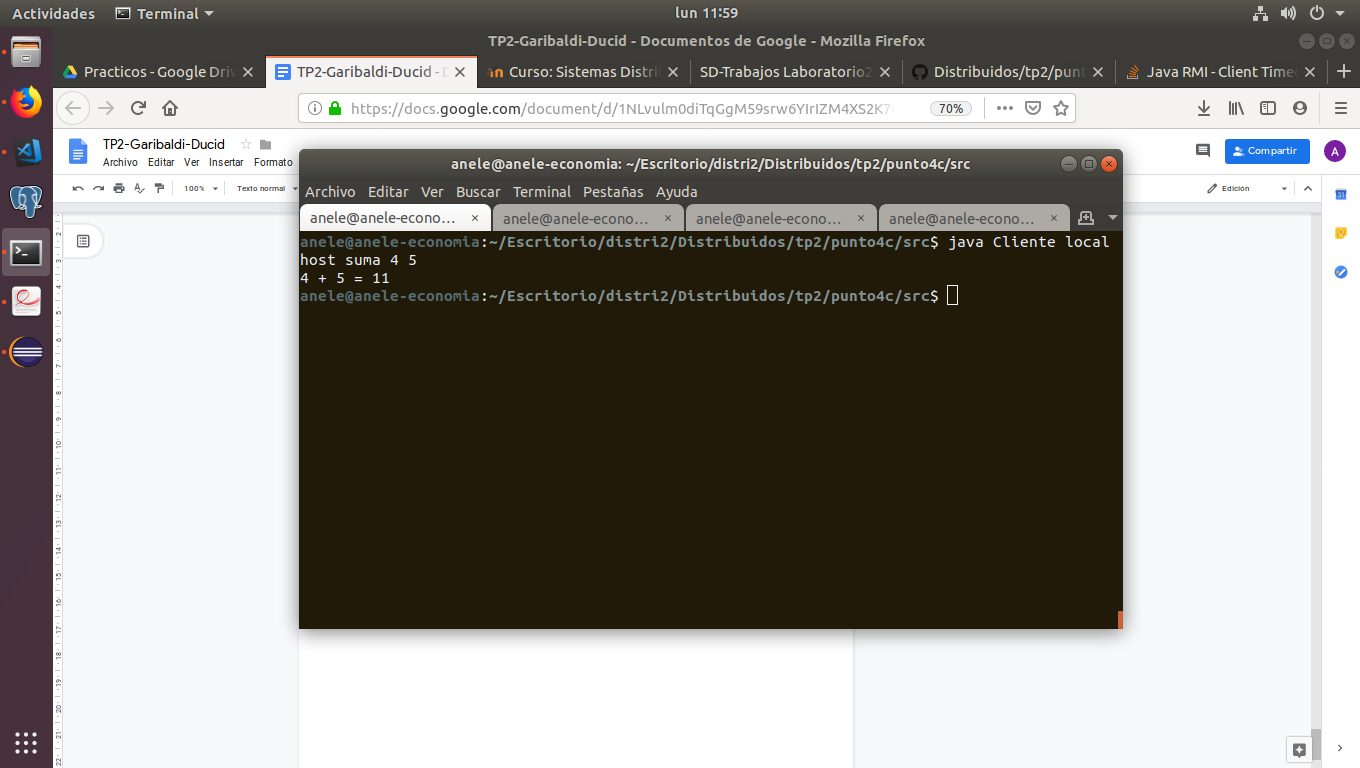
Luego en la segunda petición se realiza el cálculo (10+1) que da resultado 11 y este se vuelve a almacenar en la misma variable *resultado* sobreescribiendo el valor anterior (9).



Si observamos el segundo cliente obtiene un resultado correcto:



Pero el cliente que realiza la primer petición recibe un resultado de operación incorrecto:



Esto se debe a que ambos clientes alteran la misma variable de instancia del servidor y el único resultado almacenado es el último escrito (en este caso el resultado de la suma del segundo cliente).

**5) Implementación en código.**

**6) Implementación en código.**

**7) Para cada afirmación determine si es verdadera o falsa:**

**a) En los criptosistemas simétricos no puede garantizarse el no repudio porque ambas partes de la transacción conocen la clave utilizada.**

**Falso**; Si, puede garantizarse el no repudio en un criptosistema simétrico, ya que ser simétrico implica que tanto el emisor y receptor comparten una clave privada y para respetar el principio de no repudio se deben cumplir dos reglas, la primera es el “no repudio de origen”, que significa que el emisor no puede negar el envió porque el receptor tiene pruebas del mismo. La segunda regla es el “no repudio de destino”, que como su nombre indica el receptor no puede negar que recibió el mensaje porque el emisor tiene pruebas de la recepción. Estos dos conceptos no se contraponen.

**b) Si únicamente me importara la eficiencia del método que uso para encriptar, debería optar por un algoritmo de cifrado asimétrico.**

**Falso**; si el requerimiento principal del sistema distribuido es la eficiencia lo más conveniente es usar un método de encriptación simétrico. Estos métodos no aumentan el tamaño del mensaje y proveen una gran velocidad.

**c) Con ambos tipos de criptosistemas necesito contar con un mecanismo seguro para transmitir la clave.**

**Falso**; en el caso de usar un criptosistema simétrico, la clave privada usada por el emisor y receptor, debe ser comunicada entre ellos de manera segura, por ejemplo juntándose en persona y pasandosela.

En el caso de usar un criptosistema asimétrico, no es necesario efectuar ningún intercambio de claves.

**8)**

**a) ¿Qué información es necesaria para que quien recibe un mail firmado, pueda verificar la firma del mismo?**

Para que el receptor del mensaje pueda verificar la firma del mismo se debe tener la firma a verificar y el certificado de autenticación, éstos se envían junto con los datos (en este caso el mail) por medio de un mensaje desde el emisor hacia el receptor.

**b) ¿Qué información necesito para poder enviar un mail encriptado?**

Para poder enviar un mail encriptado se necesita el certificado con la clave pública del receptor.

En caso de usar un método de encriptamiento simétrico se debe conocer la clave privada usada para poder encriptar el mail.

En caso de usar un método de encriptamiento asimétrico basta con conocer la clave pública del receptor. Se usa esta clave pública para encriptar el mensaje y solo puede desencriptarla quien posea la clave privada, es decir, solo el receptor a quien se le desea enviar el mensaje.

**c) ¿Qué información es necesaria para que quien recibe un mail encriptado, pueda abrirlo?**

En caso de usar un método de encriptación simétrico se debe conocer la clave privada usada en la comunicación para poder desencriptar el mail.

En caso de usar un método de encriptación asimétrico depende de cómo se encripto el mensaje, si se encripto con la clave pública del receptor se debe desencriptar el mensaje con la clave privada del receptor. En caso de que se haya encriptado el mensaje con la clave privada del receptor, se debe desencriptar con la clave pública del receptor.

**9) Evalúe las siguientes situaciones en el marco de una infraestructura de PKI:**

**Su clave privada fue comprometida, entonces:**

**a) ¿Qué consecuencias sufro respecto de la información firmada con la clave privada asociada a mi certificado?**

Si mi clave privada ha sido comprometida debe reportarla a la autoridad certificante lo antes posible, en caso contrario puede suceder que alguien utilice mi clave sustituyendo mi identidad y firmando documentos por mi. En este caso las personas que crean recibir mensajes auténticos firmados por mi, no se darían cuenta que en realidad mi clave privada está siendo manipulada a menos que la autoridad certificante notifique que esa clave está en desuso y no me pertenece más.

**b) ¿Qué consecuencias sufro respecto de la información encriptada para que solo usted pueda ver?**

La información que es encriptada con intenciones de garantizar confidencialidad, es encriptada con la clave pública, por lo tanto, al tener comprometida nuestra clave privada, cualquier persona que la posea puede leer el mensaje y estaría viéndose comprometida la confidencialidad.

**c) ¿Qué acciones se pueden llevar a cabo? ¿cómo?**

La principal acción a tomar en cuenta es informar a la autoridad certificante que esa clave ha sido comprometida y que en caso que alguien quiera firmar algún documento con ella, ya no posea validez alguna. Además es posible solicitar un nuevo par de claves.

10)

a)

En un contexto de transacciones anidadas, la transacción de mayor nivel (la que está más cerca de la raíz viéndolo como un árbol de transacciones) se llama transacción de nivel superior, mientras que las internas se llaman sub transacciones.

Cuando una sub transacción se completa, toma una decisión sobre si comprometer provisionalmente (no graba nada en disco), o abortar. Cuando todas las sub transacciones son completadas, todas las que hayan comprometido provisionalmente llevan a cabo el protocolo de compromiso en dos fases expresando su intención de comprometer la transacción y preparándose para comprometer, si es que es el resultado del compromiso en dos fases. Todas las sub transacciones cuyos ancestros (transacciones superiores) hayan abortado, serán abortadas también.

Cabe destacar que si una transacción superior fue abortada, todas sus sub transacciones serán forzadas a abortar también. Sin embargo, la recíproca no es cierta; si una sub transacción fue abortada, no significa que su padre ni sus sub transacciones hermanas sean forzadas a abortar. Si una sub transacción fue

comprometida provisionalmente, debe ser abortada si su ancestro fue abortado, sin importar que luego su ancestro pueda comprometer exitosamente.

La transacción superior cumple el rol de coordinador en el protocolo de compromiso en dos fases, siendo los otros integrantes las sub transacciones que comprometieron previsionalmente y no tienen ancestros abortados. Si las sub transacciones deciden comprometer (final de la primer fase), entonces deben preparar sus transacciones almacenando su estado en disco.

La segunda fase consiste en que el coordinador recupera las decisiones de los integrantes y les indica la decisión final. Cuando se completó el procedimiento, el coordinador y los participantes habrán comprometido o abortado sus transacciones consistentemente \autocite{ColourisTwoPhaseCommitProtocol}