Programación Distribuida y Tiempo Real

Práctica 3

1) Utilizando como base el programa ejemplo de RMI:
a.- Analice si RMI es de acceso completamente transparente (access transparency, tal como está definido en Coulouris-Dollimore-Kindberg). Justifique.

RMI es de acceso completamente transparente, ya que luego de la creación del objeto en el cliente, es posible acceder al método remoto como si fuera local. De esta forma es posible por ejemplo utilizar polimorfismo para utilizar el mismo método de forma local y remota según el objeto utilizado.

b.- Enumere los archivos .class que deberían estar del lado del cliente y del lado del servidor y que contiene cada uno.

Servidor
- RemoteClass.java
    - Implementa las declaraciones de la interface (IfaceRemoteClass)
    - Contiene la implementación de los métodos remotos.
    - Posee declaraciones propias de RMI.
    - Extiende UnicastRemoteObject.
- StartRemoteObject.java
    - Crea el objeto remoto y lo registra de forma que pueda ser accedido por otros objetos.
- IfaceRemoteClass.java
    - Interfaz con las definiciones de los métodos remotos.
    - Realiza las importaciones necesarias para definir métodos remotos.
    - extiende Remote.

Cliente
- AskRemote.java
    - Obtiene una referencia al objeto remoto
    - Castea el objeto remoto con la interfaz.
    - realiza la invocación de los métodos remotos.
- IfaceRemoteClass.java
    - Interfaz con las definiciones de los métodos remotos.
    - Realiza las importaciones necesarias para definir métodos remotos.
    - extiende Remote.
    - Se castea el objeto remoto a esta clase y de esta forma es posible llamar sus métodos.

2) Identifique similitudes y diferencias entre RPC y RMI.

RMI es exclusivo de Java y orientado a objetos, en cambio RPC se puede utilizar en múltiples lenguajes.
En RPC se llama las funciones remotas definidas en el servidor directamente, en cambio, con RMI es necesario tener una referencia al objeto remoto y se le envían a este objetos los mensajes.
Al ser orientado a objetos, RMI obtiene toda la expresividad extra de este.
En RMI, todos los objetos tienen referencias únicas, las cuales pueden ser pasadas por medio de parámetros.
Tanto RPC como RMI permiten programar con interfaces.
Ambos suelen programarse sobre protocolos de request-reply y pueden ofrecer semanticas de llamadas como at-least-once o at-most-once.
Ambos emplean un nivel similar de transparencia, la sintaxis local y remota es la misma, aunque las interfaces remotas suelen exponer la naturaleza distribuida de la operacion, por ejemplo, soportando excepciones remotas.

3) Investigue porque con RMI puede generarse el problema de desconocimiento de clases en las JVM e investigue como se resuelve este problema.

Este problema puede ocurrir debido a que el registro RMI no puede encontrar de donde cargar las clases. Es necesario utilizar el comando rmiregisrty & sobre el directorio que albergue las clases para realizar el registro del objeto remoto y conocer que path utilizar para encontrar las clases.

4) Implementar con RMI el mismo sistema de archivos remoto implementado con RPC en la práctica anterior:
a.- Defina e implemente con RMI un servidor cuyo funcionamiento permita llevar a cabo las operaciones desde un cliente enunciadas informalmente como (definiciones copiadas aquí de la práctica anterior):

- **leer: dado un nombre de archivo, una posición y una cantidad de bytes a leer, retorna 1)la cantidad de bytes del archivo pedida a partir de la posición dada o en caso de haber menos bytes, se retornan los bytes que haya y 2) la cantidad de bytes que efectivamente se retornan leıdos.**
- **escribir: dado un nombre de archivo, una cantidad de bytes determinada, y un buffer a partir del cual están los datos, se escriben los datos en el archivo dado. Si el archivo existe, los datos se agregan al final, si el archivo no existe, se crea y se le escriben los datos. En todos los casos se retorna la cantidad de bytes escritos.**

Se crearon las clases IfaceRemoteClass, RemoteClass y StartRemoteObject de la siguiente forma:

- IfaceRemoteClass: se le pusieron los metodos read(String) y write(String) que reciben y devuelven Strings.
- RemoteClass: se implementan los metodos read y write:
    - read(String): primero decodifica el string recibido como JSON y luego utiliza una instancia de RandomAccessFile para lograr un acceso rapido a los datos

del archivo con un offset y una cantidad de bytes a leer recibidas en el JSON. Una vez leido retorna la cantidad de datos leídos y los datos leidos en un string con formato JSON.
    - write(String): primero decodifica el string recibido como JSON y luego escribe utilizando los datos recibidos por parametro en un archivo local mediante Files.write(). Finalmente devuelve un string en formato JSON con la cantidad de datos escritos, que sera igual a la cantidad de datos que posea el vector de bytes pasados a Files.write().
- StartRemoteObject: inicia el objeto remoto.

b.- Implemente un cliente RMI del servidor anterior que copie un archivo del sistema de archivos del servidor en el sistema de archivos local y genere una copia del mismo archivo en el sistema de archivos del servidor. En todos los casos se deben usar las operaciones de lectura y escritura del servidor definidas en el item anterior, sin cambios específicos del servidor para este item en particular. Al finalizar la ejecución del cliente deben quedar tres archivos en total: el original en el lado del servidor, una copia del original en el lado del cliente y una copia en el servidor del archivo original. El comando diff no debe identificar ninguna diferencia entre ningún par de estos tres archivos5) Investigue si es posible que varias invocaciones remotas estén ejecutándose concurrentemente y si esto es apropiado o no para el servidor de archivos del ejercicio anterior. En caso de que no sea apropiado, analice si es posible proveer una solución (enunciar/describir una solución, no es necesario implementarla). Nota: diseñe un experimento con el que se pueda demostrar fehacientemente que dos o más invocaciones remotas se ejecutan concurrentemente o no. Compare este comportamiento con lo que sucede con RPC.

Se creó la clase AskRemote de la siguiente forma:
- Se crea la referencia al objeto remoto.
- Se crea un String en formato JSON que indicará el archivo a leer.
- Se envia el método read al objeto remoto con los datos del JSON creado en el punto anterior.
- Se decodifica el JSON recibido del método remoto y se crea la copia local del archivo.
- Finalmente se codifican en formato JSON los datos de la copia que será creada en el servidor y se envía el método remoto write con este JSON.

6) Tiempos de respuesta de una invocación:
a.- Diseñe un experimento que muestre el tiempo de respuesta mínimo de una invocación con JAVA RMI. Muestre promedio y desviación estándar de tiempo respuesta.

Se creó un procedimiento remoto que no recibe nada, no hace nada y retorna nada.
En el cliente se toma el tiempo y se guarda en un csv que luego fue procesado usando la librería pandas.

Media:

645251.0

Desviación estándar:
110118.439043

b.- Investigue los timeouts relacionados con RMI. Como mínimo, verifique si existe un timeout predefinido. Si existe, indique de cuanto es el tiempo y si podría cambiarlo. Si no existe, proponga alguna forma de evitar que el cliente quede esperando indefinidamente.

Para la librería java.rmi es posible setear un timeout mediante RMISocketFactory.setSocketFactory seteando un timeout al socket que será utilizado en el RMI cuando este es creado (el socket).

Existen varias propiedades relacionadas con el timeout utilizadas en Java RMI para microsistemas Sun, las cuales podrán ser seteadas por System.setProperty():

- Properties that can be set on rmid

sun.rmi.activation.execTimeout (1.2 and later)
El valor de esta propiedad representa en milisegundos el tiempo que el sistema esperara por el inicio de un grupo de activacion. Default: 30 segundos.

sun.rmi.activation.groupTimeout (5.0 and later)
El valor de esta propiedad representa en milisegundos el tiempo que rmid esperara luego de destruir el proceso de un grupo de activacion antes de crear una nueva encarnacion del grupo de activacion. Default: 60 segundos.

- Properties that are useful to set on VMs that export remote objects

sun.rmi.dgc.ackTimeout (1.4 and later)
El valor de esta propiedad representa el tiempo en milisegundos que el lado del servidor Java RIM corriendo referenciará fuertemente un objeto remoto que fue retornado desde la máquina virtual actual como parte de un llamado a método remoto, hasta que reciba los acknowledgment del cliente de que esa referencia remota fue correctamente recibida y procesada. El timeout se aplica sólo a situaciones de fallo (en las que el cliente falla en enviar el acknowledgment). Defecto: 5 minutos.

sun.rmi.transport.tcp.localHostNameTimeOut (1.1.7 and later)
El valor de esta propiedad representa el tiempo en milisegundos en el que el Java RMI runtime esperará obtener un nombre de dominio para el host local completamente calificado. Defecto: 10 segundos.

sun.rmi.transport.tcp.readTimeout (1.2.2 and later)

El valor de esta propiedad representa en milisegundos el tiempo usado como un timeout para las conecciones TCP entrantes que la Java RMI runtime usa de forma que finalicen las conexiones que el cliente no utiliza y no finalizó. Defecto: 2 horas.

- Properties that are useful to set on VMs that make remote method calls

sun.rmi.transport.connectionTimeout (1.1.6 and later)
El valor de esta propiedad representa el periodo en milisegundos que una conexion a un socket puede perdurar en un estado “unused” (sin usar) antes que el Java RMI runtime permita que esas conexiones sean cerradas. Defecto: 15 segundos.

sun.rmi.transport.proxy.connectTimeout (1.1 and later)
El valor de esta propiedad representa el maximo tiempo en milisegundos que la Java RMI runtime esperara por el completado de un intento de conexion antes de intentar contactar el servidor utilizando HTTP. Esta propiedad solo sera utilizado cuando http.proxyHost este seteado y java.rmi.server.disableHttp sea falso. Defecto: 15 segundos.

sun.rmi.transport.tcp.handshakeTimeout (1.4 and later)
El valor de esta propiedad representa el tiempo en milisegundos que el Java RMI runtime esperara antes de decidir que una conexión TCP aceptada por un servidor remoto no puede ser usada. Default: 1 minuto.

sun.rmi.transport.tcp.responseTimeout (1.4 and later)
El valor de esta propiedad representa el tiempo en milisegundos que el lado del cliente de Java RMI runtime utilizara como timeout de socket de lectura en una conexion JRMP ya establecida cuando lee la respuesta de la invocacion a un metodo remoto. Esta propiedad es usada para imponer un timeout a la espera de un resultado en el servidor remoto. Default: 0 (sin timeout).