

Instituto Politécnico Nacional

Escuela Superior de Cómputo

*Aplicaciones para comunicaciones de red*

*“Practica 7 – Buscador de archivos distribuido”*

**Grupo:** 3CM15

**Integrantes:**

* Ramírez Olvera Guillermo.
* Sánchez Méndez Edmundo Josue.

**Fecha de entrega:** 18 de junio de 2021

**Profesor:** Moreno Cervantes Axel Ernesto



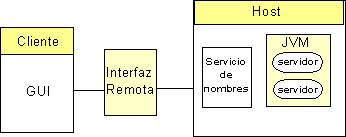
Para el desarrollo de esta última practica se hace uso de 3 diferentes servidores con diferentes propósitos, los servidores son RMI, Multicast y unicast, con diferentes propósitos que veremos en el desarrollo, de esta práctica, pero antes de empezar necesitamos saber que es RMI y en especial para que nos sirve el MD5Checksum.

**RMI**

RMI (*Remote Method Invocation*) es un mecanismo que permite realizar llamadas a métodos de objetos remotos situados en distintas (o la misma) máquinas virtuales Java, compartiendo así recursos y carga de procesamiento a través de varios sistemas.

RMI permite exportar objetos como objetos remotos para que otro proceso remoto pueda acceder directamente como un objeto Java. Todos los objetos de una aplicación distribuida basada en RMI deben ser implementados en Java. Esta es una de las principales ventajas de RMI, ya que RMI forma parte del API de Java, con lo que la integración de objetos remotos en aplicaciones distribuidas se realiza sin necesidad de usar recursos adicionales (como por ejemplo un lenguaje de descripción de interfaces o IDL). De hecho, se utiliza la misma sintaxis para una llamada a un objeto remoto o un objeto local.

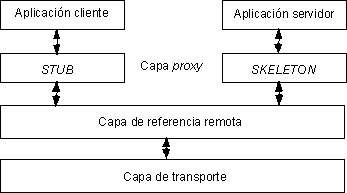
La siguiente figura se ilustra los componentes implicados en un sistema RMI: un interfaz remoto, un cliente, y uno o más servidores (objetos remotos) residentes en un host.

*.*

El cliente invoca a los objetos remotos mediante la interfaz remota. Un **servicio de nombres** (registro RMI) reside en el *host* proporcionando el mecanismo que el cliente usa para encontrar uno más servidores iniciales RMI.

La interacción con el objeto remoto se lleva a cabo a través de la **interfaz remota**. Esencialmente, ésta describe los métodos que pueden ser invocados de forma remota, y que el objeto remoto implementa. Cuando se obtiene una referencia a un objeto remoto, el objeto no se envía a través de la red al cliente que lo solicita. En su lugar se genera un objeto *proxy* o *stub* que constituye el *proxy* de la parte del cliente del objeto remoto. Todas las interacciones del cliente se realizarán con esta clase *stub*, la cual es responsable de gestionar los datos entre el sistema local y el remoto. Muchos clientes pueden tener referencias a un único objeto remoto. Cada cliente tiene su propio objeto *stub* que representa al objeto remoto, pero dicho objeto remoto NO se replica.

En la parte del servidor, una clase *skeleton* es la responsable de gestionar las llamadas al método y los datos enviados al objeto real referenciado. Éste es el *proxy* de la parte del servidor para el objeto remoto. El sistema completo puede verse como un modelo de cuatro capas, tal y como se ilustra en la siguiente figura.

*.*

La primera capa es la de **aplicación**, y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una interfaz que extienda *java.rmi.Remote*. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete *java.rmi.server*), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

La capa 2 es la capa ***proxy***, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones sobre sus parámetros y retorno de objetos tienen lugar en esta capa.

La capa 3 es la de **referencia remota**, es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo *stream* (*stream-oriented* *connection*) desde la capa de transporte.

La capa 4 es la de **transporte**. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (*Java Remote Method Protocol*), que solamente es "comprendido" por programas Java.

**MD5Checksum**

MD5 (Message Digest Algorithm 5) es un algoritmo que se utiliza como una función de codificación o huella digital de un archivo. De esta forma, a la hora de descargar un determinado archivo como puede ser un instalador, el código generado por el algoritmo, también llamado hash, viene “unido” al archivo. Un hash MD5 está compuesto por 32 caracteres hexadecimales y una codificación de 128 bits.

El código MD5, existe un software que analiza el archivo descargado y con el hash de la descarga, acudimos a la web del desarrollador del programa que queremos instalar y buscamos el código original, comprobando si coincide el que nos ha dado el hash con el que aparece en la web. De esta forma veremos si el archivo es fiable o no.

**Usos del cifrado MD5**

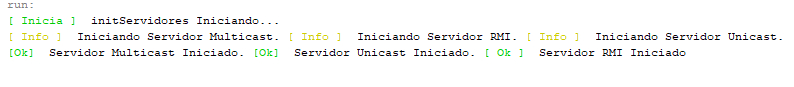
Además de para asegurarnos si nos estamos descargando e instalando un sofware fiable y no malicioso, el cifrado MD5 tiene otros usos que tal vez te interesen:

* A través de un programa podemos crear un código MD5 para alguno de nuestros archivos, y de esta manera asegurarnos que únicamente nosotros podemos hacer uso de él.
* Una vez hayamos realizado una descarga y dispongamos del archivo completo, podemos utilizarlo en una instalación de firmware como puede ser un router.
* Comprobar que un texto no ha sido modificado a la hora de enviárselo a otra persona y pueda llegar de forma distinta a como era el original. Existen páginas webs en donde nosotros metemos el texto que vamos a enviar, esta web nos devuelve el hash, y es este el que enviamos a nuestro destinatario para que compruebe que el texto es el correcto.

**Desarrollo**

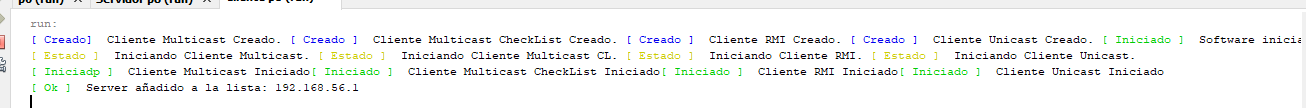
Como se menciono al inicio para esta practica se hace uso de 3 diferentes tipos de servidores corriendo en 3 hilos distintos, uno es el servidor Multicast en donde residen los otros 2 servidores, ademas de ser en donde los clientes se conectan para poder hacer la búsqueda y la descarga de los archivos en el servidor. El servidor RMI nos sirve para poder realizar la búsqueda de los archivos en nuestro servidor, ademas de que ahí también hacemos uso del MD5Checksum y finalmente el servidor unicast para poder hacer el envió de nuestro archivo encontrado si es lo que solicito el cliente.

Este es el resultado al correr los 3 servidores al mismo tiempo:

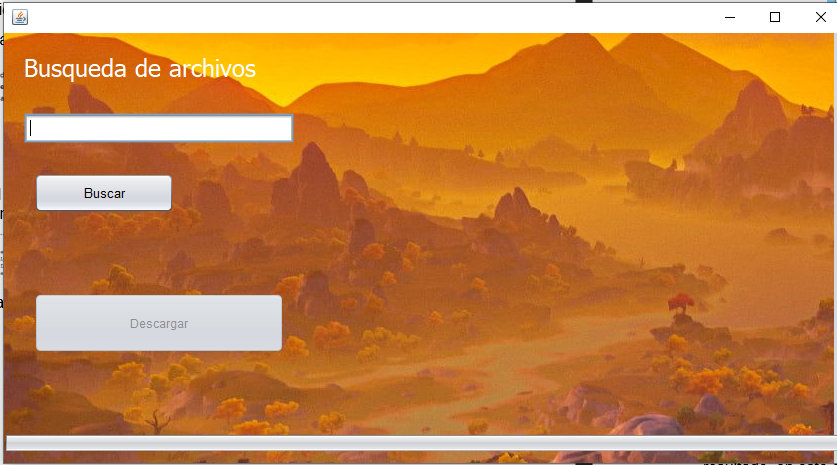


En este caso ya los servidores están a la espera de clientes con los que realizar la conexión.

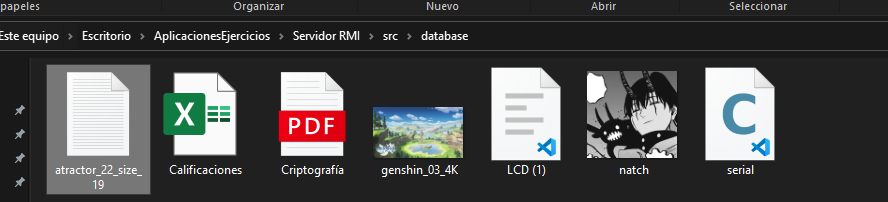
Por otro lado, el cliente intenta hacer contacto con los 3 servidores, en el caso de correr el cliente nos arroja lo siguiente del lado de cliente en consola:



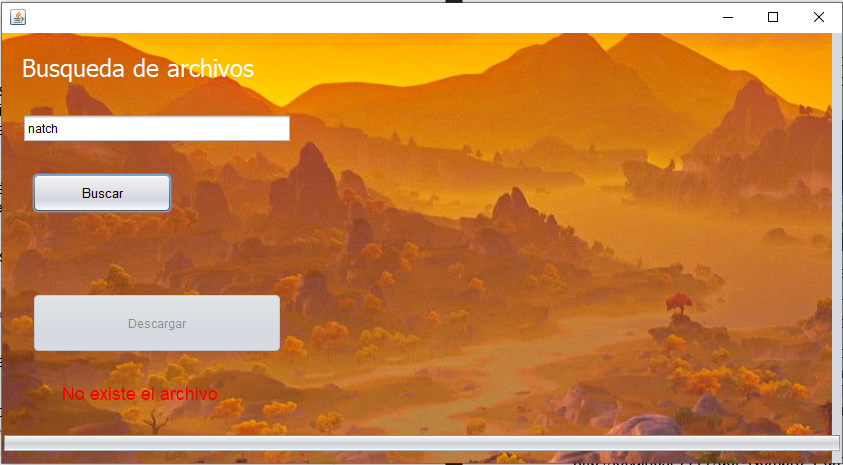
De forma grafica nos da el siguiente resultado:

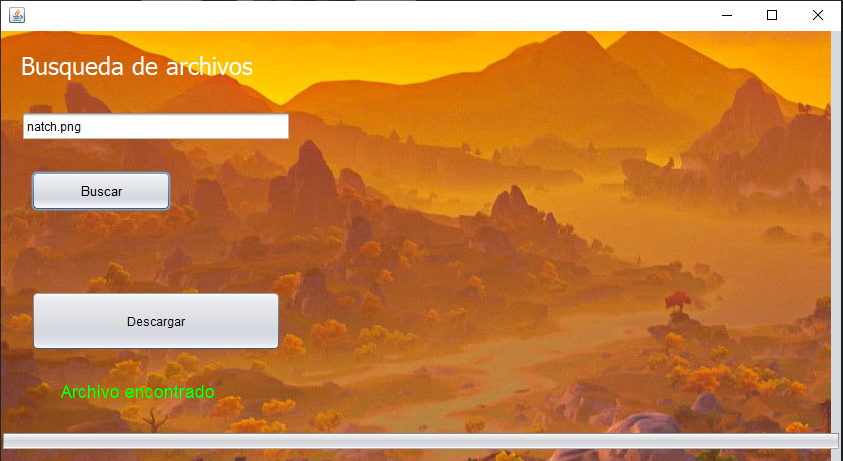


Como vemos se conforma de un campo para hacer la búsqueda de los archivos y en caso de encontrarlos nos permitirá descargarlos.

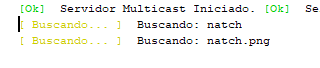
En este caso tenemos los siguientes archivos en nuestro servidor: 

Si buscamos alguno por ejemplo el que tiene el nombre de natch

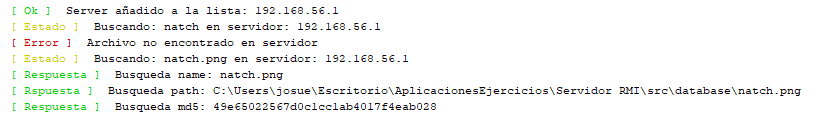


Nos dice que el archivo no existe ya que no ponemos extensión del archivo la cual es necesaria, cuando ponemos la extensión correcta nos da el siguiente resultado: 

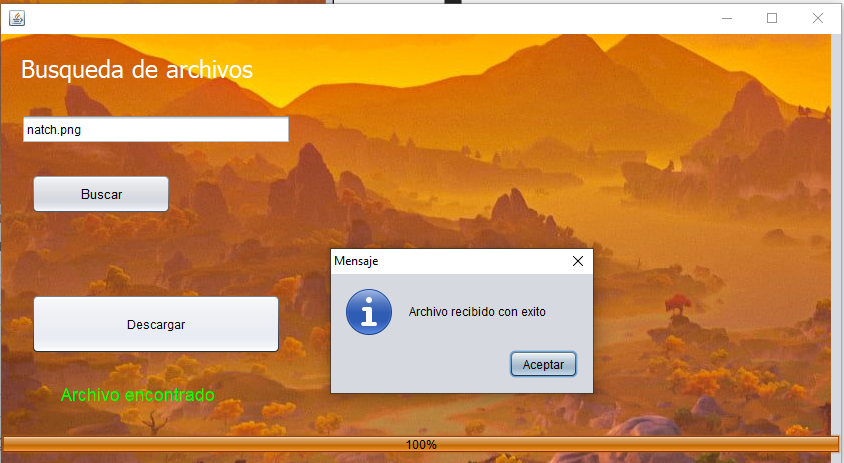
Mientras tanto en la consola del servidor nos arroja lo siguiente:

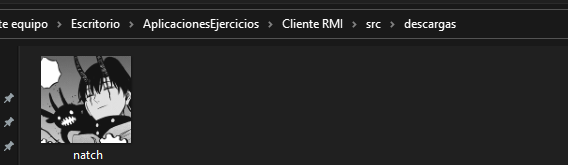


Y en el lado del cliente:

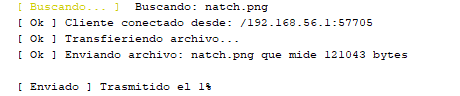


Si descargamos él archivo la barra inferior la GUI se empezará a llenar con la información que se ha descargado y cuando se a complete nos dará un mensaje de que la descarga se realiza con éxito.

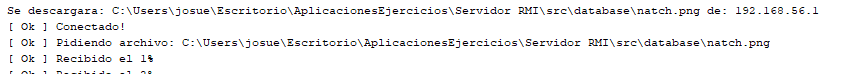


Y se descarga dentro de la carpeta llamada descargas en el proyecto y es guardado con el mismo nombre que tiene el servidor.

Del lado del servidor nos aparece lo siguiente, en donde se hace la conexión y envió por medio del servidor unicast imprimiendo que porcentaje del archivo a sido enviado al cliente.



Por parte del cliente nos dice la ruta completa del archivo que será descargada, se conecta al servidor y empieza la recepción del archivo por partes.



En esta práctica se puede observar de manera perfecta el uso de servidores RMI, multicast y unicast, prácticamente usando todo lo visto en el curso, lo cual es muy interesante, ademas, de que la aplicación que se desarrollo es una aplicación muy famosa y usada, permite el rápido entendimiento de estos temas y ademas de ser entretenido el desarrollo, agregar que el hecho de hacer uso de MD5Checksum es interesante ya que vemos un añadido bastante interesante e importante, el cual muy posiblemente será útil para futuros programas.