**ESCUELA SUPERIOR POLITECNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

**Sistemas operativos**

14

**DOCUMENTACION SNAKEBOX**

**Gustavo Totoy, Frank Hermida**

Contenido

[Introducción 2](#_Toc392832782)

[Diagrama 3](#_Toc392832783)

[Diseño Propuesto 4](#_Toc392832784)

[Manejo de archivos 4](#_Toc392832785)

[Sincronización de archivos 4](#_Toc392832786)

[Método push() 4](#_Toc392832787)

[Método pull() 4](#_Toc392832788)

[Semáforo: ReentrantReadWriteLock 5](#_Toc392832789)

[Algoritmo de sincronización 5](#_Toc392832790)

[PathDescriptor 5](#_Toc392832791)

[Bibliografía 6](#_Toc392832792)

# Introducción

**SnakeBox** es una aplicación cliente-servidor que simula el almacenamiento de archivos de texto en la nube mediante el uso de mecanismos de sincronización, el cual le permite a un usuario local almacenar archivos en su directorio local y estos son subidos automáticamente al servidor.

En este documento se describen los mecanismos de sincronización usados para resolver la subida de archivos, la comunicación entre el cliente y el servidor y la resolución de problemas como la colocación de archivos en los directorios correspondientes para cada usuario, entre otros.

# Diagrama

![](data:None;base64,)![](data:None;base64,)![](data:None;base64,)![](data:None;base64,)![](data:None;base64,)

o o o o o

![](data:None;base64,)![](data:None;base64,)

Pull()

Actualiza los archivos del cliente usando los archivos del servidor

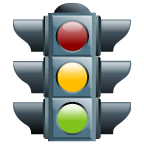


Push()

Actualiza los archivos del servidor usando los archivos del cliente



**RemoteInputStream**

****

**ReentrantReadWriteLock**

* readLock()
* writeLock()

# Diseño Propuesto

## Manejo de archivos

Para el mecanismo de comunicación cliente-servidor se usó **RMI (**java remote method invocation sus siglas en inglés.**)** [1]El cual nos ayuda a invocación de métodos de manera remota, el cual es un mecanismo simple para la comunicación de servidores basados en **JAVA.**

La librería **RMIIO** [2] se encarga de transferir grandes cantidades de datos entre un cliente y un servidor mediante el uso del mecanismo **RMI.**

Se usa la clase **RemoteInputStream** [3] que se encarga de transferir los archivos de manera local

La librería **commons IO** ayuda a las funcionalidades de entrada y salida de archivos [4].

Se usa la clase **FileUtils** [5] la cual ofrece operaciones utilitarias para la manipulación de archivos y directorios.

## Sincronización de archivos

Para la sincronización de los archivos, se usó **bloques Syncronized de java**, el cual sirve para especificar en ciertas partes del código están sincronizadas [6] y las operaciones de semáforos [7].

### Método push()

El método **push()** de la clase **SyncSnake**, sincroniza los archivos del servidor enviando los archivos alojados en el cliente. Usando **synchronized** con el candado de escritura (writeLock()) del servidor.

### Método pull()

El método **pull()** de la clase **SyncSnake**, sincroniza los archivos del cliente recibiendo los archivos alojados en el servidor. Usando **synchronized** con el candado de lectura (readLock()) del servidor.

### Semáforo: ReentrantReadWriteLock

El semáforo **ReentrantReadWriteLock** implementado en la clase **SnakeServer** tiene un atributo llamado **lock\_** el cual ofrece un candado de lectura y de escritura.

### Algoritmo de sincronización

El algoritmo consiste en actualizar los descriptores de los archivos locales, hacer **pull()** del servidor y luego hacer **push()** hacia el servidor. Se repite este proceso cada cierto tiempo.

Este algoritmo corre en un proceso en el Background[[1]](#footnote-1).

## PathDescriptor

Permite llevar la descripción de cada archivo dentro de un directorio padre, guardando información como:

* Ruta relativa al directorio padre con el atributo **relative\_path.**
* Gestiona el status del archivo mediante el uso del atributo **status.**
* Los atributos **statusTimePoint, lastModified, lastCreationTime** permiten establecer las últimas modificaciones del archivo.

# Bibliografía

|  |  |
| --- | --- |
| [1] | Oracle, "Remote Method Invocation Home," Oracle Corporation, 2014. [Online]. Available: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. [Accessed 06 Julio 2014]. |
| [2] | openhms, «Souceforge,» 23 Septiembre 2013. [En línea]. Available: http://openhms.sourceforge.net/rmiio/. |
| [3] | Openhms, «Uses of Interface com.healthmarketscience.rmiio.RemoteInputStream (RMI IO Utilites 2.0.4 API),» [En línea]. Available: http://openhms.sourceforge.net/rmiio/apidocs/com/healthmarketscience/rmiio/class-use/RemoteInputStream.html. |
| [4] | Apache, «Commons IO,» Apache Org, 17 Febrero 2014. [En línea]. Available: http://commons.apache.org/proper/commons-io/. |
| [5] | Commons, «Class FileUtils,» Apache org., [En línea]. Available: http://commons.apache.org/proper/commons-io/apidocs/org/apache/commons/io/FileUtils.html. |
| [6] | sc, «Sincronizacion,» [En línea]. Available: http://www.sc.ehu.es/sbweb/fisica/cursoJava/applets/threads/sincronizacion.htm. |
| [7] | Oracle, «Class Semaphore,» Oracle corp., [En línea]. Available: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html#acquire(). |
| [8] | J. Jenko, «Java Synchronized Blocks,» [En línea]. Available: http://tutorials.jenkov.com/java-concurrency/synchronized.html. |
| [9] | Oracle, «Class ReentrantReadWriteLock,» [En línea]. Available: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html. |

1. **Background** se utiliza para nombrar a todos aquellos procesos rutinas de ejecución que se realizan en segundo plano. Fuente: http://es.wikipedia.org/wiki/Background [↑](#footnote-ref-1)