PROYECTO ENTREGA No. 1

Introducción Sistemas Distribuidos.

PRESENTADO POR:

Fabián Merchán

Martha Vargas

PROFESOR:

Rafael Páez



Pontificia Universidad Javeriana

Introducción a los Sistemas Distribuidos.

05 de Abril. de 2016

Contenido

[Descripción 2](#_Toc447610874)

[Diagramas 2](#_Toc447610875)

[Descripción de las funciones más importantes 2](#_Toc447610876)

[Servidor 2](#_Toc447610877)

[Clase Cliente 4](#_Toc447610878)

[Servidor Intermediario 6](#_Toc447610879)

[Clase Directorio 6](#_Toc447610880)

[Clase Zócalo 7](#_Toc447610881)

# Descripción

Para el presente proyecto se utilizaron 3 clases importantes que fueron Servidor intermediario llamado *TPCServer*, Cliente, llamado *Client*, y Servidor de contenidos, llamado *Server*.

Servidor de Contenidos y Cliente

Adicionalmente fueron utilizadas las siguientes clases:

* ***Zocalo*:** la cual guarda la información de los clientes como es si IP y Puerto por el cual deben comunicarse,
* **Archivo:** Guarda la información básica de los archivos, como es nombre, peso, hash, es importante aclarar que la función de hash está definida dentro de esta clase.

El servidor intermediario utilizó además las siguientes clases:

* **Directorio:** dentro de esta clase se definieron las funciones que realiza el Servidor intermediario.
* **TorrentServer:** Se utilizó para poder correr en hilos las diferentes peticiones.

# Diagramas

El diagrama de [secuencia](Copy%20of%20secuencia.bmp) representa la interacción entre las clases, SI representa al servidor intermediario o directorio el cual contiene los archivos de todos los clientes, con su respectivo hash y nombre. El Cliente puede agregar archivos, pedir la lista de todos los archivos existentes y pedir descargar un archivo específico. Finalmente el Cliente2 representa al Servidor de Contenidos, quien se encuentra en la misma máquina del cliente, pero es el encargado de distribuir las partes necesarias de los archivos a los clientes que lo requieran.

El [diagrama de clases](Copy%20of%20Diagrama%20de%20clases%20servidor.bmp) representa al proyecto del Servidor Intermediario.

# Descripción de las funciones más importantes

El presente documento pretende dar una breve descripción de las clases más importantes en el desarrollo de la primera entrega del proyecto.

## Servidor

Dado que el servidor de contenidos y el cliente son ejecutados desde una misma máquina y sus operaciones deben ser transparentes para el usuario, se utilizaron hilos para que el servidor de contenidos pudiera escuchar todo el tiempo y así enviar a los clientes que requieran de su servicio la parte del archivo que este pida.

#### Clase Server

Esta clase extiende de Thread

Constructor

public Server() {

start();

}

Dentro del constructor se llama a la función que permite que corra el hilo, al invocar la función start(), es llamada la función run().

public void run(){

listenSocket();

}

Esta función llama al método listenSocket().

public void listenSocket()

{

try {

ServerSocket serverSocket = new ServerSocket(this.serverPort);

while(true){

Connection client;

Socket cliente = serverSocket.accept();

client = new Connection(cliente);

} } catch (IOException ex) {

Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);

}

}

Este método se encuentra en un ciclo continuo, que permite escuchar, a los clientes que necesiten descargar una parte del contenido, como se observa, se crea un socket con el cliente y se crea una instancia de la clase Connection, la cual se encuentra definida dentro de la misma clase Servidor.

#### Clase Connection

class Connection extends Thread {

Esta clase se encuentra definida dentro de la clase Server, extiende de Thread.

Dentro de su constructor podemos observar, que ingresa como parámetro el socket de la conexión con el cliente, se abre un OutputStream y se llama a la función run().

public Connection (Socket aClientSocket) {

try{

clientSocket = aClientSocket;

out = clientSocket.getOutputStream();

this.start();

} catch(IOException e) {System.out.println("Connection:"+e.getMessage());}

}

Dentro de la función run(), se lee la parte del archivo que el cliente está pidiendo al servidor de contenidos, al igual que el hash de este archivo, de esta forma el servidor de contenidos puede identificar fácilmente que archivo le están pidiendo.

DataInputStream in = new DataInputStream(clientSocket.getInputStream());

parte = Integer.parseInt(in.readUTF());

hash = in.readUTF();

Cuando se obtiene el hash se busca dentro de la carpeta que el servidor de contenido tiene guardados todos los archivos y se selecciona el que coincida con el hash recibido, para esto se buscan los todos los archivos y se llama a la función que genera el hash que es md5 y se compara con lo recibido, cuando se encuentra una coincidencia se guarda el nombre.

Archivo a = new Archivo();

File carpeta = new File ("Archivos");

File[] archivos = carpeta.listFiles();

for(File e : archivos)

{

if (a.md5("Archivos/"+e.getName()).equals(hash))

nombre = e.getName();

}

Una vez se identificó el archivo se abre un flujo con este y se guarda la información, se decidió utilizar ObjectOutputStream, para realizar la serialización y enviar los bytes necesarios.

ObjectOutputStream os = new ObjectOutputStream(clientSocket.getOutputStream());

os.flush();

Se utilizó el código visto en clase para dividir los archivos, se obtiene el número de particiones y se va guardando este contador, cuando el contador iguala el número de partición en el cual se encuentra el ciclo, esta parte es guardada en un arreglo de bytes y se escribe el objeto, para que el cliente pueda leerlo.

if(parte == nChunks)

{

String newFileName = "test.part" + nChunks;

File archivoF = new File (newFileName);

filePart = new FileOutputStream(archivoF);

os.writeObject(bytes);

archivoF.delete();

}

## Clase Cliente

La clase cliente tiene los siguientes métodos.

public int pedirDirectorio (String archivo);

public void agregarArchivo(double size, String name, String ip, String port, String path);

public ArrayList<String> pedirTodos ();

public boolean verificarConexion(InetAddress ip);

public void descargar (List<Zocalo> servidores, int tam, int partes, String nombre, String hash)

##### Pedir todos

El cliente tiene la opción de pedir todos los archivos que se subieron al torrent en algún momento, esto se realizó pensando en que el cliente puede no tener información sobre los archivos disponibles en los diferentes servidores de contenidos, en este caso deberá crear un socket y enviar “obtener todos”, el director le responderá con el número total de archivos y luego el nombre de cada uno de ellos. Esta información se imprime por pantalla.

s = new Socket(ipS,this.serverPortS);

out =new DataOutputStream( s.getOutputStream());

out.writeUTF("obtener todos");

total = Integer.parseInt(in.readUTF());

for(int i=0; i<total; i++)

archivos.add(in.readUTF());

##### agregarArchivo

Esta función es utilizada, cuando el cliente quiere adicionar un nuevo archivo, se le piden el nombre. Se obtiene el tamaño, el path y hash y estos datos son enviados al directorio.

s = new Socket(ipS,this.serverPortS);

out =new DataOutputStream( s.getOutputStream());

out.writeUTF("agregar");

out.writeUTF(name);

out.writeUTF(a.getHash());

out.writeUTF(String.valueOf(size));

out.writeUTF(ip);

out.writeUTF(port);

##### Pedir directorio.

Esta función es utilizada cuando el cliente, escribe el nombre de un archivo que quiere descargar, si el archivo existe el directorio le envía los datos de los servidores de contenidos que lo tienen y hash del archivo, para que el cliente pueda realizar las peticiones.

Cuando se tienen estos datos, se llama a la función *descargar()*. Es importante recalcar que el directorio también enviará al cliente el número de partes que este debe leer.

##### Descargar

Se crea un ciclo con el número de partes que se deben pedir, y dado que en la función *pedirDirectorio()* se obtiene la lista de todos los servidores de contenidos que tienen esa película, a medida que el ciclo aumenta se selecciona aleatoriamente el índice dentro de la lista al cual se le pedirá el archivo

servidor = ThreadLocalRandom.current().nextInt(0, servidores.size());

Seguido de esto se crea la conexión con el Servidor de Contenidos y se le envía el número de parte que se le quiere pedir y el hash para que este pueda identificar fácilmente el archivo a enviar.

s = new Socket(servidores.get(servidor).getIp().getHostAddress(),this.serverPort);

DataOutputStream output = new DataOutputStream(s.getOutputStream());

output.writeUTF(String.valueOf(i));

output.writeUTF(hash);

Luego se recibe el arreglo de bytes y este debe ir guardando en el archivo principal lo que se lleva. Al final se realiza un flush que permite borrar información de anteriores iteraciones.

ObjectInputStream is;

is = new ObjectInputStream(s.getInputStream());

bytes = new byte [2048];

bytes = (byte[]) is.readObject();

fos.write(bytes);

fos.flush();

bytes = null;

Al finalizar la función se llama a la función *AgregarArchivo()*, para que el Directorio pueda reconocer que la máquina cuenta con ese nuevo archivo.

agregarArchivo(tamfinal, nombre, InetAddress.getLocalHost().getHostAddress(), String.valueOf(7895), "Archivos/"+nombre);

# Servidor Intermediario

## Clase Directorio

La clase directorio maneja un mapa, con un archivo y una lista de Zócalos que contienen la información de los sockets disponibles en los otros clientes. La concurrencia se maneja gracias a una librería de contenedores concurrentes de java.

List<Zocalo> zocalosDisponibles(String nombre): Devuelve la información de los sockets disponibles para el archivo dado.

List<String> archivosDisponibles(): Devuelve una lista con todos los archivos disponibles.

agregarZocalo( String archivo, String hash, Double peso, String ip, String puerto ): Añade el socket indicado al archivo con los atributos indicados.

Clase Archivo

La clase archivo contiene la información de un archivo, tal como nombre, código hash y peso, además calcula el número de partes en que un archivo debería ser dividido.

int numeroPartes():Retorna el número de partes en que un archivo debería ser dividido.

## Clase Zócalo

La clase zocalo contiene la información de un socket disponible: puerto e ip.