Memoria: Trabajo de prácticas.

Nombre: Alejandro García Casanova
Correo: GarciaCasanovaAlejandro@gmail.com

Indice:

1.Procedimiento para utilizarlo:

El programa y lo necesario para probarlo está subido al laboratorio en el usuario ssd27, dentro de la carpeta entregaTrabajo.
/home/alumnos/ssd27/entregaTrabajo

1. Primero abres un terminal en la carpeta tracker y ejecutas el archivo archivo tracker.java desde la consola con el comando: “java -jar tracker.jar”

2. Luego abres un terminal en la carpeta seed y ejecutas el archivo archivo seed.java desde la consola con el comando: “java -jar seed.jar”.Este proceso lo puedes repetir varias veces, para tener más de un archivo que se puedan descargar. Cada vez que ejecutes el comando y cargue saldrá una ventana en la que tendrás que elegir el archivo a seedear.

3. Por ultimo abres un terminal en la carpeta peer y desde el terminal de linux tienes que indicar que ejecute el programa tantas veces como quieras, cada vez con una JVM distinta, para ello se puede utilizar la siguiente función: “for i in {1..5}; do konsole -e java -jar peer.jar &; done” puedes sustituir el número 5 por el número de peer que quieres.

4. Los archivos descargados aparecerán en la carpeta resultados con el puerto del peer en el nombre.

5. Para la comprobación puedes abrir un terminal en la carpeta de resultados y poner el comando: “md5sum *“ y ver que los archivos tienen el hash correcto.

Nota:

He intentado probarlo en el servidor del laboratorio desde ssh y no he podido porque falla al lanzar el JfileChooser para elegir archivo(no puedo lanzar la ventana por ssh), en local me funciona y en el laboratoria también debería.

2.Descripción de la aplicación.

Desde el tracker:

Para empezar se necesita el servidor tracker (“TrackerThread”) el cual al ejecutarlo crea un serversocket en un puerto fijo, y se abre para recibir peticiones. Cuando recibe una petición por parte de un peer (actúe como peer o como seed), lanza un thread que llama a la clase tracker.

Este thread comprueba si se trata de un seed o un peer. Si es un seed le pide la información del archivo que quiere registrar (ip, puerto, nombre del archivo, tamaño del archivo, hash del archivo completo y listado con los hash del archivo dividido en partes) y la guarda en un constructor de tipo “listado”, el cual a su su vez se guarda en un list<listado> del tipo “listado” ,creado previamente en “TrackerThread”.

En caso de ser un peer actualizo mi lista de archivos y se la envió, luego el peer me dice el archivo que quiere descargar (esta configurado para elegir un archivo de forma aleatoria entre los del listado) y el tracker le devuelve un listado con las ip y los puertos de los peer que tienen el archivo elegido, despues tambien se le envia la informacion del archivo en si (nombre, tamaño, hash del archivo entero, hashes de las partes). Cuando el peer tiene toda la información se registra en el constructor “listado” y posteriormente en la list<listado>. Al finalizar se corta el socket entre el peer y tracker, y se cierra el thread.

Desde el seed:

Al iniciar la clase peer como seed (parámetro registrar) me conecto con el tracker, luego tengo que elegir un archivo a seedear, al elegirlo , obtengo su tamaño, su hash, y una lista con los hashes de el archivo por partes y le envió la información al tracker. Una vez tengo la informacion creo un AtomicIntegerArray llamado “mibitmap” y lo relleno con unos, para indicar que tengo todas las partes del archivo. Termino mi conexión con el tracker y abro un thread (“PeerAsServer”), el cual se encarga de manejar las peticiones que otros threads me envían, creando un thread para cada petición (“traspaso”), hay 3 posible peticiones:

Enviame tu bitmap: la cual se suele utilizar la primera vez que se conectan a ti, yo le envío mi bitmap, y recibo el suyo junto con su información y el conjunto es guardado en un constructor llamado “matrix”, que a su vez es guardado en una list<matrix> llamada “mimatrix”, cada peer, ya sea seed o peer tiene su propio “mibitmap” y su propia “mimatrix”.

Update: un peer quiere que yo actualice la información suya que tengo en mi “mimatrix” con los avances de su bitmap.

Pedir parte: El peer me dice que quiere descargarse una parte en concreto de mi. Yo voy al archivo, voy a la parte en concreto que desea y se la envío por el socket, y cierro la conexión con el peer.

Ese sería el funcionamiento del seed, en cualquier momento puedo salir del enjambre ejecutando la orden salir, la cual borraría mi información del tracker.

Desde el peer:

Al iniciar la clase peer como peer lo primero que hago es conectarme al tracker (identificandome como peer) y pidiéndole el listado de archivos que tiene, luego elijo uno (está puesto que sea aleatorio, pero previamente se ha probado con un scanner, indicando el nombre por teclado), me devuelve un listado con los peer que tienen ese archivo (completo o alguna parte), además de toda la información necesaria. Después me registro en el tracker para que otros se puedan conectar a mi.

Después creo "mibitmap" rellenando con ceros (aún no tengo ninguna parte), creo el RandomAccesFile donde reservaré el espacio para el archivo y inicializo el thread "PeerAsServer" que tendrá la misma función que en el seed, aceptar peticiones de otros seed, de forma que las partes que yo me vaya descargando ellos se la puedan descargar de mí.

Inicializo "mimatrix" y la relleno preguntando a cada uno de los peer del listado que me dio el tracker que me de su "mibitmap", y yo le doy el mio.

Ahora toca elegir las partes y descargarlas, elijo 5 partes al azar de todas las posible, hago un listado con los peer que tienen la parte elegida, y elijo uno de ellos aleatoriamente (si no lo tiene nadie vuelvo atrás), me conecto a el, le pido la parte elegida previamente, el me la envia y yo la guardo directamente en el espacio de memoria asignado, antes de dar por finalizado el proceso compruebo el hash de la parte recibida con el listado de hash de las partes, si es correcto actualizo "mibitmap" y se lo mando a todo mi listado de "mimatrix", una copia de "mibitmap" actualizado, y continuo con la siguiente parte.

Después de tener las 5 aleatorias, pasa a ejecutar un proceso en el que revisando "mimatrix" obtengo las partes menos frecuentes y intento descargar esas partes utilizando el proceso anterior.

Al finalizar todo el proceso compruebo el hash final, con el obtenido por el tracker, si falla, compruebo el listado de los hash por partes, y vuelvo a descargarme, las partes que no coincidan.

3. Clasificación de clases.

Mirar archivo esquema.html

4.Definición de clases y métodos.

Mirar javadoc/index.html