Enunciado del proyecto de prácticas: *nanoFiles*

Redes de Comunicaciones - Curso 2023/24

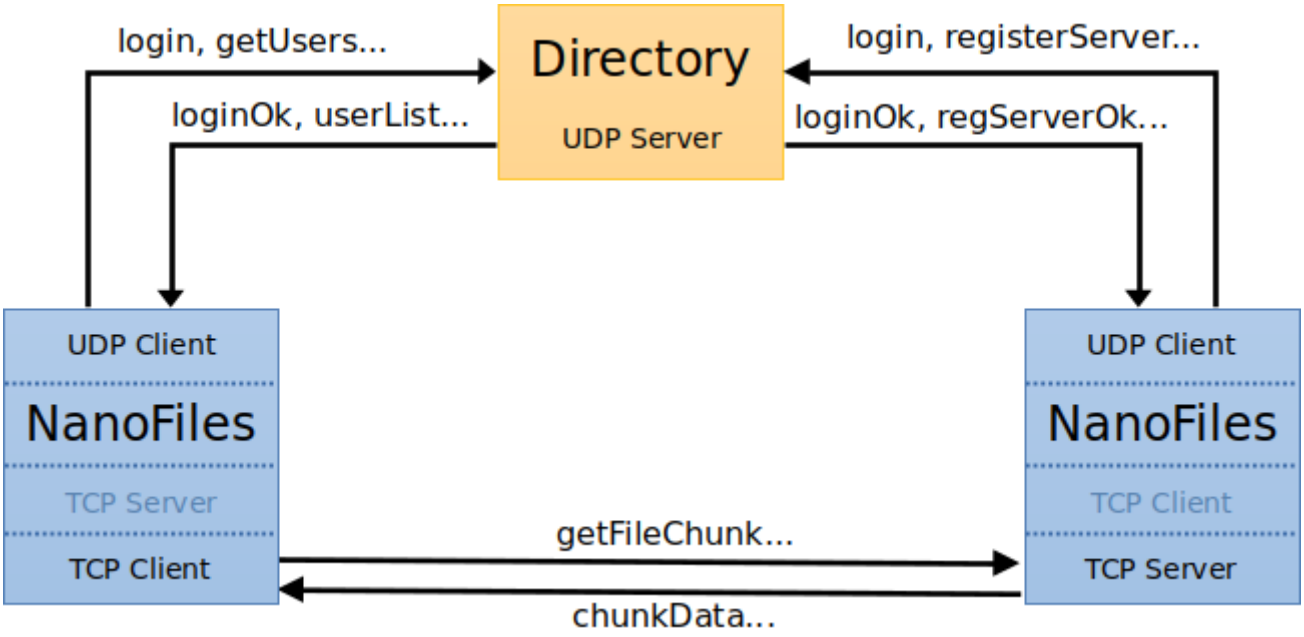
Visión global y objetivos

En el desarrollo de esta práctica, el alumnado aprenderá a diseñar y a programar protocolos de comunicación en Java. La práctica consiste en realizar un sistema de compartición y transferencia de ficheros al que denominaremos *nanoFiles*, y que se describe a continuación.

El sistema *nanoFiles* está formado por un servidor de directorio (programa *Directory*) y un conjunto de *peers* o pares (programa *NanoFiles*), que se comunican entre sí de la siguiente forma:

- Por un lado, la comunicación entre cada *peer* de *NanoFiles* y el servidor de directorio se rige por el modelo *cliente-servidor*. Un *peer* actúa como cliente del directorio para *loguearse* con un nombre de usuario, consultar los nombres de usuario logueados, consultar los ficheros que pueden ser descargados de otros *peers*, publicar los ficheros que quiere compartir con el resto de pares, etc.
- Por otro lado, el modelo de comunicación entre pares de *NanoFiles* es *peer-to-peer* (P2P).
    - Cuando un *peer* actúa como cliente de otro *peer* servidor, el cliente puede descargar los ficheros disponibles en el servidor, o parte de los mismos.
    - De manera complementaria, un *peer* puede convertirse a petición del usuario en servidor de ficheros, de forma que escuche en un puerto determinado en espera de que otros *peers* se conecten para solicitarle los archivos que el servidor está compartiendo.

La siguiente figura muestra un esquema general orientativo del conjunto de componentes de *nanoFiles*:



El objetivo global de esta práctica es que el alumnado sea capaz de diseñar los protocolos necesarios para intercambiar los mensajes involucrados así como de implementar el software de todos los elementos que constituyen el sistema *nanoFiles*.

Especificación de la práctica

Dependiendo de la funcionalidad implementada por el alumnado, es posible que un *peer* pueda llegar a actuar simultáneamente como cliente y como servidor de otros *peers* (modelo de comunicación P2P), si bien como paso previo (funcionalidad mínima obligatoria) se contempla el escenario en que un *peer* únicamente pueda actuar en uno de los dos roles (cliente o servidor) en un instante dado (modelo de comunicación cliente-servidor).

En el sistema *nanoFiles*, el primer paso que obligatoriamente deben llevar a cabo todos los pares es acceder (*login*) al servidor de directorio para confirmar que está disponible. Una vez el *peer* se ha *logueado* con éxito, el directorio devolverá al *peer* una **clave de sesión única para cada usuario** (un entero no negativo) que el *peer* deberá utilizar en lo sucesivo para identificarse ante el directorio adecuadamente a la hora de realizar cualquier solicitud. A partir de ese momento, se asumirá que el resto de acciones que requieran de comunicación con el directorio se efectuarán con el mismo servidor, hasta que el usuario decida *cerrar sesión*, para así poder *loguearse* con otro servidor de directorio distinto.

Una vez se han *logueado* en el directorio, un *peer* tiene la posibilidad de convertirse en servidor de ficheros, de forma que acepte conexiones de otros peers para solicitarle descargar alguno de los ficheros que comparte. Para que el resto de *peers* puedan conocer qué servidores de ficheros hay disponibles en un instante dado, es necesario que cada nuevo par servidor comunique al directorio en qué puerto está a la escucha de conexiones de otros pares. Opcionalmente, el servidor enviará también la lista de ficheros que comparte.

En *NanoFiles*, los pares cliente que desean descargar ficheros de otros pares obtienen la información necesaria para conectarse a los pares servidor a través del directorio. Existen dos formas alternativas de llevar a cabo una descarga: bien directamente de un único *peer* servidor indicado explícitamente al descargar, o bien de todos los *peer* servidores que comparten el fichero que el usuario desea descargar.

Para facilitar la tarea del alumnado, como paso previo a la descarga *directa* desde un único servidor, será posible conectarse al *peer* servidor especificando directamente la IP y puerto al que conectarse. De esta forma, no será necesario ni registrar en el directorio al arrancar el servidor de ficheros, ni tampoco consultar al directorio para averiguar la IP y puerto de escucha, sino que esta información será obtenida por medios externos al sistema *nanoFiles*, p.ej., conociendo de antemano la IP del *peer* servidor y acordando el puerto en el que escuchará.

Diseño de los protocolos necesarios

Específicamente, el alumnado deberá llevar a cabo el diseño de dos protocolos de comunicación de nivel de aplicación del sistema *NanoFiles*:

- Para la comunicación entre cada *peer* y el directorio, el alumnado diseñará un protocolo confiable semi-dúplex basado en parada y espera, que operará sobre un protocolo de nivel de transporte no confiable como UDP.
- Para la comunicación entre dos *peers*, el alumnado diseñará un protocolo asumiendo que el protocolo de nivel de transporte es confiable (como TCP).

En particular, los estudiantes deberán realizar:

- Respecto a los autómatas, se pide especificar, por separado, los siguientes autómatas:

    - Autómata para el servidor de directorio
    - Autómata para un *peer* cuando actúa como cliente del directorio
    - Autómata para un *peer* cuando actúa como cliente de otro *peer* servidor.
    - Autómata para un *peer* cuando actúa como servidor de otro *peer* cliente.
- Respecto a las especificaciones de los mensajes a intercambiar, se deben utilizar los siguientes lenguajes:
    - **Mensajes textuales** (*Field:Value*) para el protocolo de comunicación con el directorio.
    - **Mensajes binarios** para la comunicación entre pares.

Se deja a elección del alumnado las decisiones relativas a los diferentes tipos de mensajes necesarios así como a la codificación de los distintos campos.

Implementación del software

En paralelo a la fase de diseño, los estudiantes tendrán que implementar los elementos software necesarios para la práctica. Para facilitar esta tarea, se proporciona al alumnado un material de partida que consta de los siguientes elementos:

- Esqueleto de código del servidor de directorio
- Implementación sin terminar de la parte cliente del programa *NanoFiles*, incluyendo:
    - Interfaz de usuario, basada en un sencillo intérprete de comandos (ya implementado).
    - Lógica del programa cliente (parcialmente implementado).
- Implementación esbozada de la parte servidor del programa *NanoFiles*, incluyendo:
    - Servidor capaz de ejecutarse en primer plano
    - Servidor capaz de ejecutarse en segundo plano
    - Clase que modela los hilos que pueden ser creados por el servidor para atender a los clientes conectados.
- Clases auxiliares:
    - Para la representación de mensajes (sólo esbozadas).
    - Para la obtener la base de datos de ficheros compartidos (ya implementada)

Así pues, el trabajo del alumnado consistirá en completar tanto el programa NanoFiles (código que ejecuta cada uno de los *peers*) como el programa Directory (servidor de directorio) para conseguir que la práctica sea funcional.

Lógica del programa *Directory*

El programa *Directory* quedará a la espera, una vez lanzado, de los mensajes que vaya recibiendo y respondiendo a peticiones/consultas de los *peers*: *loguearse* en el directorio, registrarse con un nombre de usuario, publicar su lista ficheros compartidos, etc. El servidor de directorio utilizará el **puerto 6868/udp** para recibir mensajes entrantes.

Con la finalidad de que el alumnado pueda depurar el correcto funcionamiento de los clientes que le solicitan servicio, **el directorio debería imprimir por consola un breve resumen de los mensajes recibidos y enviados**, así como de aquellos mensajes que se han perdido.

El programa *Directory* ya tiene implementado el mecanismo por el cual se le puede indicar cuál es la probabilidad de que se pierda un paquete en la comunicación con el directorio. Esto se consigue mediante la opción `-loss <probability>` y resultará útil para verificar el correcto funcionamiento de la

implementación del protocolo entre el directorio y los pares, ya que dicho protocolo de nivel de aplicación debe garantizar la entrega confiable a pesar de que se produzcan pérdidas en el nivel de transporte (UDP). El valor de probabilidad oscila entre 0 (no se pierde ningún datagrama) y 1 (se pierden todos). Por defecto, la probabilidad de pérdida es 0.

Lógica del programa *NanoFiles*

El programa *NanoFiles* interactuará con el usuario mediante una línea de comandos. Al ejecutarlo, se le puede proporcionar opcionalmente un argumento, que es la ruta al directorio donde se ubican los ficheros que el *peer* comparte en el sistema. Por defecto, el programa *NanoFiles* buscará en un directorio `nf-shared` a partir del directorio actual. Dicho directorio será creado en el directorio actual al ejecutar el programa en caso de que no exista. En el escenario habitual en el que el programa se ejecuta desde el IDE Eclipse, el directorio compartido `nf-shared` se creará en el directorio raíz del proyecto Java (p.ej., `nanoFilesP2Palumnos`), al mismo nivel que los directorios `bin` y `src`.

Una vez en ejecución, el programa cliente acepta un conjunto de órdenes, si bien la funcionalidad asociada a la mayor parte de dichas órdenes no está programada y por tanto no hay ningún efecto al teclearlas. Dicha funcionalidad debe ser añadida por el alumnado, aunque la implementación de varias de las órdenes que ya acepta el *shell* del programa *NanoFiles* no forma parte de funcionalidad mínima exigida sino que constituyen posibles mejoras. Tanto la funcionalidad básica exigida como las posibles mejoras serán descritas más adelante en este documento.

Las órdenes cuya funcionalidad ya está implementada, y que no suponen ninguna comunicación con entre procesos, son:

- `quit`: Sale del programa.
- `help`: Muestra ayuda sobre las órdenes soportadas.
- `myfiles`: Muestra los ficheros de la carpeta compartida por este *peer*, indicando su nombre, tamaño y *hash*. El directorio por defecto es `nf-shared` y por lo general se ubicará dentro del directorio del proyecto Eclipse.
- `sleep <seconds>`: Bloquea el shell (*duerme*) durante el número de segundos indicado. Esta órden puede resultar útil a la hora de la automatización de pruebas mediante *shell scripts*.

Por otro lado, las órdenes que reconoce el *shell* de *NanoFiles* cuya funcionalidad está pendiente de implementar son:

- `login <directory_host> <nickname>`: Accede al servidor de directorio dado por el nombre (o IP) del host que se pasa como primer parámetro (`<directory_host>`), y registra a este *peer* en el directorio con el nombre de usuario especificado como segundo parámetro (`<nickname>`). El login puede fallar si el nombre de usuario especificado ya está siendo utilizado por otro *peer*.

  > Durante el desarrollo de las prácticas, el escenario más habitual será que tanto el programa *Directory* como el programa *NanoFiles* se ejecuten en la misma máquina; en tal caso, bastará con teclear `login localhost mynick` para establecer la comunicación entre ambos procesos a través de la interfaz *loopback*.

- `userlist`: Muestra los nombres de usuario (*nicknames*) de los *peers* que se han logueado en el directorio.
- `fgserve`: Lanza un servidor de ficheros en primer plano (*foreground*), para escuchar conexiones en un puerto predefinido. Este servidor se ejecuta en el hilo principal del programa, por lo que una vez en marcha no resulta posible realizar ninguna otra interacción con el programa *NanoFiles* a través del *shell*.
- `bgserve`: Lanza un servidor de ficheros en segundo plano (*background server*). Este servidor se ejecuta en un hilo del programa distinto al principal, por lo que es posible continuar utilizando el programa a través del *shell*, e interaccionando como cliente tanto con el directorio como con otros *peers*.
- `downloadfrom <nickname> <filehash> <filename>`: Descarga el fichero identificado por el segundo parámetro (`<filehash>`) del servidor de ficheros indicado como primer parámetro (`<nickname>`), y lo guarda con el nombre indicado como tercer parámetro `<filename>`. En su versión mínima, debe tolerar que el primer parámetro pasado sea una cadena en formato `<IP>:<puerto>`, en vez del *nickname* del servidor.
- `publish`: Publica en el directorio los metadatos de los ficheros que este *peer* tiene en su carpeta compartida.
- `filelist`: Muestra los ficheros compartidos por otros *peers* que han sido publicados en el directorio.
- `search <filehash>`: Busca en el directorio el fichero identificado por el primer parámetro (`<filehash>`), y devuelve la lista de los *nicknames* de los servidores de ficheros que lo tienen disponible (y lo han publicado al directorio con `publish`).
- `download <filehash> <filename>`: Descarga el fichero identificado por el primer parámetro (`<filehash>`) de todos los servidores de ficheros que lo tengan disponible, y lo guarda con el nombre indicado como segundo parámetro `<filename>`. El cliente descargará una parte del fichero de cada servidor disponible, e idealmente lo hará de manera simultánea.
- `stopserver`: Detiene el servidor de ficheros en segundo plano lanzado por una orden `bgserve` anterior.
- `logout`: Cierra sesión en el directorio, y da de baja el *nickname* registrado anteriormente mediante el comando `nick`.

La secuencia correcta en la cual se pueden teclear dichas órdenes, o las situaciones en las que algunas de ellas no están permitidas, dependerá del autómata que el alumnado haya diseñado. Por tanto, será responsabilidad de los mismos controlar dicho orden.

Con la finalidad de que el alumnado pueda depurar el correcto funcionamiento de sus servidores de ficheros, se sugiere que se imprima un resumen de los mensajes recibidos y enviados.

Funcionalidad obligatoria a implementar

Ninguno de los programas que se entregan al alumnado está concluido: la fase de implementación de las prácticas de la asignatura consiste en terminar su funcionalidad. Para ello el alumnado tendrá que llevar a cabo, entre otras, las siguientes tareas:

- Implementar el formato de mensajes diseñado para que los segmentos UDP intercambiados con el servidor de directorio contengan mensajes adecuadamente formateados.
- Implementar el formato de mensajes diseñado para que los segmentos TCP intercambiados entre un *peer* cliente y un *peer* servidor contengan mensajes adecuadamente formateados.

Con respecto a los requisitos mínimos para que un proyecto de prácticas se considere satisfactorio, será necesario implementar al menos las siguientes funcionalidades:

1. Comando `login`: Acceder al servidor de directorio para comprobar que está operativo y registrar un nombre de usuario (*nickname*). El directorio deberá verificar si el *nickname* proporcionado por un usuario es válido (i.e., no está duplicado) e informar consecuentemente. Tras un *login* exitoso, se debe mostrar la clave de sesión asignada por el directorio a ese usuario.

2. Comando `userlist`: Mostrar la lista de nombres de usuario (*nicknames*) que hay registrados en el servidor de directorio

3. Comando `fgserve`: Lanzar un servidor de ficheros en primer plano, que escuche conexiones en un puerto prefijado (10000).

4. Comando `downloadfrom`: Descargar un fichero identificado por su *hash*, de entre los ficheros disponibles en un *peer* servidor de ficheros del que conocemos de antemano su IP y puerto. Se debe poder indicar una subcadena del *hash*, no necesariamente el *hash* completo. El servidor deberá informar en caso de que la subcadena proporcionada no concuerde con ningún fichero compartido, o sea ambigua.

5. Comando `logout`: Cerra sesión en el directorio, dando de baja el nombre de usuario registrado en el directorio anteriormente.

La implementación de esta **funcionalidad básica** permite alcanzar un calificación de hasta un **máximo de 5 puntos**.

Funcionalidad opcional a implementar (mejoras)

Para obtener una mayor calificación en la práctica se propone la siguiente funcionalidad adicional que podría implementarse, cuya puntuación se detalla en la tabla posterior. Junto a las mejoras propuestas, se valorarán también de forma positiva otras mejoras que el alumnado quiera plantear al profesorado.

1. **Ampliar comando *fgservers***: Ampliar comando `fgserve` para si el puerto de escucha predeterminado (10000) no está disponible, el servidor pueda utilizar otro números de puerto que sí lo estén (10001...). De esta forma, será posible tener varias instancias del programa *NanoFiles* ejecutándose en la misma máquina, actuando como servidor.

2. **Download por *nick***: Ampliar comando `downloadfrom`: Descargar un fichero identificado por su *hash*, de entre los ficheros disponibles en un *peer* servidor de ficheros del que conocemos únicamente su *nickname*. La IP y puerto del servidor se deberá obtener del directorio, que deberá mantener un registro de qué servidores hay activos junto con sus direcciones de *socket*.

3. **Comando `bgserve`**: Ejecutar el servidor de ficheros en segundo plano, de forma que el programa pueda seguir aceptando comandos del usuario a través del *shell* mientras se sirven ficheros. Se considerará suficiente que servidor de ficheros sea capaz de gestionar únicamente un cliente conectado en cada instante.

4. **Servidor multi-hilo**: Ampliar funcionalidad de `bgserve` para atender a múltiples clientes conectados simultáneamente al mismo servidor en segundo plano. Esto es una ampliación de la mejora anterior para eliminar la limitación de una única conexión en cada instante.

5. **Comando `stopserver`**: Detener el servidor de ficheros en segundo plano lanzado por una orden `bgserve` anterior.

6. **Puerto efímero**: Ampliar comando `bgserve` para que el servidor pueda utilizar cualquier número que esté disponible (*ephemeral port*) para escuchar, en lugar de un número de puerto prefijado.

7. **Ampliar comando `userlist`**: En la lista de nombres de usuario (*nicknames*) que hay registrados en el servidor de directorio, mostrar cuáles son servidores de ficheros.

8. **Comando `publish`**: Enviar al directorio los ficheros compartidos por un *peer* (comando `publish`). Esta mejora por sí sola no es completa, sino que debe realizarse con al menos una de las dos siguientes mejoras:

9. **Comando `filelist`**: Mostrar la lista de todos los ficheros que los *peers* han publicado en el servidor de directorio. Para cada fichero habrá que indicar su nombre, tamaño y *hash*. Esta mejora implica implementar la funcionalidad de `publish`.

10. **Comando `search`**: Mostrar los *nicknames* de los servidores que tienen disponible un determinado fichero identificado por su *hash*. Esta mejora implica implementar la funcionalidad de `publish`.

11. **Ampliar comando `filelist`**: Mostrar junto a cada fichero disponible, la lista de servidores (*nicknames*) que lo están compartiendo, además del nombre, tamaño y *hash* del fichero.

12. **Baja ficheros y servidores**: Mantener permanentemente actualizada la información sobre ficheros compartidos (comando `filelist`). Si un *peer* deja de servir ficheros (detiene su servidor de ficheros) deberá comunicarlo al directorio para dar de baja la lista ficheros que comparte.

13. **Comando `download` secuencial**: Descargar un fichero de múltiples servidores que comparten el fichero, de forma que se descarga de cada servidor un fragmento del fichero.

14. **Comando `download` paralelo**: Ampliar comando `download` para descargar simultáneamente los fragmentos del fichero obtenidos de diferentes servidores de manera simultánea.

La siguiente tabla resume las mejoras propuestas e indica la puntuación adicional que podría obtenerse con la implementación de cada una de ellas.

Mejora	**Puntuación máxima**
`fgserve` puerto variable	0,5 punto(s)
`downloadfrom` por *nickname*	1 punto(s)
`bgserve` secuencial	1 punto(s)
`bgserve` multihilo	0,5 punto(s)
`stopserver`	0,5 punto(s)
`bgserve` puerto efímero	0,5 punto(s)
`userlist` ampliado con servidores	0,5 punto(s)
`publish` + `filelist`	0,5 punto(s)
`publish` + `search`	0,5 punto(s)
`filelist` ampliado con servidores	0,5 punto(s)
Baja ficheros y servidores	0,5 puntos
`download` secuencial	1 punto(s)
`download` paralelo	2 punto(s)

Detalles de la entrega

El trabajo que los estudiantes deberán desarrollar es el siguiente:

- Programa cliente/servidor *NanoFiles* y programa servidor de directorio *Directory*.
- Documentación de la práctica.

Instrucciones detalladas:

- Las prácticas deben ser realizadas obligatoriamente por **grupos de dos personas**.
- Los grupos deberán subir al Aula Virtual, a la tarea denominada "Práctica de *nanoFiles*" un archivo comprimido .ZIP que contenga lo siguiente:
    - El **código fuente** en Java del proyecto, listo para ser importado y ejecutado en Eclipse.
    - Los programas `Directory` y `NanoFiles` en formato **JAR ejecutable**. Los ficheros se deben llamar **obligatoriamente** `NanoFiles.jar` y `Directory.jar`, y deben funcionar con **Java 17** usando los scripts de automatización de tests que se proporcionarán antes de la entrega.
    - La documentación del proyecto. Debe estar en formato de documento PDF e incluir al menos los siguientes apartados:
        - Introducción.
        - Protocolos diseñados.
            - Directorio:
                - Formato de los mensajes y ejemplos de los mismos.
                - Autómatas cliente y servidor
            - Peer-to-peer:
                - Formato de los mensajes y ejemplos de los mismos.
                - Autómatas cliente y servidor
        - Mejoras implementadas y breve descripción sobre su programación.
        - Capturas de pantalla que muestren mediante Wireshark un intercambio de mensajes con el directorio.
        - Opcional: Enlace a grabación de pantalla mostrando los programas en funcionamiento.
        - Conclusiones.
- **El proyecto entregado debe funcionar en los laboratorios de la Facultad**, en el sistema operativo **Ubuntu**, y en un escenario en el que tanto los *peers* como el directorio se ejecuten en **hosts distintos**.

La fecha tope de entrega será el **día 3 de mayo de 2023 a las 23:55**, a través de la tarea del Aula Virtual creada a tal efecto.

Las entrevistas se desarrollarán preferentemente durante la siguiente semana, en el horario y lugar habitual de las clases prácticas, aunque, de ser necesario se añadirán turnos adicionales. Los estudiantes se incribirán para las entrevistas a través de la herramienta *Apúntate*, que cada profesor publicará con suficiente antelación.