GÁLVEZ DÍAZ, JOAQUÍN
URBELZ ALONSO-CORTÉS, JORGE

REDES DE COMUNICACIONES
GRUPO 3.1 y 3.3
CURSO 2023/24 – MAYO
GAONA RAMÍREZ, EPIFANIO

MEMORIA NANOFILES

Contenido

Introducción 3
Formato de los mensajes del protocolo de comunicación con el Directorio 4
Formato de los mensajes del protocolo de transferencia de ficheros 6
Autómatas de protocolo 8
Ejemplo de intercambio de mensajes 10
Conclusiones 13

Introducción

Para una buena realización de la práctica de NanoFiles, debemos tener primero en cuenta el funcionamiento de esta. Además, es nuestra labor como creadores de la práctica poner en contexto a los corregidores de las distintas ideas que hemos tenido durante el diseño de esta. Es por esto en esta entrega vamos a dejar constancia de los distintas partes de la práctica que hemos ido trabajando. Por un lado, dejaremos definido el formato de los mensajes de cada uno de los protocolos: en la comunicación con el Directorio (cliente-Directorio) y en la transferencia de ficheros (cliente actuando como peer-cliente).

En cuanto a las mejoras que hemos considerado hasta este momento tratan únicamente de la implementación del comando “registeredUsers”, el cual tiene una idea similar al comando “userlist”. Este nuevo comando, será utilizado por el cliente con el director. El cliente debe enviar una lista de nombres, mientras que el directorio responderá afirmando verdadero o falso, dependiendo de si el nombre corresponde a algún usuario conectado.

Otra tarea opcional ha sido la de ampliar el número de puerto del servidor. Por defecto, intentará conectarse al puerto 10000, pero si no puede lo intentará con el siguiente. Así, hasta un máximo de 10 veces.

Formato de los mensajes del protocolo de comunicación con el Directorio

En esta fase del diseño aplicaremos las distintas decisiones que hemos tomado para el envío de mensajes entre el cliente y el Directorio. La base de la comunicación ronda sobre el cliente, ya que será este el que se encargue de enviar un mensaje y esperar su respectiva respuesta. El Directorio solamente enviará mensajes como respuesta a los mensajes que reciba de cada cliente.

Tipos y descripción de los mensajes

Mensaje: login
Sentido de la comunicación: Cliente → Directorio
Descripción: Este mensaje lo envía el cliente de NanoFiles al Directorio para solicitar "iniciar sesión" y registrar el nickname indicado en el mensaje. Este mensaje es necesario para que cualquier usuario inicie sesión con su nombre en el directorio.
Ejemplo:

operation: login\n
nickname: pepe\n
\n

Mensaje: loginOK
Sentido de la comunicación: Directorio → Cliente
Descripción: Cuando el directorio reciba el mensaje, tiene que comprobar que el usuario no está registrado en el directorio. Si no está registrado, le asigna una clave de sesión y se la envía mediante un mensaje al usuario, que la necesitará para solicitudes posteriores.
Ejemplo:

operation: loginOK\n
nickname: pepe\n
sessionKey: 5000\n
\n

Mensaje: loginFAIL
Sentido de la comunicación: Directorio → Cliente
Descripción: En caso de que la operación login resulte en un error, debido a, por ejemplo, que ya existe otro usuario con ese nombre, el Directorio debe enviar al cliente un mensaje expresando que se éste ha producido. Debido a que la creación de usuario con su correspondiente nickname ha resultado en error, no se le vinculará ningún sessionKey.
Ejemplo:

operation: loginFAIL\n
nickname: pepe\n
sessionKey: -1\n
\n

Mensaje: userlist
Sentido de la comunicación: Cliente → Directorio
Descripción: Este comando permite mostrar al Directorio cada uno de los usuarios que están registrados en el servidor del Directorio. El usuario, una vez registrado (no puede hacer uso de este comando sin estar registrado) tan sólo debe preguntar al Directorio por la lista.
Ejemplo:

operation: userlist\n
\n

Mensaje: userlist
Sentido de la comunicación: Directorio → Cliente
Descripción: Esta es la respuesta del Directorio al cliente por el comando userlist. El Directorio debe ser capaz de enlistar cada uno de los usuarios conectados y enviárselo al cliente. La longitud en líneas de la respuesta dependerá de la cantidad de usuarios registrados en el servidor del Directorio.
Ejemplo:

operation: registeredUsers\n
sessionKey: 5000\n
user: pepe\n
user: juan\n
…
user: alicia\n
\n

Mensaje: logout
Sentido de la comunicación: Cliente → Directorio
Descripción: Un usuario enviaría este mensaje al directorio. Este mensaje es necesario para que cualquier usuario que esté conectado al directorio cierre su sesión.
Ejemplo:

operation: logout\n
sessionKey: 5000\n
\n

Mensaje: logoutOK
Sentido de la comunicación: Directorio → Cliente
Descripción: Cuando el directorio reciba el mensaje, tiene que comprobar que el usuario está registrado en el directorio. Si está registrado, eliminará su nickname y clave de sesión asociada en el directorio.
Ejemplo:

operation: logoutOK\n
\n

Mensaje: logoutFAIL
Sentido de la comunicación: Directorio → Cliente
Descripción: En el caso de que el usuario no esté registrado deberá mandar un mensaje loginFail informando de que este usuario (sessionKey) no existe.
Ejemplo:

operation: logoutFAIL\n
\n

Formato de los mensajes del protocolo de transferencia de ficheros

Ahora os detallaremos todos aquellos mensajes enviados entre un cliente y un servidor de ficheros peer. De momento, solo se comunicarán cuando el cliente quiera descargar un fichero del servidor y este le conteste afirmativamente o con un mensaje de error. Como ocurría con en la comunicación entre el cliente y el Directorio, el servidor únicamente se comunicará con el cliente para enviarle una respuesta, ya sea afirmativa o negativa.

Estos mensajes pueden tener un tamaño distinto, dependiendo de la información que decidamos enviar en cada mensaje. Algunas veces, nos interesará enviar un mensaje de control (un mesaje simple con un único campo de un byte para el opcode o código de operación) Otras, sin embargo, enviaremos más que eso. Por ejemplo, el envío del hash de un fichero será pieza clave de nuestro protocolo de transferencia de ficheros. Por ello, el código de operación se encontrará acompañado de uno o más parámetros. El lector de esta memoria podrá comprobar que, en algunos casos, hemos enviado información que no resulta importante o necesaria, pero queríamos asegurarnos de poder tener al alcance todos los datos posibles.

El formato será el siguiente para los mensajes con un solo parámetro (el código de operación):

Opcode (1 byte)

En el resto de casos, se añadirán más parámetros, en función de lo se exija para una buena comunicación. En cuanto a los bytes que requiere cada parámetro, dependerá del tamaño de lo que queramos enviar. No es lo mismo, un trozo de hash de un fichero, que el fichero en sí:

Opcode (1 byte)	**Parámetro 1 (m bytes)**

Opcode (1 byte)	**Parámetro 1 (m bytes)**	**Parámetro n (m bytes)**

Tipos y descripción de los mensajes

Mensaje: InvalidOpCode (opcode=0)
Sentido de la comunicación: Servidor de ficheros → Cliente y Cliente → Servidor de ficheros
Descripción: Este mensaje puede ser enviado en ambos sentidos y se usa cuando se envía previamente un opcode inválido debido a que no existe o no hay ninguno asignado a ese byte.
Ejemplo:

Opcode (1 byte)
0

Mensaje: DownloadFrom (opcode=1)
Sentido de la comunicación: Cliente → Servidor de ficheros
Descripción: Este mensaje es enviado por el cliente a un servidor de ficheros cuando quiera obtener un fichero. Para ello, deberá enviar un array de bytes, el cual contendrá el hash o trozo de hash desde el cual partimos.
Ejemplo:

Opcode (1 byte)	Hash (n bytes)
1	**9348afj3**

Mensaje: DownloadOK (opcode=2)
Sentido de la comunicación: Servidor de ficheros → Cliente
Descripción: Este mensaje es enviado por el servidor de ficheros a un cliente cuando la operación anterior (DownloadFrom) ha sido aceptada. El servidor devolverá el hash completo, ya que primero lo habrá reconstruido. Seguidamente, mandará los datos del fichero (también en un array de bytes). Si el tamaño del fichero es demasiado grande, se necesitará mandar más de un DownloadOK (hasta que no se envíen todos los datos). En los sucesivos, también se enviará el código de operación y el hash completo.
Ejemplo:

Opcode (1 byte)	Hash (n bytes)	Datos (n bytes)
2	**9348afj33429wfjl…**	**fichero**

Mensaje: FileNotFound (opcode=3)
Sentido de la comunicación: Servidor de ficheros → Cliente
Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) de fichero para indicar que no es posible encontrar el fichero con la información proporcionada en el mensaje de petición de descarga.
Ejemplo:

Opcode (1 byte)	Hash (n bytes)
3	**9348afj3**

Mensaje: AmbiguousHash (opcode=4)
Sentido de la comunicación: Servidor de ficheros → Cliente
Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) de fichero para indicar que el trozo de hash que se le ha enviado tiene coincidencia con más de uno. Esto quiere decir, que el servidor no es capaz de acertar el fichero al que se refiere el cliente, ya que se le envió un trozo de hash que pertenece a más de un hash del total.
Ejemplo:

Opcode (1 byte)
4

Mensaje: Test (opcode=9)
Sentido de la comunicación: Ambos sentidos
Descripción: Este mensaje lo envía el par servidor de ficheros al par cliente (receptor) de fichero y viceversa. Lo usamos al principio en los casos en los que tuviésemos dudas de cómo implementar un nuevo mensaje, o simplemente queríamos probar algo nuevo sin tocar los mensajes que ya funcionaban. Podríamos haberlo eliminado y no incluirlo en esta memoria, pero consideramos preciso que conozcáis los procedimientos que llevamos a cabo.
Ejemplo:

Opcode

(1 byte)
9

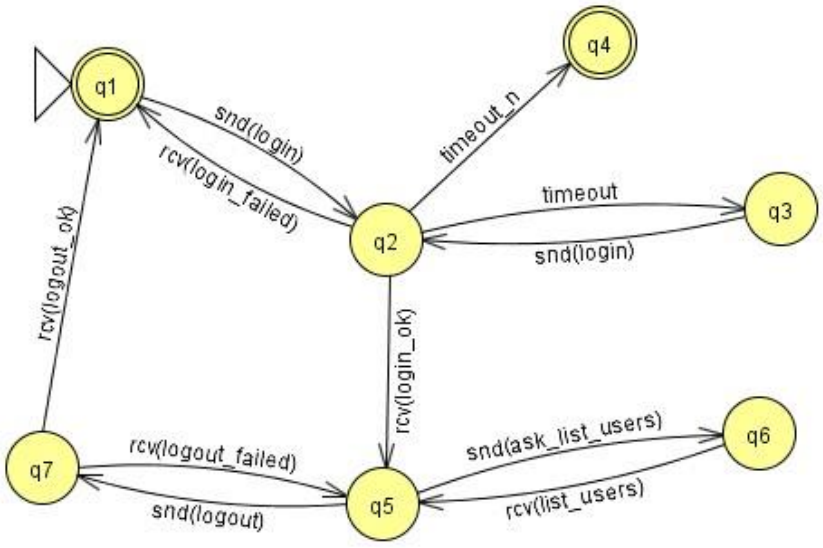
Autómatas de protocolo

Hemos ideado las siguientes restricciones:

- Un cliente no podrá enviar ningún mensaje específico hasta que inicia conexión con el Directorio (login) y reciba una respuesta exitosa (loginOK). Estos mensajes incluyen: userlist, fgserve, bgserve y downloadfrom.
- El usuario no podrá terminar hasta que cierre la conexión con el Directorio (logout) y reciba una respuesta exitosa (logoutOK).
- Un usuario no podrá actuar de peer hasta que no envíe un mensaje pidiéndolo (fgserve).

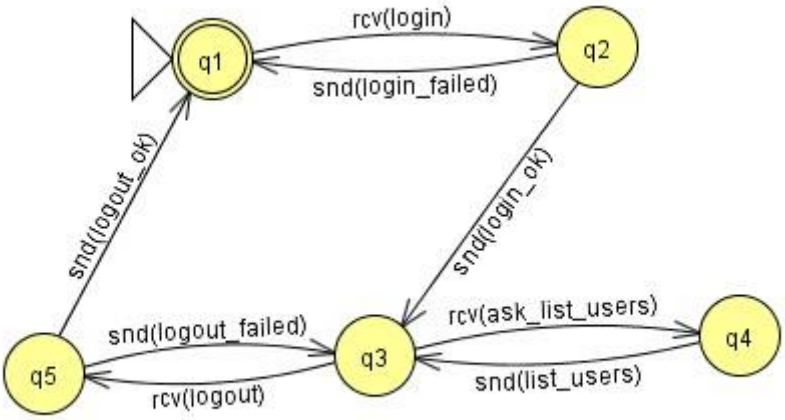
Autómata rol cliente de directorio

Desde el punto de vista del cliente, este podrá hablar con el directorio, además de convertirse en un servidor de ficheros



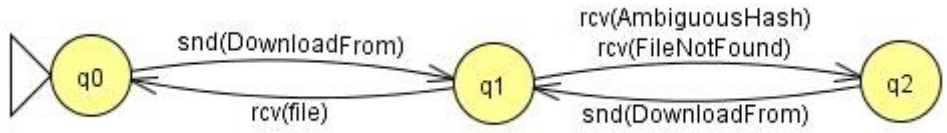
Autómata rol servidor de directorio

Desde el punto de vista del del directorio, su único propósito es controlar el servicio que presta a los clientes.



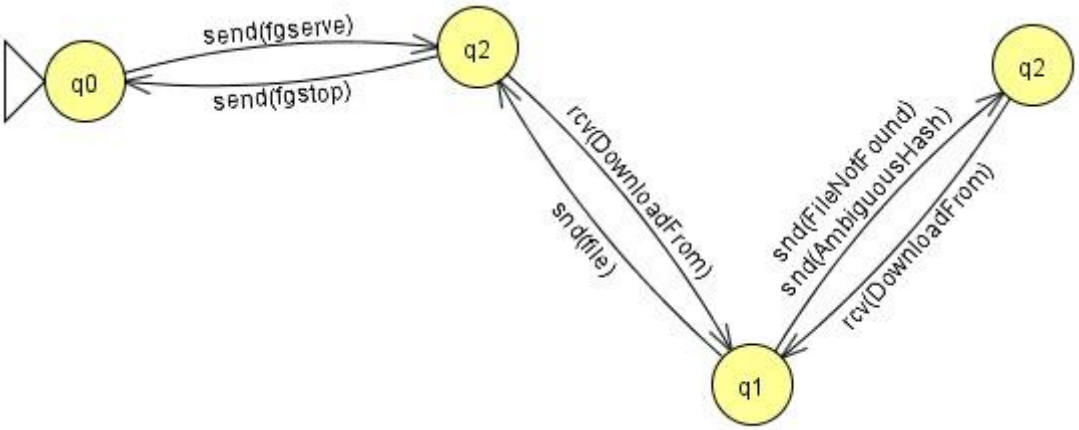
Autómata rol cliente de ficheros

Aquí, el cliente quiere obtener un fichero de alguno de los servidores.



Autómata rol servidor de ficheros

Una vez que se ha convertido el cliente en un servidor, ya podrá enviar ficheros al resto de clientes.



Ejemplo de intercambio de mensajes

Para terminar, os dejamos algunos ejemplos que ponen en situación de cada una de las acciones que pueden ocurrir entre el cliente y el directorio o el servidor de ficheros. Esperamos que os sean de utilidad.

DIRECTORIO

```
Probability of corruption for received datagrams: 0.0
Directory starting...
Waiting to receive datagram...
```

CLIENTE 1

```
NanoFiles shell
For help, type 'help'
(nanoFiles@nf-shared) help
List of commands:
quit -- quit the application
login -- log into <directory> using <nickname>
userlist -- show list of users logged into the directory
filelist -- show list of files tracked by the directory
myfiles -- show contents of local folder (files that may be served)
fgserve -- run file server in foreground (blocking)
bgserve -- run file server in background (non-blocking)
publish -- publish list of served files to the directory
stopserver -- stop file server running in background
downloadfrom -- download from server given by <nickname> the file identified by <hash>
search -- show list of servers sharing the file identified by <hash>
download -- download the file identified by <hash> from all available server(s)
logout -- log out from the current directory
sleep -- sleep during <num> seconds
help -- shows this information
```

CLIENTE 1

```
(nanoFiles@nf-shared) fgserve
*You cannot run this command because you are not logged into the directory
```

CLIENTE 1

```
(nanoFiles@nf-shared) login localhost mario
Created UDP socket at local addresss /[0:0:0:0:0:0:0:0]:51390
DirMessage read from socket:
operation:loginOK
nickname:mario
sessionKey:6006


mario has logged in successfully (6006).
Login was successful with 6006 key
(nanoFiles@nf-shared)
```

DIRECTORIO

```
Directory received datagram from /127.0.0.1:51390 of size 32 bytes
Contents interpreted as 32-byte String: "operation:login
nickname:mario

"
DirMessage read from socket:
operation:login
nickname:mario


operation:loginOK
nickname:mario
sessionKey:6006

Waiting to receive datagram...
```

CLIENTE 1

```
(nanoFiles@nf-shared) userlist
DirMessage read from socket:
```

```
operation:registeredUsersResp
user:mario


user:mario
```

DIRECTORIO

```
Directory received datagram from /127.0.0.1:51390 of size 43 bytes
Contents interpreted as 43-byte String: "operation:registeredUsers
sessionKey:6006

"
DirMessage read from socket:
operation:registeredUsers
sessionKey:6006


operation:registeredUsersResp
user:mario

Waiting to receive datagram...
```

CLIENTE 1

```
(nanoFiles@nf-shared) fgserve

Server is listening on port 10000
Server socket is running on 0.0.0.0/0.0.0.0:10000
Write the command fgstop to stop the server
fgstop
NFServerSimple stopped. Returning to the nanoFiles shell...

(nanoFiles@nf-shared) myfiles
List of files in local folder:
Name Size Hash
Captura de pantalla 2024-01-24 204322.png 20657 5870dafa799721697a144c8bb845d6e83ce80685
prueba.txt 18 a609295021744033e92fd3fb5c758fc7f19ed697
```

CLIENTE 2

```
(nanoFiles@nf-shared) downloadfrom localhost:10000 9502174403 prueba.txt
*You cannot run this command because you are not logged into the directory
(nanoFiles@nf-shared) login localhost ana
Created UDP socket at local addresss /[0:0:0:0:0:0:0:0]:52146
DirMessage read from socket:
operation:loginOK
nickname:ana
sessionKey:5516


ana has logged in successfully (5516).
Login was successful with 5516 key
```

DIRECTORIO

```
Directory received datagram from /127.0.0.1:52146 of size 30 bytes
Contents interpreted as 30-byte String: "operation:login
nickname:ana

"
DirMessage read from socket:
operation:login
nickname:ana


operation:loginOK
nickname:ana
sessionKey:5516

Waiting to receive datagram...
```

CLIENTE 2

```
(nanoFiles@nf-shared) downloadfrom localhost:10000 9502174403 prueba.txt
```

```
File successfully written: C:\Users\TESTER\git\rc-2324\nanoFilesP2Palumnos\prueba.txt
Same hash!
File downloaded successfully
```

CLIENTE 1

```
New client connected: /127.0.0.1:50111
The download of the file was successful
```

Conclusiones

Todo el mundo gira sobre las redes de comunicaciones. Hoy en día necesitas un móvil o acceso a Internet desde un dispositivo para poder hablar con alguien o mandarle la última foto que os echasteis juntos. Cuando usamos una aplicación para pedirle ayuda a nuestros padres, lo estamos haciendo a través de un modo no físico. ¿Somos realmente conscientes de cómo se produce esa comunicación internamente?

Gracias a lo nuevo aprendido en las clases de teoría, hemos podido saber de primera mano el funcionamiento de las redes de comunicaciones. Los protocolos UDP (User Datagram Protocol) y TCP (Transmission Control Protocol), tan básicos para el intercambio de datos entre dos hosts, son implementados de primera mano. Además, somos capaces de interiorizar la diferencia entre los mensajes ASCII (American Standard Code for Information Interchange) y los binarios. Debo puntualizar, la compresión interna de la estructura de estos últimos nos resultaba un tanto confuso al principio, pero una vez le dedicamos bastante más tiempo, fuimos capaces de entenderlo a la perfección.

En conclusión, nos quedamos satisfechos de todo lo aprendido durante la realización de la práctica. Podemos asegurar con firmeza, que estamos ante uno de los proyectos más importantes de la carrera, mas se encuentra en su segundo curso. Resulta un tanto desafiante, aunque qué no lo es en nuestra carrera. La curva de aprendizaje que presenta es de la más duras. Sin embargo, con el devenir de las semanas, acudiendo a clase y escuchando con atención al profesor, uno puede mentalizarse del funcionamiento interno de la práctica.

El lenguaje Java sigue siendo, hoy en día, muy utilizado (en todos los ámbitos). Es por eso por lo que debemos aprender muy concienzudamente todos los aspectos que desentraña. Creemos que esta asignatura es la indicada para dar ese pequeño salto, y pasar de ser simples programadores a ingenieros.