Diseño Nanochat

Pablo Asensio Muñoz y Germán Gil Planes / GRUPO PCEO / DNIs: 49696207F ; 49246858V

ÍNDICE

1. Introducción……..……3
2. Formato de los mensajes del protocolo de comunicación con el Directorio……...……3
   2.1. Formato de mensajes……..…….3
   2.2. Tipos y descripción de los mensajes..…….4
3. Formato de los mensajes del protocolo de comunicación entre Cliente de chat y Servidor de Chat……..…….6
   3.1. Formato de mensajes……..…….6
   3.2. Tipos y descripción de los mensajes……7
4. Autómatas de protocolo……...10
5. Ejemplo(s) de intercambio de mensajes……..13
6. Implementaciones extra y cambios con respecto al diseño entregado el 29/03/2022…… 14
7. Conclusiones……...15

1. Introducción

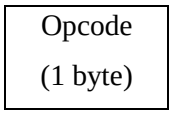
A lo largo de este documento vamos a hablar sobre el diseño de los protocolos necesarios para la implementación de cada parte del NanoChat. Estas partes son un servidor de directorio, un servidor de chat y un cliente de chat, de los cuáles el cliente y servidor de chat intercambian mensajes usando el protocolo a nivel de transporte llamado TCP con mensajes codificados en ASCII con el formato field:value. Por otro lado, tanto el cliente como el servidor de chat se comunican con el servidor de directorio mediante protocolos UDP y mensajes codificados en binario (en el apartado 2 describimos con más detalle el formato de estos mensajes). El funcionamiento de cada una de las partes esta descrito mediante autómatas (descrito con detalle en el apartado 4) y finalmente mostramos un pequeño ejemplo de cómo funcionan las comunicaciones entre cada una de las partes del NanoChat.

2. Formato de los mensajes del protocolo de comunicación con el Directorio

Hemos mencionado anteriormente que los mensajes entre el directorio y el cliente o el servidor se realiza mediante mensajes codificados en binario y con protocolo UDP. Debido a que este protocolo no es fiable, necesitamos mensajes de confirmación para asegurar que toda la información ha llegado. En los siguientes apartados describiremos el formato y el contenido de estos mensajes.

2.1. Formato de los mensajes

Como ya hemos visto en las sesiones de prácticas (en concreto en la sesión 3), hay unos formatos de mensajes binarios más o menos establecidos. Basándonos en ellos vamos a describir el formato de los mensajes que hemos usado nosotros:



- Control

- OneParameter

Opcode (1 byte)	Parameter (1 bytes)

- TwoParameters

Opcode (1 byte)	ByteParameter (1 byte)	Parameter (4 bytes)

- FiveParameters

Opcode (1 byte)	IP1 (1 byte)	IP2 (1 byte)	IP3 (1 byte)	IP4 (1 byte)	Puerto (4 bytes)

2.2. Tipos y descripción de los mensajes

- Mensaje: **RegistrationRequest** (Opcode = 1)

Formato: TwoParameters

Sentido: Servidor de chat → Directorio

Descripción: Este mensaje solicita al directorio que registre un servidor de chat con nuestra IP y el puerto que se pasa como parámetro con el ID indicado de protocolo.

Ejemplo:

Opcode (1 byte)	IDprotocol (1 byte)	Port (4 bytes)
1	67	6969

- Mensaje: **OkRegistration** (Opcode = 2)

Formato: Control

Sentido: Directorio → Servidor de chat

Descripción: Este mensaje confirma que el registro del servidor se ha realizado con éxito.

Ejemplo:

Opcode (1 byte)
2

- Mensaje: **QueryProtocol** (Opcode = 3)

Formato: OneParameter

Sentido: Cliente de chat → Directorio

Descripción: Este mensaje sirve para consultar al directorio la dirección IP y puerto del servidor con la ID indicada de protocolo.

Ejemplo:

Opcode (1 byte)	IDprotocol (1 byte)
3	47

- Mensaje: **OkQuery** (Opcode = 4)

Formato: FiveParameters

Sentido: Directorio → Cliente de chat

Descripción: Este mensaje confirma al cliente que su solicitud se ha encontrado con éxito y envía la información requerida: la IP y el puerto del servidor.

Ejemplo:

Opcode (1 byte)	IP1 (1 byte)	IP2 (1 byte)	IP3 (1 byte)	IP4 (1 byte)	Puerto (4 bytes)
4	127	0	0	1	6969

- Mensaje: **Fail** (Opcode = 5)

Formato:Control Sentido: Directorio → Servidor/Cliente de chat

Descripción: Envía un mensaje que notifica que la operación tuvo algún error.

Ejemplo:

Opcode (1 byte)
5

3. Formato de los mensajes de protocolo de comunicación entre Cliente de chat y Servidor de Chat

3.1. Formato mensajes

Haciendo los cálculos que se requieren en la práctica 5 tenemos de identificador de protocolo es impar ((49696107+49246858) mod 2 = 1) luego usamos mensajes en formato “field:value” que tienen la siguiente estructura:

- Mensaje entre el cliente y el servidor

Formato: SimpleMessage

operation:

xxxx\n

\n

- Mensajes en una sala de chat

Formato: RoomMessage

operation: xxxx\n

name: xxxx\n

\n

- Mensaje del servidor al cliente para mostrarle las salas

Formato: RoomListMessage

operation:xxxx\n

name:xxxx\n

lastMessage:xxxx\n

members:xxxx\n

3.2. Tipos y descripción de los mensajes

- Mensaje: **nick**

Formato: RoomMessage

Sentido: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para que el cliente se registre en un servidor con un nick cuyo valor viene dado en el campo “name”.

Ejemplo: operation: nick

name: Pepe

\n

- Mensaje: **DuplicatedNick**

Formato: SimpleMessage

Sentido: Servidor de chat → Cliente de chat

Descripción: Este mensaje se envía en respuesta a un mensaje de “RegisterNick” para indicarle que no se ha podido realizar el registro del cliente porque el nick dado ya estaba registrado por otro cliente de chat.

Ejemplo: operation: DuplicateNick\n

\n

- Mensaje: **OkNick**

Formato: SimpleMessage

Sentido: Servidor de chat → Cliente de chat

Descripción: Este mensaje se envía en respuesta a un mensaje de “RegisterNick” para indicarle que se ha podido realizar el registro con éxito porque el nick dado no estaba registrado por otro cliente de chat.

Ejemplo: operation: OkNick\n

\n

- Mensaje: **roomlist**

Formato: SimpleMessage

Sentido: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para que el cliente solicite los nombres de las salas del servidor.

Ejemplo: operation: roomlist\n

\n

- Mensaje: **ListRoom**

Formato: RoomListMessage

Sentido: Servidor de chat→ Cliente de chat

Descripción: Este mensaje muestra toda la lista de salas disponibles con sus respectivas informaciones.

Ejemplo:

operation:ListRoom\n

name: RoomA\n

lastMessage: `Last message: Fri Jun 17 12:26:51 CEST 2022`\n

members: Germán Pablo\n

\n

- Mensaje: **NoRoom**

Formato: SimpleMessage

Sentido: Servidor de chat → Cliente de chat

Descripción: Este mensaje sirve para notificar que no hay una sala con el nombre solicitado.

Ejemplo: operation: NoRoom\n

\n

- Mensaje: **enter**

Formato: RoomMessage

Sentido: Cliente de chat → Servidor de chat

Descripción: Este mensaje sirve para que el cliente solicite al servidor la entrada a la sala indicada.

Ejemplo: operation: enter

name: roomA

\n

- Mensaje: **send**

Formato: RoomMessage

Sentido de la comunicación: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para escribir un mensaje del cliente al servidor, cuyo valor viene dado en el campo “name”.

Ejemplo: operation: send

name: Hola buenos días

\n

- Mensaje: **private**

Formato: RoomMessage

Sentido de la comunicación: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para escribir un mensaje del cliente al servidor, cuyo valor viene dado en el campo “name” y cuya recepción es el cliente del campo “who”

Ejemplo: operation: private

who: pepe

name: Hola buenos días

\n

- Mensaje: **ExitRoom**

Formato: SimpleMessage

Sentido: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para que el cliente salir de la sala.

Ejemplo: operation: ExitRoom\n

\n

- Mensaje:**OkExit**

Formato: SimpleMessage

Sentido: Servidor de chat→ Cliente de chat

Descripción: Este mensaje sirve para informar al cliente de que ha salido de la sala.

Ejemplo: operation: OkExit\n

\n

- Mensaje: **noExit**

Formato: SimpleMessage

Sentido: Servidor de chat→ Cliente de chat

Descripción: Este mensaje sirve para informar al cliente de que no ha salido de la sala.

Ejemplo: operation: noExit\n

\n

- Mensaje: **info**

Formato: SimpleMessage

Sentido: Cliente de chat→ Servidor de chat

Descripción: Este mensaje sirve para que el cliente reciba información sobre la sala.

Ejemplo: operation: info

\n

- Mensaje: **RoomInfo**

Formato: RoomListMessage

Sentido: Servidor de chat→ Cliente de chat

Descripción: Este mensaje contiene la información de la sala en la que el cliente está metido actualmente.

Ejemplo:

operation:ListRoom\n

name: RoomA\n

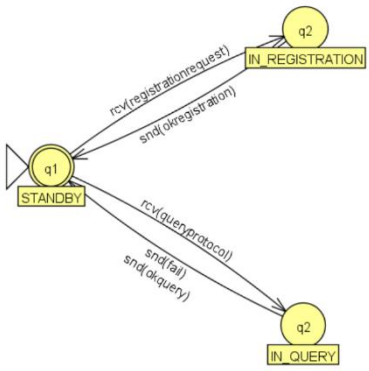
lastMessage: `Last message: Fri Jun 17 12:26:51 CEST 2022`\n

members: Germán Pablo\n

\n

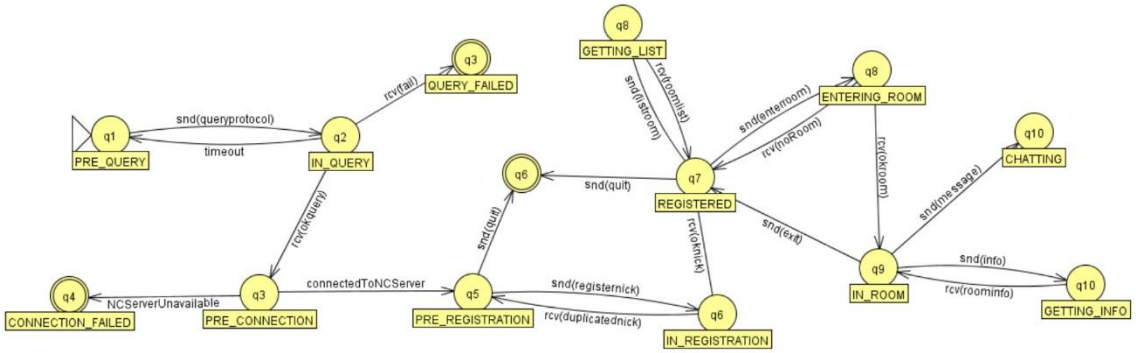
4. Autómatas de protocolo

Autómata Directorio



En q1 el directorio se encuentra en un estado de escucha (reposo o standby) a la espera de recibir una consulta de servidor por parte del cliente o una solicitud para registrar un servidor de chat. Tras eso, les devuelve a cada uno sus correspondientes confirmaciones y vuelve al estado de q1.

Autómata Cliente de chat (unificado TCP/UDP)



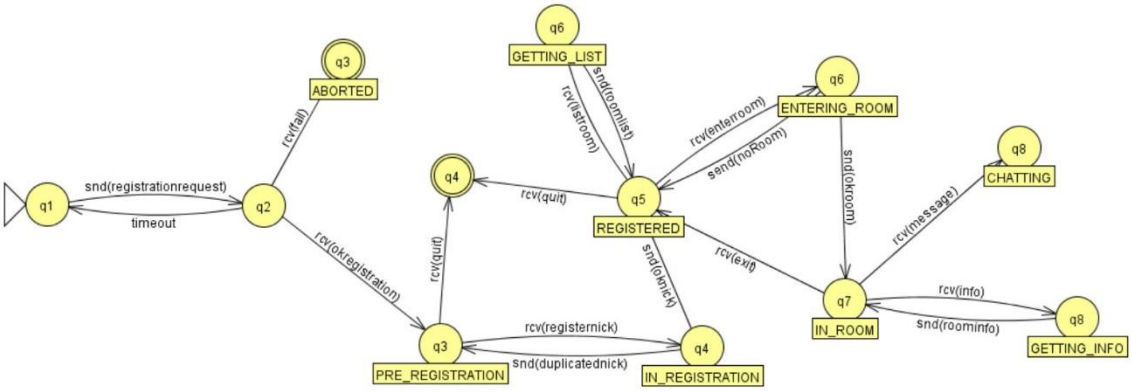
En q1 el cliente envía al directorio un query. Como puede ser que la consulta se pierda, es posible que se lo tenga que volver a enviar tras un timeout. Una vez que le llega la consulta al directorio en q2, este puede devolverle un fail o confirmarle que la consulta ha tenido éxito (q3).

Después puede ser que la conexión tenga éxito (q5) o no(q4). Si logra conectarse con el servidor, el cliente podrá enviarle un nick para registrarse(q6). El servidor le responderá con duplicatedNick hasta que le entregue un nick válido. Cuando esto ocurra, le mandará un okNick y el cliente se habrá registrado en el servidor con éxito(q7). Cabe destacar que, en cualquier momento desde que el cliente está conectado con el servidor podrá ejecutar la instrucción quit y acabará en q6.

Desde q7 el cliente podrá solicitar una lista de las salas disponibles y también podrá solicitar entrar en alguna (ambos mensajes deben ser respondidos por el servidor para informarle al cliente de cómo ha ido su solicitud).

Si la solicitud de entrar en una sala ha tenido éxito y el servidor no le ha respondido roomBanned o noRoom, recibirá una confirmación y entrará en una sala (q9). Desde la sala el cliente podrá enviar mensajes, solicitar información acerca de la sala y salirse de ella. Todos estas solicitudes deben ser confirmadas por el servidor también.

Autómata Servidor de chat (unificado TCP/UDP)



Este autómata es prácticamente similar al del cliente ya que, al fin y al cabo, el servidor y el cliente se comunican entre ellos de tú a tú y, a partir de q3, el autómata es idéntico. Tan sólo cambia que los mensajes que uno enviaba ahora los recibe porque lo estamos contemplando desde la perspectiva del servidor, y no desde la del cliente. La variación con respecto al autómata anterior es, por tanto, sólo lo correspondiente a los primeros estados (la comunicación del servidor con el directorio).

En q1 el servidor le manda la solicitud de registro y tendrá que esperar a que el directorio se lo confirme. Es posible que sea después de uno o varios timeouts. Desde q2, el mensaje que le puede devolver es fail o okRegistration. Una vez que le confirma el registro, el servidor crea un hilo para un nuevo cliente y pasa a q3, donde comenzaría la parte del autómata que acabamos de comentar.

5. Ejemplo(s) de intercambio de mensajes

Cliente N: n+ / Servidor: -

1+ snd(RegisterNick)(Nick pedro)[PreRegistration]

- rcv(RegisterNick)

- snd(OkayNick)

1+ rcv(OkNick)[Resgitrated]

2+ snd(RegisterNick) (nick pablo) [PreRegistration]

- rcv(RegisterNick)

- snd(DuplicatedNick)

2+ rcv(DuplicatedNick) [PreRegistration]

2+ snd(Help) [PreRegistration]

- rcv(Help) - snd(SendedHelp)

2+ rcv(SendedHelp) [PreRegistration]

2+ snd(RegisterNick) (nick jose) [PreRegistration]

- rcv(RegisterNick)

- snd(OkNick)

2+ rcv(OkNick) [PreRegistrated]

2+ snd(enterRoom) [Registrated]

- rcv(enterRoom)

-snd(InRoom)

2+ rcv(InRoom) [IN_ROOM]

1+ snd(enterRoom) [Registrated]

- rcv(enterRoom)

-snd(InRoom)

1+ rcv(InRoom) [IN_ROOM]

2+ snd(ExitRoom) [IN_ROOM]

- rcv(ExitRoom)

-snd(OkExit)

+2 rcv(OkExit) [Registrated]

6. Implementaciones extra y cambios con respecto al diseño entregado el 29/03/2022

A continuación, se mostrarán todos los cambios que hemos visto conveniente hacer:

- En las instrucciones de “info” y “roomlist”, encargadas de mostrar la información de las salas existentes. Antes la definíamos como un “RoomMessage”. Ahora lo hemos adecuado como se debe, incorporando un nuevo tipo de mensaje, “RoomListMessaje”.
- Hemos eliminado “RoomBanned”, ya que, en el proyecto final, no se incorpora la posibilidad de que un cliente pueda estar baneado.
- Hemos visto que los mensajes "Help" y "SendedHelp" no eran necesarios, ya que el contenido de la ayuda lo tenemos directamente en el código fuente del cliente de chat.
- Hemos eliminado el “okMessage”. Lo que hacía era confirmar a un cliente de que su mensaje había sido enviado con éxito. Sin embargo, esto no tiene sentido porque un cliente debería estar colgado esperando la confirmación de cada mensaje que envía. Lo mismo ocurre con “okExit” y “noExit”. Ninguno de estos tres mensajes era adecuado implementarlos.
- También hemos obviado el comando “quit”, ya que no es un mensaje como tal, sino que el cliente simplemente cierra su socket.

**** TODO ESTO HA SIDO MODIFICADO TAMBIÉN DE LOS AUTÓMATAS PRESENTADOS EN UN PRINCIPIO****

Además, tenemos 3 mejorar implementadas (2 de medio punto y 1 de un punto):

- Notificar a los demás usuarios de la sala las entradas/salidas de otros usuarios en las salas conforme se vayan produciendo. Muestra de ejemplo:

```
(nanoChat) enter RoomA
* You have entered room : RoomA
(nanoChat-room) * Message received from server...
Germán: ** Ha entrado en la sala **
(nanoChat-room) exit
* Your are out of the room
```

- Posibilidad de mandar mensajes privados.

```
(nanoChat-room) private Pablo hola
```

- Posibilidad de renombrar una sala:

```
(nanoChat-room) rename RoomB
(nanoChat-room) * Message received from server...
La sala ha cambiado de nombre
```

7. Conclusiones

En la realización de este proyecto, hemos comprendido a la perfección el funcionamiento de los protocolos UDP y TCP, por lo que se puede decir que la práctica ha cumplido con sus objetivos iniciales. Además, nos ha servido para aprender técnicas nuevas en la programación que nos han ayudado a menudo para localizar errores. Por ejemplo, el uso de breakpoints ha sido nuestro nuevo descubrimiento y nuestra herramienta más fiable. Por lo tanto, podemos decir que, además de ayudarnos a comprender la asignatura, nos ha ayudado a progresar en Java, en Eclipse y en la programación, en general.