PRÁCTICA 1: DISEÑO E IMPLEMENTACIÓN DE UN ESQUEMA EDITOR/SUSCRIPTOR CON SOCKETS

SISTEMAS DISTRIBUIDOS

Curso 2015-2016

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

JAVIER PRIETO CEPEDA: **100 307 011**

PEDRO ROBLEDO QUINTERO: **100 282 657**

ÍNDICE

1 Introducción

En este documento se procede a la explicación de práctica 1, *"Diseño aplicación distribuida con sockets"* de la asignatura Sistemas Distribuidos del grado en Ingeniería Informática de la Universidad Carlos III de Madrid. La realización de la misma la han llevado a cabo los alumnos Javier Prieto Cepeda y Pedro Robledo Quintero.

En primer lugar, se realizará una breve descripción del sistema editores/suscriptores, que nos servirá para explicar las decisiones de diseño tomadas a cabo para la implementación del sistema.

Una vez descrito el sistema, se realizará una descripción de la implementación llevada a cabo en los 3 actores que componen la aplicación.

En tercer lugar, se expondrá la batería de pruebas llevada a cabo para la verificación de la aplicación.

Por último, se detalla la forma de compilar y obtener los distintos ejecutables de la aplicación.

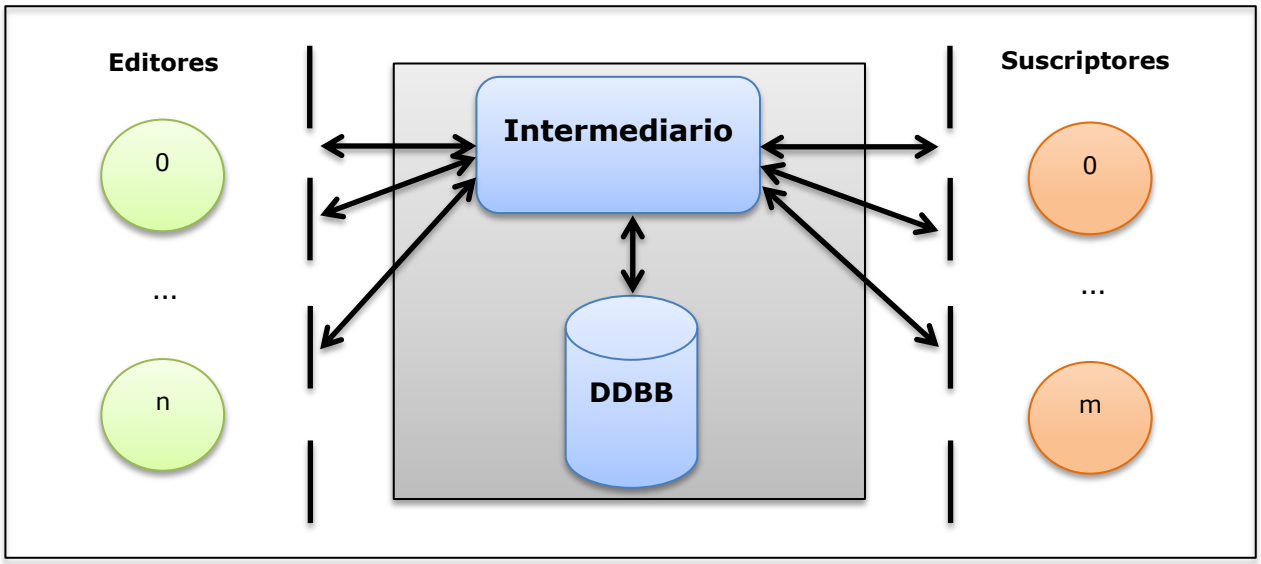
2 Diseño de la aplicación distribuida

En este apartado, se va a proceder a explicar el esquema del sistema Editores/Suscriptores que conforma la aplicación.

El sistema, constará de 3 tipos diferentes de actores: editores, intermediario y suscriptores.

- **Editores:** Son los encargados de generar mensajes de un tema definido en su lanzamiento. Estos mensajes, son enviados al intermediario para que gestione su difusión.
- **Intermediario:** Es el encargado de mediar en la comunicación entre editores y suscriptores. Esta mediación, se realiza de forma que cuando un suscriptor desea darse de alta en un tema para la recepción de sus mensajes, el intermediario se encarga de su registro en una base de datos, a la que tan solo él tendrá acceso. Cuando un editor envíe un nuevo mensaje de un tema, el intermediario lo recibe, busca los suscriptores que están suscritos al tema, y les envía el mensaje nuevo. De esta forma, entre editores y suscriptores, no hay comunicación directa.
- **Suscriptores:** Son actores que se suscriben a temas para la recepción de los nuevos mensajes asociados a ellos. De igual forma, un suscriptor, puede darse de baja de la suscripción de un tema una vez que no quiere seguir recibiendo nuevos mensajes de él.

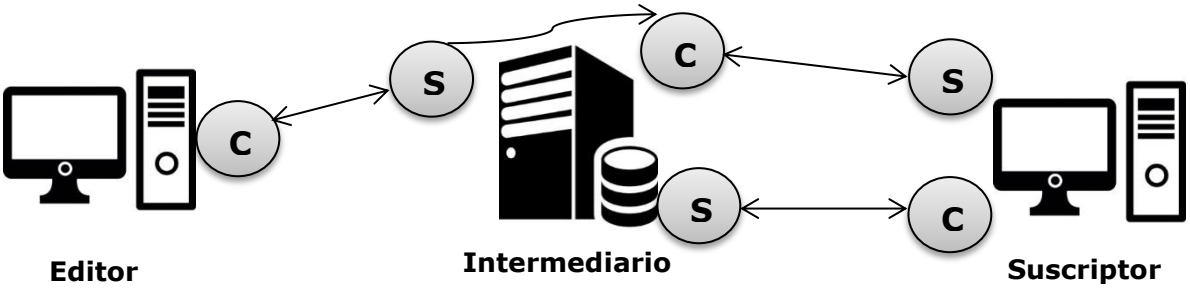
El esquema a alto nivel de la aplicación, sería el siguiente:



Una vez realizado éste análisis, podemos concluir que el sistema, consta de una serie de componentes, activos y pasivos, que permiten la correcta ejecución de la aplicación. Los componentes que son activos, son aquellos que inician una acción, mientras que los pasivos, son quienes la reciben. La aplicación está compuesta por los siguientes componentes:

- **Cliente-Editor:** Los editores, actúan como clientes en la comunicación que realizan con el intermediario, puesto que únicamente se encargan de emitir mensajes.
- **ServidorEditores-Intermediario:** El intermediario, actúa como servidor en la comunicación con los editores. Esto se debe, a que se encuentra a la espera de recibir un inicio de comunicación por parte de algún editor, para la gestión de un nuevo mensaje.
- **ServidorSuscriptores-Intermediario:** El intermediario, actúa como servidor en la comunicación con los suscriptores. Esto se debe, a que se encuentra a la espera de recibir un inicio de comunicación por parte de algún suscriptor, para la gestión de suscripciones.
- **Cliente-Intermediario:** El intermediario, además de cómo servidor, actúan como cliente, puesto que cada vez que recibe un nuevo mensaje por parte de los editores, se encarga de difundirlo a aquellos suscriptores suscritos al tema del nuevo mensaje.
- **Cliente-Suscriptor:** Los suscriptores, actúan como clientes en la comunicación que realizan con el intermediario para la gestión de suscripciones, puesto que son la entidad activa de la comunicación.
- **Servidor-Suscriptor:** Los suscriptores, además, actúan como servidor, puesto que una vez suscritos a un tema, se encuentran esperando nuevos mensajes de los temas suscritos.

El esquema resultante de la aplicación a bajo nivel, es el siguiente:



Por una parte, se ha decidido que el cliente del editor, sea un único proceso por editor. El servidor que espera peticiones por parte de los editores en el intermediario, será un proceso, que genera un *thread* por cada petición recibida, para su tratamiento. Una vez recibida la petición y finalizada la comunicación con el editor, este *thread*, deja de actuar como servidor, y pasa actuar como cliente de aquellos suscriptores que se encuentran suscritos al tema enviado por el editor. De este modo, el intermediario, cuenta con un servidor concurrente para las peticiones de los editores.

Por otra parte, se ha decidido que el suscriptor, cuente con 2 procesos. Por un lado, cuenta con un proceso cliente, que se encarga de realizar peticiones para la gestión de suscripciones con el intermediario. Cuando el suscriptor se suscribe a un tema por primera vez, crea un *thread* como servidor, que se encargará de escuchar todos los mensajes enviados por el intermediario de los temas suscritos. Cuando el suscriptor, deja de estar suscrito a temas, este *thread* servidor, se destruye. De este modo, se tiene que cada suscriptor, únicamente cuenta con un proceso cliente, y un proceso servidor siempre que el número de suscripciones sea mayor que 0.

2.1 Protocolo de la aplicación

Puesto que se desea que el intercambio de información entre los distintos componentes de la aplicación, se realice de forma segura, y conforme a los requisitos establecidos en el enunciado de la práctica, se ha decidido utilizar el protocolo de red TCP.

El protocolo de mensajes, queda descrito en el enunciado de la práctica, por lo que no se ha creído necesario incluirlo en este documento.

3 Descripción del código

En este apartado, se va a proceder a explicar la implementación realizada de la aplicación. Se explicarán por separado, los distintos componentes implementados que conforman la aplicación. Se ha decidido, que un tema

exista en la base de datos, siempre y cuando, algún suscriptor esté suscrito a él. En caso de dejar de tener suscriptores, el tema será eliminado.

3.1 Editor

El editor, ha sido implementado en el lenguaje de programación C. Se encontrará en un bucle esperando que se escriba un nuevo mensaje de texto a enviar. Para facilitar la finalización del programa, si el texto escrito para enviar es "exit", se finalizará la ejecución del editor.

En primer lugar, una vez insertado un texto a enviar, se realiza la formación del mensaje a enviar. Este mensaje, se forma concatenando el tema (pasado por argumento), el carácter ':' y el texto escrito. Una vez formado el mensaje, el editor crea un nuevo descriptor de socket. Una vez obtenido el descriptor, se procede a configurar las estructuras que conforman el *socket*, asignándole como puerto de servidor y dirección ip las proporcionadas como argumento en el lanzamiento del programa, siendo ésta última en formato decimal-punto. Una vez configurado el *socket*, se procede a realizar la conexión con el servidor. En caso de realizarse correctamente la conexión, se envía por el socket el mensaje al intermediario. Una vez enviado, se cierra el descriptor del *socket*, y se espera a la inserción de un nuevo texto.

3.2 Intermediario

El intermediario, ha sido implementado en el lenguaje de programación C. Al comienzo de su ejecución, se encarga de comprobar que los puertos pasados como parámetros, para atender las peticiones, de los editores y los suscriptores, se encuentra en el rango permitido (1024<= puerto <= 65535). En caso de no encontrarse en este rango, se notificará el error, finalizando la ejecución. En caso de ser correctos, se procede a la creación del *thread* que realizará la tarea de servidor para los suscriptores.

De esta forma, el hilo principal del programa, se utilizará para atender las peticiones de los editores, mientras que el *thread* creado, se utilizará para atender las peticiones de los suscriptores.

3.2.1 Servidor Editores

En este hilo del programa, en primer lugar, se obtiene el descriptor del socket que se utilizará para atender a las peticiones de los editores. Una vez obtenido el descriptor, se realiza la configuración de los atributos que conforman el socket, asignándole como dirección ip la de la máquina en la que se encuentra, y como puerto el pasado por parámetro. Una vez configurado, se asigna la dirección al socket, y se comienzan a escuchar las peticiones.

El proceso servidor, se mantendrá en un bucle infinito, esperando que se realice una conexión a él. Por cada conexión recibida, se crea un *thread*, que se encarga de obtener el mensaje enviado por el editor. Una vez leído el mensaje, finaliza la conexión con el editor, y consulta en busca en la base de datos, si el tema del mensaje, existe.

Si el tema del mensaje enviado, existe, se procede a difundir el mensaje a todos los suscriptores suscritos al tema. En cada envío, se realizará creará un socket, asignándole la ip y puerto del servidor que espera los mensajes. Al conectar, se procede a enviar el mensaje, y tras el envío, se cierra la conexión. Por último, se destruye el *thread*.

3.2.2 Servidor Suscriptores

En este hilo del programa, en primer lugar, se obtiene el descriptor del socket que se utilizará para atender a las peticiones de los suscriptores. Una vez obtenido el descriptor, se realiza la configuración de los atributos que conforman el socket, asignándole como dirección ip la de la máquina en la que se encuentra, y como puerto el pasado por parámetro. Una vez configurado, se asigna la dirección al socket, y se comienzan a escuchar las peticiones.

El proceso servidor, se mantendrá en un bucle infinito, esperando que se realice una conexión a él. Por cada conexión recibida, se crea un *thread*, que se encarga de atender la petición recibida del suscriptor. En primer lugar, se recibe una cadena de texto, que como máximo, tendrá el tamaño de la operación más larga. Esta cadena, determina la operación a realizar.

Si la operación recibida, es *"SUBSCRIBE"*, tras leer la operación, se procede a leer el tema al que se quiere suscribir el suscriptor. Posteriormente, se procede a realizar la lectura de 2 bytes, que determinan el puerto en el que el suscriptor esperará la recepción de mensajes. Además, se obtiene de la estructura del socket, la dirección ip del suscriptor, que será utilizada en el registro en la base de datos. Una vez obtenidos los datos, se procede a realizar la suscripción del suscriptor en el tema deseado. Para ello, se buscará el tema. En caso de no existir, se creará. Encontrado el tema, se realizará la suscripción, registrando la dupla <ip,puerto> del suscriptor. Por último, se enviará un byte al suscriptor, indicando si la operación se ha realizado de forma satisfactoria, o si por el contrario, ha ocurrido algún error.

Si la operación recibida, es *"UNSUBSCRIBE"*, tras leer la operación, se procede a leer el tema del cual se quiere dar de baja el suscriptor. Posteriormente, se proceder a realizar la lectura de 2 bytes, que determinan el puerto en el que el suscriptor espera la recepción de mensajes. Además, se obtiene de la estructura del socket, la dirección ip del suscriptor, que será utilizada para dar de baja al suscriptor. Estos dos atributos, son

necesarios, puesto que cada suscriptor de identifica de forma unívoca con la dupla <ip,puerto>. Por último, se envía un byte al suscriptor, indicando el resultado de la operación.

Una vez realizada la operación, se cierra el descriptor del socket, y se finaliza el *thread*.

3.3 Suscriptor

El suscriptor, ha sido implementado en el lenguaje de programación JAVA. Al comienzo de su ejecución, lanza una *Shell* para la realización de las operaciones de suscripción con el intermediario. En primer lugar, comprueba que los argumentos pasados al ejecutar el programa, son correctos, verificando que el puerto del intermediario dado, se encuentra en el rango permitido (1024<= puerto <= 65535). Una vez comprobado, la *Shell* espera a recibir operaciones. Estas operaciones, serán 3: *SUBSCRIBE*, UN*SUBSCRIBE* y *QUIT*.

La operación *SUBSCRIBE*, se encarga de suscribir al suscriptor a un tema. En caso de ser el primer al que se suscribe, en primer lugar, se creará un *thread*, como servidor, que se encargará de atender los mensajes de los temas suscritos. Este *thread*, constará de un socket para la escucha. Este socket se asignará, de forma que se busca el primero libre en el rango permitido. Posteriormente, se comprueba que la dirección ip del intermediario, pasada por parámetro (en formato decimal-punto o nombre), es correcta. En caso de ser correcta, se procede a realizar la conexión con el suscriptor para suscribirse al tema. Una vez establecida la conexión, se envía en primer lugar, la cadena de texto que describe la operación a realizar. Posteriormente, se envía el tema, y por último, el puerto en el que se escucharán los mensajes. El suscriptor, una vez enviada toda esta información, espera a recibir un byte, que identifica el resultado de la operación en el intermediario. En caso de no haberse realizado correctamente, se procede a eliminar el hilo escuchador, de forma que mediante una función *kill()* implementada, se envía un mensaje al socket, con el mensaje "kill", que hace que finalice su ejecución.

La operación UN*SUBSCRIBE*, se encarga de dar de baja al suscriptor de un tema. En primer lugar, se comprueba que la dirección ip del intermediario, pasada por parámetro (en formato decimal-punto o nombre), es correcta. En caso de ser correcta, se procede a realizar la conexión con el suscriptor para realizar la operación. Una vez establecida la conexión, se envía en primer lugar, la cadena de texto que describe la operación a realizar. Posteriormente, se envía el tema, y por último, el puerto en el que se escuchan los mensajes. El suscriptor, una vez enviada toda esta información, espera a recibir un byte, que identifica el resultado de la operación en el intermediario. En caso de haberse realizado correctamente, se procede a comprobar si el número de suscripciones, es igual a 0. En este

caso, se elimina el *thread* escuchador mediante la función *kill()* mencionada anteriormente.

La operación *QUIT*, se encarga de dar de baja al suscriptor de todos los temas suscritos. Para ello, únicamente, realizará tantas operaciones *UNSUBSCRIBE* como temas esté suscrito.

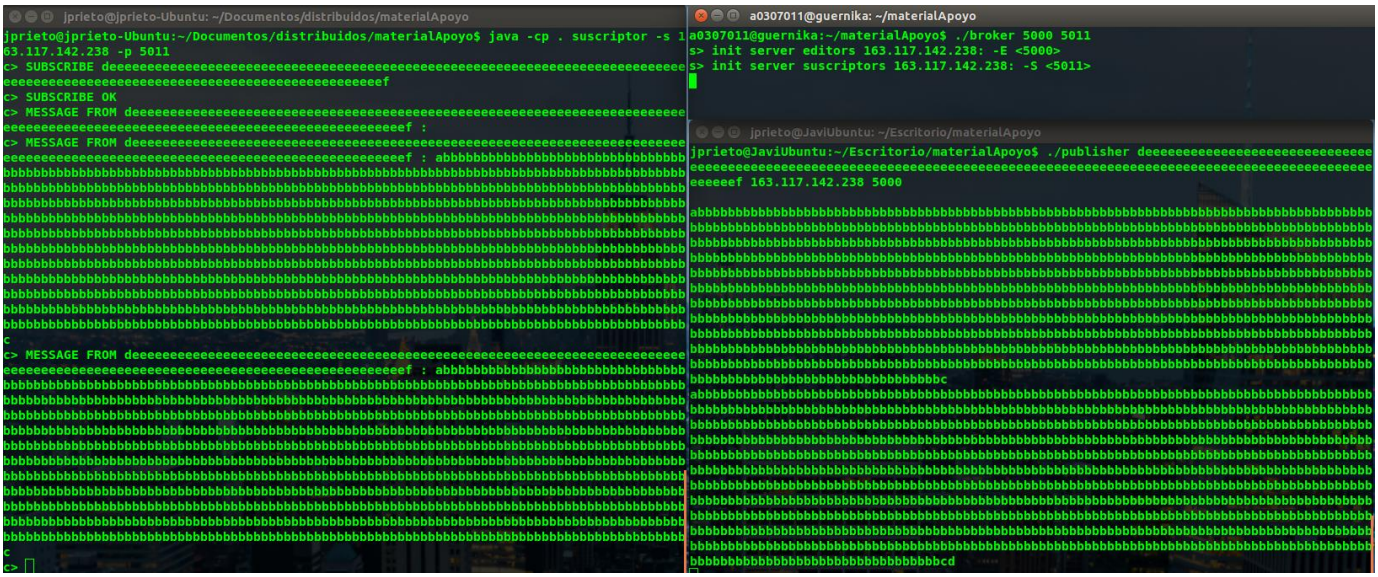
4 Batería de pruebas

En este apartado, se procede a mostrar la batería de pruebas que verifica el correcto funcionamiento de la implementación. Para ello, la ejecución de cada componente se ha realizado en máquinas distintas, demostrando el correcto funcionamiento de la aplicación distribuida. Además, se puede observar que los suscriptores, aceptan la dirección del intermediario tanto en formato decimal-punto como en nombre.

4.1 Prueba 1: Longitudes de cadenas

Con esta prueba, se pretende verificar el correcto funcionamiento del protocolo de comunicación, en su apartado de longitudes de cadena. Para ello, se ha probado a utilizar el tamaño máximo de tema, y de texto.

Resultado esperado: En primer lugar, se realiza la suscripción a un tema de longitud 128 bytes. En editor, genera mensajes de éste tema. Posteriormente, el editor, enviará dos mensajes. El primer mensaje, con una longitud de texto de 1024 bytes, y el segundo, con una longitud de 1025 bytes.



Resultado obtenido: Como se puede observar, el primer mensaje, se recibe correctamente, puesto que se encuentra dentro del límite de longitud establecido por el protocolo. El segundo mensaje, no recibe el último byte,

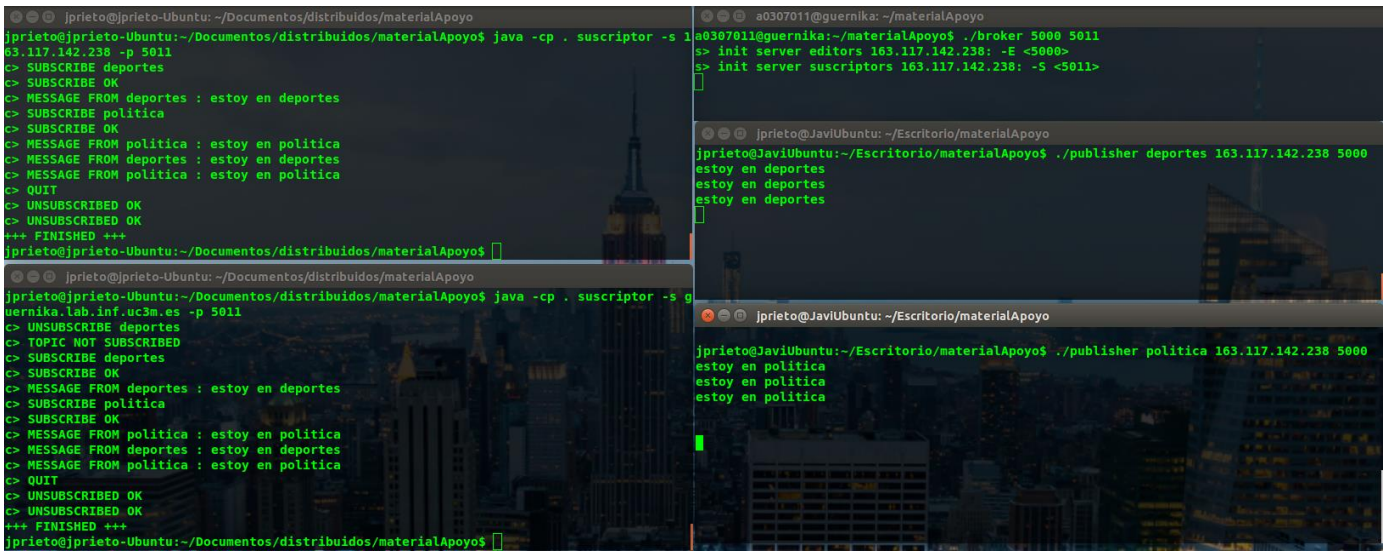
puesto que excede el tamaño en 1 byte. Por tanto, podemos concluir, que se realiza de forma satisfactoria.

En este apartado, se procede a mostrar la batería de pruebas que verifica el correcto funcionamiento de la implementación. Para ello, la ejecución de cada componente se ha realizado en máquinas distintas, demostrando el correcto funcionamiento de la aplicación distribuida.

4.2 Prueba 2: Gestión de suscripciones

Con esta prueba, se pretende verificar el correcto funcionamiento en la gestión de suscripciones, tanto al dar de alta, como al dar de baja. Para ello, contamos con 2 suscriptores, y 2 editores.

Resultado esperado: En primer lugar, el primer suscriptor, se suscribe a deportes. Posteriormente, el segundo suscriptor, pretende darse de baja en un tema no suscrito. Tras ello, el editor de deportes, envía un mensaje, que únicamente debería llegar al sucriptor1. Posteriormente, el suscriptor2, se suscribe a deportes. Después, el editor de deportes, vuelve a generar un mensaje, que debería llegar a los dos suscriptores. Después, ambos se suscriben al tema política. Cuando el editor de política manda un mensaje, debería llegar a ambos. Por último, ambos se dan de baja en todos los temas, mediante la operación *QUIT*, y salen. Cuando los dos editores envían nuevos mensajes, no debería haber problema al no existir los suscriptores.

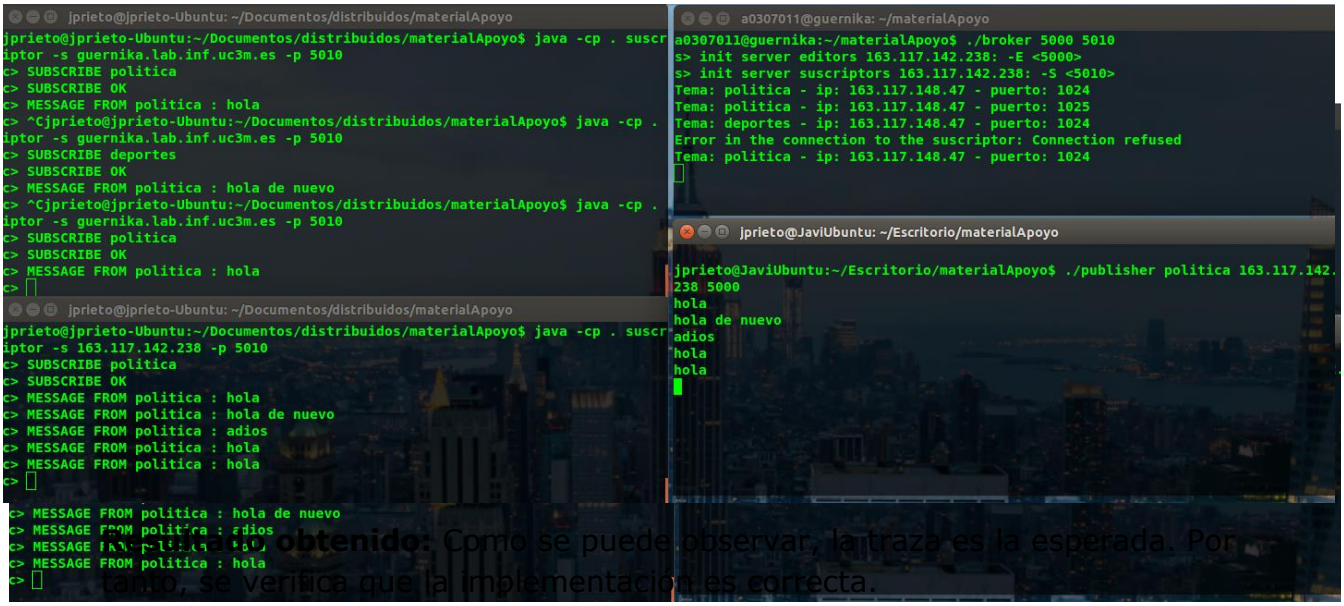


Resultado obtenido: Como se puede observar, la traza es la esperada. Por tanto, se verifica que la implementación es correcta.

4.3 Prueba 3: Suscriptores no conectados

Con esta prueba, se pretende verificar el correcto funcionamiento cuando un suscriptor suscrito a un tema, no se encuentra conectado.

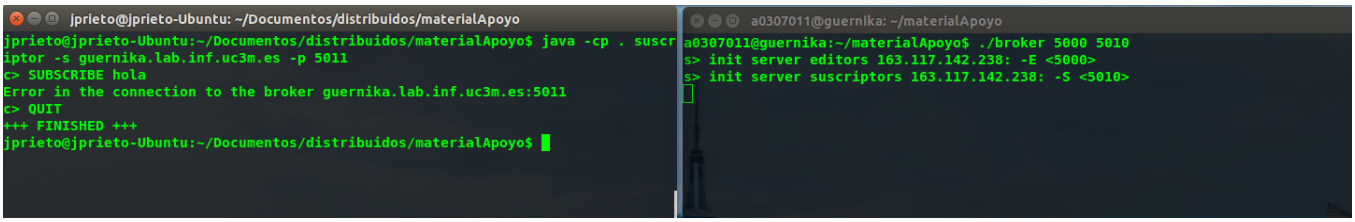
Resultado esperado: En primer lugar, el suscriptor1 (con puerto 1024), se suscribe a política. Posteriormente, el suscriptor2 (con puerto 1025, se suscribe también a política. Después, el editor, envía el mensaje "hola", que debería llegar a ambos. Tras ello, el suscriptor1, se desconecta inesperadamente, volviéndose a conectar antes de que suceda nada, y tomando el primer puerto libre, el 1024 que dejó libre al cerrarse. Cuando el editor manda el nuevo mensaje, pese a no suscribirse, debería llegarle, puesto que en la base de datos, no se ha recibido notificación de su desconexión. Posteriormente, vuelve a cerrarse el suscriptor1 inesperadamente. Pero en este caso, el editor, manda un mensaje, "adiós". En el envío, debería comprobar que no está el suscriptor1 conectado, dándole de baja. Cuando vuelve a tomar ejecutar, toma el puerto 1024. El editor envía un mensaje, pero no debería de llegarle.



4.4 Prueba 3: Conexión a dirección ip:puerto errónea

Con esta prueba, se pretende verificar el correcto funcionamiento en la conexión a una dirección ip:puerto erróneo.

Resultado esperado: El suscriptor, pretende conectarse al servidor, con la ip correcta, pero con un puerto distinto al que debe, por lo que no debería de conectarse.



Resultado obtenido: Como se puede observar, la traza es la esperada, puesto que la conexión, no se puede realizar. Por tanto, se verifica que la implementación es correcta.

5 Compilación

Para la compilación de la práctica, hemos generado un MAKEFILE, que se encarga de la generación de todos los ejecutables. Para ello, únicamente es necesaria la utilización de dos comandos.

El comando ***make clean***, se encarga de eliminar los ejecutables y ".o" generados. El comando ***make***, se encarga de generar los .o y los ejecutables necesarios para la ejecución de la práctica.

```
gcc -Wall  -g -Wall  -g -c broker.c
gcc -Wall  -g -Wall  -g -c read_line.c
gcc -Wall  -g -Wall  -g -c database.c
gcc -L/lib/ broker.o read_line.o database.o -lpthread -o broker
gcc -Wall  -g -Wall  -g -c publisher.c
gcc -L/lib/ publisher.o read_line.o -lpthread -o publisher
javac -g suscriptor.java
```

6 Conclusiones

La implementación de un sistema editores/escritores, resulta de gran ayuda para la comprensión de los sockets TCP, y de un sistema cliente/servidor.

Hemos tenido ciertos problemas a la hora de poder destruir un hilo de java, pero finalmente, se ha resuelto. Además, hemos encontrado el problema de gestionar la caída de un suscriptor, de forma que el intermediario lo desconozca (prueba 3). Habría sido interesante resolver estos problemas de disponibilidad, pero no entraban en el ámbito de la práctica.

Habría sido interesante, realizar pruebas de rendimiento en cuanto al número de peticiones simultáneas que el intermediario es capaz de gestionar.

Por último, también destacar, la importancia que tiene modularizar el código, puesto que cuando toma una extensión considerable, es difícil de depurar.