Informe implementacion de sistema de chat *schat* y *cchat*

Melecio Ponte - 08-10893
Gabriel Formica - 10-11036

Diciembre, 2013

1. Introduccion

El proyecto tiene como propósito la implementación de un programa de chat con un servidor y múltiples clientes. Para lograr esto se hace uso principalmente del lenguaje de programación *C* y las librerías de *sockets* e hilos de *GNU*.

2. Implementación

Para la implementación de sistema de chat, se definen tipos y estrucuturas de datos.

2.1. Tipos de datos

Room

Consiste en un registro del tipo *struct* usado para contener la información perteneciente a cada sala de chat.

name del tipo *char**, contiene el nombre de la sala.

users del tipo *userslist*, contiene a los usuarios registrados a una sala.

User_data

Consiste en un registro del tipo *struct* usado para contener la información perteneciente a cada usuario conectado al servidor.

name del tipo *char**, contiene el nombre del usuario.

users del tipo *int*, corresponde al *socket* correspondiente a la conexión hecha por el hilo del usuario.

subscribed_rooms del tipo *list*, es una lista enlazada de apuntadores a las salas a las cuales está suscrito el usuario.

Userslist

Consiste en un alias al tipo *list*. Se crea para mejorar la legibilidad y aumentar el nivel de semántica de la representación en el código.

2.2. Estructuras de datos

list

Una lista enlazada genérica que es usada durante para la estructuración de los tipos de datos y en general para el flujo del programa.

3. Funciones del chat

La implementación del programa de chat cuenta con las siguientes funciones, las cuales pueden ser introducidas por el cliente. A continuación, una descripción del funcionamiento de las mismas:

sal Desde el cliente se envía el comando al servidor, el cual responde con el siguiente formato:

```
---These are all the rooms---

* actual

* sala1

* sala2

-----------------------------
```

usu Desde el cliente se envía el comando al servidor, el cual responde con el siguiente formato:

```
---These are all the users connected---

* usuario1

* usuario2

* usuario3

-----------------------------
```

men Desde el cliente se envía el comando al servidor, en conjunto con un mensaje, el servidor recibe dicho mensaje y lo envía a todos los usuarios conectados a las salas a las cuales está suscrito el emisor:

```
usuario1 (from room 'actual') says: este es un mensaje
```

sus Desde el cliente se envía el comando al servidor, en conjunto con el nombre de una sala, el servidor procede a agregar al usuario la sala enviada como argumento. Si la suscripción es exitosa, el usuario recibe el siguiente mensaje del servidor:

```
You are now subscribed to the room 'room1'!
```

En caso de que la suscripción sea fallida porque la sala no existe, el usuario recibe el siguiente mensaje del servidor:

```
Sorry, this room does not exist.
```

des Desde el cliente se envía el comando al servidor, el servidor procede a desuscribir al usuario de todas las salas a las cuales está suscrito. El servidor entonces envía el siguiente mensaje al usuario:

```
You have been unsusbcribed of all of the rooms.
```

cre Desde el cliente se envía el comando al servidor junto con el nombre de una sala. El servidor procede a crear dicha sala. Si la operación es exitosa, no se retorna nada. Si la sala ya existe, el servidor devuelve al usuario el siguiente mensaje:

```
Room already exists.
```

eli Desde el cliente se envía el comando al servidor junto con el nombre de una sala. El servidor procede a eliminar dicha sala. Si la operación es exitosa, no se retorna nada. Si la sala no existe, el servidor devuelve al usuario el siguiente mensaje:

```
You can't delete a room that doesn't exist.
```

Si el usuario intenta eliminar la sala por defecto creada al iniciar el servidor, la respuesta del mismo es:

```
You just can't delete the default room.
```

fue Desde el cliente se envía el comando al servidor el cual cierra la conexión para el usuario y devuelve al usuario el siguiente mensaje:

```
See you later!
```

4. Otros detalles de implementación

Debido a los inconvenientes conocidos por el algoritmo de Nagle, se configuró los sockets para desactivar dicho algoritmo, lo cual tiene como consecuencia el envío de paquetes tan pronto se termine de escribir al socket, en lugar de acumularlos hasta conseguir el máximo total enviable por el sistema (cantidad que, según las pruebas realizadas mediante *tcpdump* y en el contexto local es de 1448 bytes).

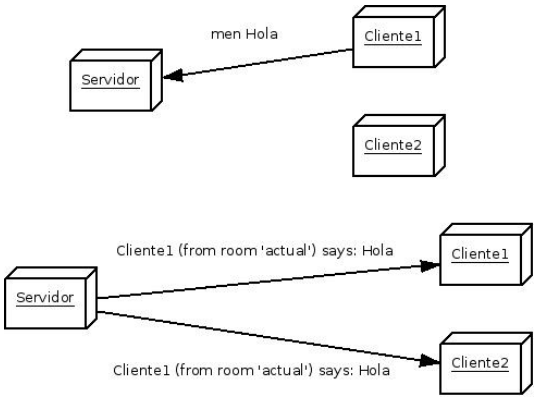
Para evitar problemas con la escritura y lectura servidor/cliente, se establece que el tamaño por defecto de los paquetes envíados es de 256 bytes. Se entiende que esto ocasiona un *overhead* pero a su vez, debido a la naturaleza e intención del programa, se determina que esto no representa un riesgo para el desempeño del mismo.

5. Diagrama de intercambio de mensajes

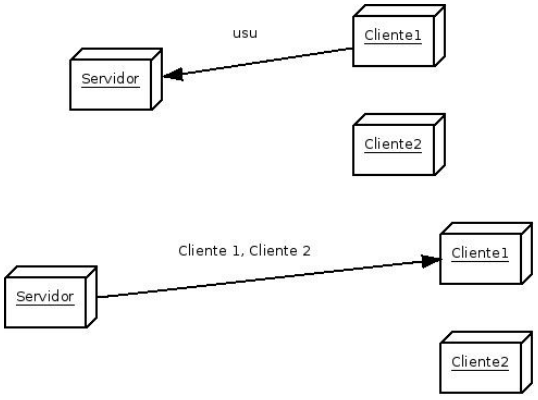
A continuación, se mostrará la secuencia de intercambio de mensajes del programa en el caso en el cual se haga la siguiente secuencia de comandos:

```
men Hola
```

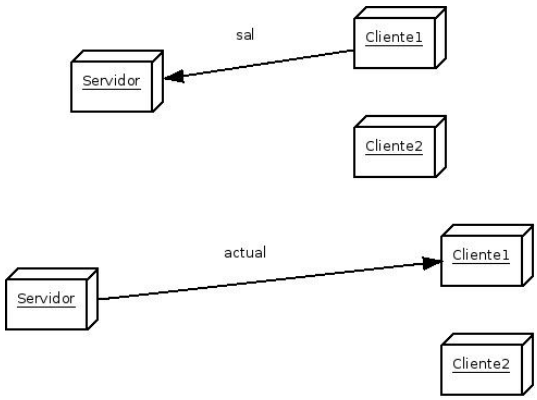
El primer cliente (Cliente 1) envía el comando «men hola» al servidor.



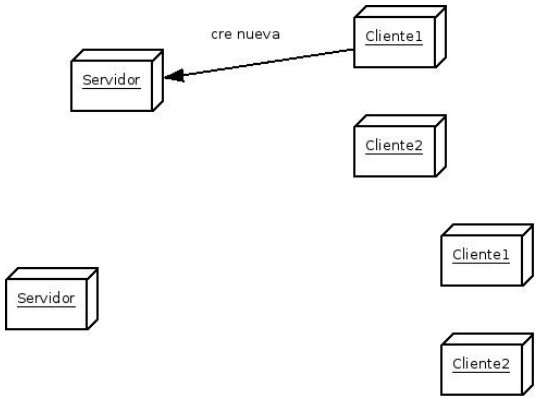
El primer cliente (Cliente 1) envía el comando «usu» al servidor. Por razones de espacio, se omite el formato en el cual el Servidor envía a Cliente1 el mensaje.



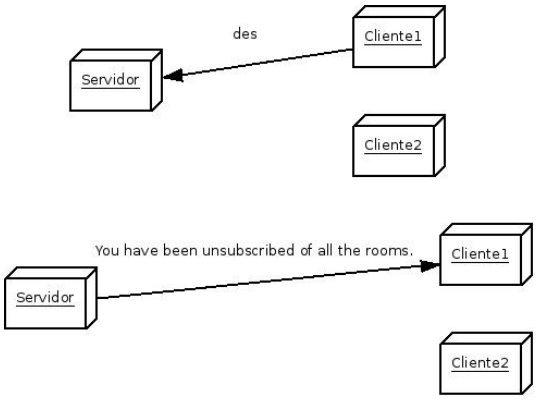
El primer cliente (Cliente 1) envía el comando «sal» al servidor. Por razones de espacio, se omite el formato en el cual el Servidor envía a Cliente1 el mensaje.



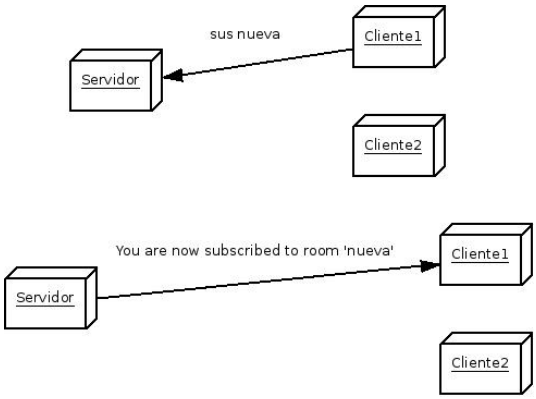
El primer cliente (Cliente 1) envía el comando «cre nueva» al servidor. En este caso, Servidor no envía mensaje a ninguno de los clientes.



El primer cliente (Cliente 1) envía el comando «des» al servidor.



El primer cliente (Cliente 1) envía el comando «sus nueva» al servidor.



El primer cliente (Cliente 1) envía el comando «men chao» al servidor. Esta vez, sólo Cliente1 recibe el mensaje, puesto que es el único suscrito a la sala ñueva'.