Práctica 7 - PC

Carlos Giraldo y Ricardo Pragnell

Ejercicio 1

Nuestra solución comienza con el módulo **messenger** obtenido del tutorial de **Erlang** en su versión robusta. Lo primero para adaptarlo a la estructura del chat fue separar la funcionalidad del **servidor** de la del **cliente** en módulos separados.

Presentamos dos módulos: **servidorchat** y **clientechat**.

- **servidorchat.erl**

  El código del servidor permanece en su mayoría inalterado de la versión anterior salvo por la adición de la variable **Message_Couter**. Esta variable se inicializa a 0 en el momento de crear el proceso **servChat**. **Message_Counter** es una variable que se transmite en la llamada recursiva del proceso **server** junto con la lista de usuarios conectados, modificamos su valor cada vez que llega una petición de difundir un mensaje por parte de un cliente. En el momento de difundir el mensaje se crea un nuevo proceso que se encarga de realizar la función **entregarMensaje**. Esto se realiza mediante el uso de la instrucción **spawn**.

```
server(User_List,Message_Counter) ->
receive
...
{From, message_all, Message} ->
...
spawn(servidorchat,entregarMensaje,[User_List,From,Message,New_Message_Counter])
…
end.
```

- **clientechat.erl**

  En el cliente llevamos a cabo varias modificaciones. La más llamativa es la creación y registro de un proceso **mess_publisher** junto con la creación del proceso principal del cliente. Este proceso se encarga de controlar la salida por pantalla de los mensajes que se reciben del servidor. **Mess_publisher** se comporta como el **publisher** de la práctica de **RMI** en cuanto a que recibe los mensajes del servidor (que pueden llegar de forma desordenada) y los muestra en orden por la consola. El proceso **mess_publisher** se encuentra en un bucle, y a cada vuelta, si recibe un mensaje lo guarda en una lista, y en caso de no recibir ninguna llamada se dedica a escribir el mensaje cuyo

Message_Counter sea el más pequeño de los que tenga. Para ello usamos las funciones **minCounter** y **del_MinCounter** que se encargan de devolver el mensaje cuyo contador sea más pequeño y la lista de mensajes sin dicho mensaje.

```
publisher(Message_List) ->
receive
...
{publish_m,FromName,Message,Message_Counter} ->
New_Message_List=lists:append([{FromName,Message,Message_Counter}],Message_List)
        publisher(New_Message_List)
after 100 ->
  case Message_List of
  [] -> publisher(Message_List);
  _ -> {FromName,Message,Message_Counter} = minCounter(Message_List),
        New_Message_List = del_minCounter(Message_List,Message_Counter),
        io:format("Message from ~p(~p): ~p~n", [FromName, Message_Counter, Message]),
        publisher(New_Message_List)
   end
end.
```

Por otro lado destaca el añadido de poder hacer **broadcast** por parte del cliente. Al hacerlo se pasa un mensaje al proceso del cliente y éste se encarga de comunicarse con el servidor.

```
broadcast(Message) ->
   case whereis(mess_client) of % Test if the client is running
      undefined ->
         not_logged_on;
      _ -> mess_client ! {message_all, Message},
end.

client(Server_Node) ->
receive
   …
   {message_all, Message} ->{servChat, Server_Node} ! {self(), message_all, Message},
       await_result();
   {message_from, FromName, Message,Message_Counter} ->
       mess_publisher ! {publish_m,FromName,Message,Message_Counter}
end,
client(Server_Node).
```