Programación de Sistemas de Telecomunicación / Informática II
Práctica 2:
Mini-Chat en modo cliente/servidor

Departamento de Sistemas Telemáticos y Computación
(GSyC)

Octubre de 2013

1. Introducción

En esta práctica debes realizar dos programas en Ada siguiendo el modelo cliente/servidor que permitan implementar un sencillo sistema de chat entre usuarios.

El programa cliente podrá comportarse de dos maneras: en modo escritor, o en modo lector. Si un cliente es escritor podrá enviar mensajes al servidor del chat. Si un cliente es lector podrá recibir los mensajes que envíe el servidor.

El servidor se encargará de recibir los mensajes procedentes de los clientes escritores, y reenviárselos a los clientes lectores.

En una sesión de chat participará un programa servidor y varios programas clientes (lectores y escritores).

2. Descripción del programa cliente `chat_client.adb`

2.1. Interfaz de usuario

El programa cliente se lanzará pasándole 3 argumentos en la línea de comandos:

- Nombre de la máquina en la que está el servidor
- Número del puerto en el que escucha el servidor
- *Nickname* (apodo) del cliente del chat. Si el *nick* es `lector`, el cliente funcionará en modo lector, con cualquier otro *nick* el cliente funcionará en modo escritor.

Una vez lanzado:

- Si el cliente se lanza en modo **escritor**, pedirá al usuario cadenas de caracteres, y se las irá enviando al servidor. El programa terminará cuando el usuario introduzca la cadena `.salir`
- Si el cliente se lanza en modo **lector**, esperará a recibir mensajes del servidor (conteniendo las cadenas enviadas por los clientes escritores del chat), y los mostrará en pantalla. En este modo el cliente nunca terminará su ejecución.

Ejemplos de ejecución:

```
$ ./chat_client zeta12 9001 carlos
Mensaje: Hola
Mensaje: quién está ahí?
Mensaje: ana dime algo
Mensaje: adios
Mensaje: .salir
$

$ ./chat_client zeta12 9001 ana
Mensaje: entro
```

```
Mensaje: estoy yo, soy ana
Mensaje: hola carlos
Mensaje: hasta luego
Mensaje: .salir
$

$ ./chat_client zeta12 9001 lector
ana: entro
carlos: Hola
carlos: quién está ahí?
ana: estoy yo, soy ana
carlos: ana dime algo
ana: hola carlos
carlos: adios
ana: hasta luego
```

2.2. Implementación

El programa cliente tendrá dos comportamientos diferentes según sea lanzado como lector (con el *nick* `lector`) o escritor (con cualquier otro *nick*).

Cuando es lanzado como escritor, enviará un **mensaje inicial** al servidor para indicarle su nick. Luego entrará en un bucle que pida al usuario cadenas de texto, que le irá enviando al servidor mediante **mensajes de escritor**.

Cuando es lanzado como lector, enviará un **mensaje inicial** al servidor para indicarle su nick (que siempre será `lector`) y luego entrará en un bucle infinito esperando a recibir **mensajes de servidor**, cuyo contenido mostrará en pantalla.

En el apartado 4 se explica el formato de los mensajes.

3. Descripción del programa servidor `chat_server.adb`

3.1. Interfaz de usuario

El programa servidor se lanzará pasándole 1 argumento en la línea de comandos:

- Número del puerto en el que escucha el servidor

Una vez lanzado, el servidor recibirá mensajes procedentes de clientes:

- Si recibe un **mensaje inicial**, añadirá el cliente a su lista de los clientes que conoce
- Si recibe un **mensaje de escritor**, reenviará dicho mensaje a los clientes lectores que conoce

El servidor nunca terminará su ejecución.
El servidor irá mostrando en pantalla los mensajes que vaya recibiendo para permitir comprobar su funcionamiento.
Ejemplos de ejecución:

```
$ ./chat_server 9001
recibido mensaje inicial de ana
recibido mensaje inicial de carlos
recibido mensaje de ana: entro
recibido mensaje de carlos: Hola
recibido mensaje de carlos: quién está ahí?
recibido mensaje de ana: estoy yo, soy ana
recibido mensaje de carlos: ana dime algo
recibido mensaje de ana: hola carlos
recibido mensaje de carlos: adios
recibido mensaje de ana: hasta luego
```

3.2. Implementación

El servidor debe atarse en un `End_Point` formado con la dirección IP de la máquina en la que se ejecuta, y el puerto que le pasan como argumento.

Una vez atado, entrará en un bucle infinito recibiendo mensajes de clientes. El servidor deberá tener una lista de clientes conocidos. **Para implementar dicha lista deberá utilizarse un array**, no pudiéndose utilizar una lista dinámica. De cada cliente el servidor deberá almacenar su `End_Point` y su *nick*.

Cuando el servidor reciba un **mensaje inicial** de un cliente, le añadirá a su lista de clientes conocidos.

Cuando el servidor reciba un **mensaje de escritor** de un cliente, mostrará un mensaje en la pantalla indicando que se ha recibido un mensaje (ver ejemplo de ejecución más arriba) y enviará un **mensaje de servidor** a todos los clientes conocidos que se registraran con *nick* `lector`.

En el apartado 4 se explica el formato de los mensajes.

NOTA: Para localizar el *nick* del cliente que envía un mensaje, el servidor deberá buscar en la lista de clientes conocidos. Para poder comparar dos `End_Point` es necesario poner la siguiente cláusula en la zona de declaraciones del procedimiento:

```
use type LLU.End_Point_Type;
```

4. Formato de los mensajes

Los tres tipos de mensajes que se necesitan para esta práctica se distinguen por el primer campo, que podrá adoptar los valores del siguiente tipo enumerado:

```
type Message_Type is (Init, Writer, Server);
```

Dicho tipo deberá estar declarado en el fichero `chat_messages.ads` y desde ahí ser usado tanto por el cliente como por el servidor. Así el código de cliente o del servidor tendrá el siguiente aspecto:

```
with Chat_Messages;
...
procedure ... is
   package CM renames Chat_Messages;
   use type CM.Message_Type;
   ...

   Mess: CM.Message_Type;

begin
   ...
   Mess := CM.Init;
   ...
   if Mess = CM.Server then
      ...

end ...;
```

Si este paquete `Chat_Messages` no contiene ningún procedimiento no es necesario que tenga cuerpo (fichero `.adb`), sino que sólo tendrá especificación (fichero `.ads`).

Mensaje inicial

Es el que envía un cliente al servidor al arrancar. Formato:

Init	Client_EP	Nick

en donde:

- `Init`: `Message_Type` que identifica el tipo de mensaje.

- `Client_EP`: `End_Point` del cliente que envía el mensaje.
- `Nick`: `Unbounded_String` con el *nick* del cliente. Si el *nick* es `lector`, el cliente estará en modo lector, en caso contrario estará en modo escritor.

Mensaje de escritor

Es el que envía un cliente escritor al servidor con una cadena de caracteres introducida por el usuario. Formato:

`Writer`	`Client_EP`	`Comentario`

en donde:

- `Writer`: `Message_Type` que identifica el tipo de mensaje.
- `Client_EP`: `End_Point` del cliente que envía el mensaje.
- `Comentario`: `Unbounded_String` con la cadena de caracteres introducida por el usuario

Nótese que el *nick* no viaja en estos mensajes, el cliente queda identificado por su `End_Point`, y el *nick* deberá ser recuperado por el servidor de la lista de clientes conocidos para poder componer el mensaje de servidor que reenviará a los clientes en modo lector.

Mensaje de servidor

Es el que envía un servidor a un cliente lector con el comentario que le llegó en un mensaje de escritor. Formato:

`Server`	`Nick`	`Comentario`

en donde:

- `Server`: `Message_Type` que identifica el tipo de mensaje.
- `Nick`: `Unbounded_String` con el *nick* del cliente que envía el mensaje.
- `Comentario`: `Unbounded_String` con la cadena de caracteres introducida por el usuario

5. Condiciones de Funcionamiento

1. El chat debe funcionar con cualquier número de clientes menor o igual a 50. **No se podrá utilizar en esta práctica una lista dinámica** para almacenar los datos de dichos clientes.
2. Se supone que nunca se pierden los mensajes enviados por el servidor ni por los clientes.
3. Se permite que haya diferentes clientes en modo escritor con el mismo *nick*.
4. El cliente programado por un alumno deberá poder funcionar con el servidor programado por cualquier otro alumno, y viceversa. Por ello es muy importante respetar el protocolo de comunicación entre cliente y servidor y, especialmente, el formato de los mensajes.
5. Cualquier cuestión no especificada en este enunciado puede resolverse e implementarse como se desee.

6. Normas de entrega

6.1. Alumnos de Programación de Sistemas de Telecomunicación

Debes dejar el código fuente de tu programa dentro de tu cuenta en los laboratorios de prácticas. Directamente en el directorio de tu cuenta debes crear una carpeta de nombre `PST.2013`, y dentro de ella una carpeta de nombre `P2`. En ella deben estar todos los ficheros correspondientes a los dos programas (`chat_client.adb`, `chat_server.adb` y los ficheros de los paquetes auxiliares si los tuvieras).
Importante: Pon un comentario con tu nombre completo en la primera línea de cada fichero fuente.

6.2. Alumnos de Informática II

Debes dejar el código fuente de tu programa dentro de tu cuenta en los laboratorios de prácticas. Directamente en el directorio de tu cuenta debes crear una carpeta de nombre `I2.2013`, y dentro de ella una carpeta de nombre `P2`. En ella deben estar todos los ficheros correspondientes a los dos programas (`chat_client.adb`, `chat_server.adb` y los ficheros de los paquetes auxiliares si los tuvieras).
Importante: Pon un comentario con tu nombre completo en la primera línea de cada fichero fuente.

7. Plazo de entrega

Los ficheros deben estar en tu cuenta, en la carpeta especificada más arriba, antes de las 23:59h del Miércoles 23 de octubre de 2013.