

Práctica 4

ChatsApp

Fecha de entrega: 18 de mayo de 2014

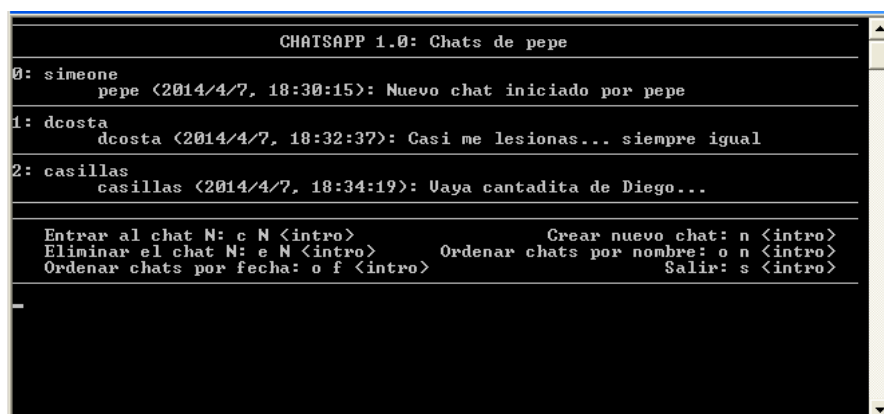
Una conocida empresa de software quiere desarrollar *ChatsApp*, una aplicación de envío de mensajes desde dispositivos móviles y no móviles que haga la competencia al conocido *WhatsApp*. Para ello deciden organizar un concurso para estudiantes de Ingeniería Informática cuyo ganador recibirá una prestigiosa beca de estudios. Se trata de desarrollar un prototipo con interfaz de consola con el que poder simular el uso de la aplicación *ChatsApp* de manera offline.

1. Descripción de la práctica

1.1. La simulación

El programa comenzará solicitando el nombre o identificador de un usuario registrado en el sistema. En caso de introducir uno válido, se accederá al menú principal de usuario del ChatsApp, desde el cual se podrá: consultar y escribir mensajes en chats existentes, crear chats nuevos y eliminarlos (ver apartado 1.2). Supongamos que el usuario *pepe* ha iniciado sesión y que envía un nuevo mensaje al chat que tiene con el usuario *dcosta*. Al salir del menú de usuario, se volverá a la pantalla inicial del programa. Si entonces accede el usuario *dcosta*, en su menú de usuario se mostrarán todos sus chats, y en concreto, al entrar en el chat con *pepe* se mostraría el mensaje que éste acababa de escribir. El programa terminará cuando se introduzca un cero.

1.2. El menú de usuario del ChatsApp



```

CHATSAPP 1.0: Chats de pepe
0: simeone
   pepe <2014/4/7, 18:30:15>: Nuevo chat iniciado por pepe
1: dcosta
   dcosta <2014/4/7, 18:32:37>: Casi me lesionas... siempre igual
2: casillas
   casillas <2014/4/7, 18:34:19>: Uaya cantadita de Diego...

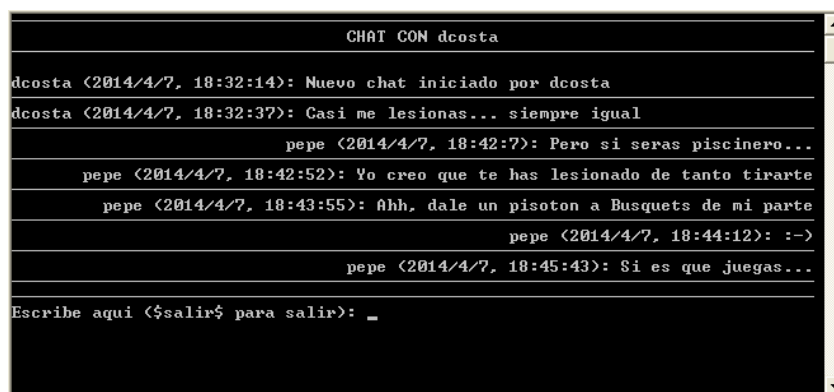
Entrar al chat N: c N <intro>          Crear nuevo chat: n <intro>
Eliminar el chat N: e N <intro>        Ordenar chats por nombre: o n <intro>
Ordenar chats por fecha: o f <intro>    Salir: s <intro>

```

Cuando un usuario inicia sesión accederá al menú de usuario del ChatsApp, en el cual aparecerán todos los chats que tenga abiertos el usuario, ordenados por defecto del menos al más recientemente actualizado (de arriba a abajo). Cada uno estará encabezado por un índice numérico, irá seguido del nombre del chat (nombre del usuario con el que se chatea) y mostrará el último mensaje enviado o recibido en el chat (incluyendo su fecha y hora). Abajo se mostrará información sobre las acciones que se pueden realizar y las teclas con las que realizarlas. En concreto: para entrar al chat de índice N se pulsará “c N <intro>”; para crear un

nuevo chat “n <intro>”, se solicitará entonces el nombre de usuario con el que conectar y se mostrará un mensaje indicando si se ha creado con éxito, si el usuario no existe o si el chat con ese usuario ya existe; para eliminar un chat existente “e N <intro>”; para re-ordenar los chats por nombre “o n <intro>”, para volver al orden por fecha “o f <intro>”; y para salir y volver al menú de la simulación “s <intro>”. Cuando se crea un chat nuevo se le añadirá un mensaje inicial con el texto “Nuevo chat iniciado por <NombreUsuario>”.

Al entrar en un chat se mostrarán todos los mensajes del chat, ordenados del menos al más reciente (de arriba a abajo), y el programa quedará a la espera de que el usuario escriba un nuevo mensaje, en cuyo caso se volverán a mostrar todos los mensajes del chat, incluyendo el nuevo mensaje. Para salir del chat y volver al menú de usuario se escribirá el mensaje especial \$salir\$.



2. Detalles de implementación

2.1. Inicialización y carga de datos

Al iniciarse la simulación el servidor del ChatsApp cargará del fichero *usuarios.txt* (ver detalles más abajo) en una lista (ver módulo ListaUsuarios abajo) los nombres de todos los usuarios registrados.

Cuando un usuario inicia sesión en el ChatsApp, cargará de su fichero *<NombreUsuario>.chats.txt* (ver detalles abajo) su lista de chats (ver módulo ListaChats), cada uno incluyendo todos sus mensajes. El prefijo *<NombreUsuario>* es necesario para poder distinguir los ficheros de chats de distintos usuarios.

2.2. Envío y recepción de mensajes

El servidor del ChatsApp mantiene un buzón de mensajes para cada usuario registrado en el sistema. Cada buzón (tListaMensajes) contiene los mensajes a ser entregados al usuario en cuestión. Cuando un usuario desde uno de sus chats envía un mensaje, invocará a la función *enviarMensaje* del módulo Servidor, la cual almacenará el mensaje en el buzón asociado al usuario receptor. De esta manera, cuando un usuario entra en el sistema, tras cargar sus chats, incluyendo los mensajes de cada uno de ellos (ver fichero *<NombreUsuario>.chats.txt* abajo), obtendrá los mensajes de su buzón y los insertará en los chats correspondientes. Tras esta operación su buzón quedará vacío. Para no perder mensajes que hayan

sido enviados pero aún no obtenidos, al salir de la aplicación el servidor guardará, junto a cada usuario registrado en el sistema, su buzón de mensajes pendientes de entrega (ver fichero *usuarios.txt* abajo).

2.3. Módulos y tipos de datos

Se deben definir al menos los siguientes módulos y tipos de datos:

Módulo Cliente

Declara el tipo de datos `tDatosCliente`, el cual incluye el identificador del usuario que está conectado (`string`) y su lista de chats (`tListaChats`). Define al menos funciones para: inicializar sus estructuras, colocar los mensajes del buzón del servidor en los chats correspondientes, mostrar y manejar el menú de usuario, y, crear un nuevo chat. En otras palabras, sería el módulo raíz de la aplicación ChatsApp que los supuestos usuarios o clientes instalarían en sus dispositivos móviles.

Módulo ListaChats

Declara el tipo de datos `tListaChats` que representa listas de chats (`tChat`) mediante un array y un contador, y define funciones al menos para inicializar, cargar y guardar de/en fichero, buscar (búsqueda lineal por nombre del chat), añadir al final, eliminar (por posición), mover un chat al final (por posición), y ordenar, tanto por nombre como por fecha del último mensaje del chat. Cada vez que un chat se actualiza (se le añaden mensajes) éste se moverá a la última posición de la lista. De esta manera la lista se mantiene ordenada por fecha de actualización (del menos al más reciente), excepto cuando se haya reordenado por nombre.

Módulo Chat

Declara el tipo de datos `tChat`, el cual incluye el nombre del chat (identificador del usuario con el que se habla), el identificador del usuario al que pertenece el chat y la lista de los mensajes que integran el chat (`tListaMensajes`). Define al menos funciones para inicializar, cargar/guardar de/en fichero, mostrar cabecera (lo que aparece en el menú de usuario) y mostrar los mensajes del chat (lo que aparece al entrar al chat).

Módulo Servidor

Declara el tipo de datos `tServidor`, el cual incluye al menos la lista de datos de los usuarios registrados en el sistema (`tListaUsuarios`). Define al menos funciones para gestionar un mensaje enviado por un usuario (función `enviarMensaje`) y para obtener el buzón de mensajes de un usuario. También define la función *main* de la aplicación.

Módulo ListaUsuarios

Declara el tipo de datos `tListaUsuarios` que representa listas ordenadas (por nombre) de datos de usuario (`tDatosUsuario`) mediante un array y un contador, y

define funciones al menos para inicializar, cargar y guardar de/en fichero, buscar e insertar.

Módulo DatosUsuario

Declara el tipo de datos tDatosUsuario, el cual incluye el identificador del usuario (string) y el buzón de mensajes (tListaMensajes). Define al menos funciones para inicializar y cargar/guardar de/en fichero.

Módulo ListaMensajes

Declara el tipo de datos tListaMensajes que representa listas de mensajes (tMensaje) mediante un array y un contador. Define al menos funciones para inicializar, añadir un mensaje al final, consultar el último elemento y cargar/guardar de/en fichero. Observa que este módulo se usa tanto para las listas de mensajes de los chats, como para los buzones de correo que mantiene el servidor.

Módulo Mensaje

Declara el tipo de datos tMensaje, el cual incluye el emisor y receptor del mensaje (strings identificadores de usuario), la fecha (tFecha) y su texto (string). Define al menos funciones para crear un mensaje nuevo, mostrar y cargar/guardar de/en fichero. La fecha se representará internamente (tanto en memoria como en los ficheros) en el formato UNIX, es decir, un entero con el número de segundos transcurridos desde el 1 de Enero de 1970. Debes por tanto declarar el tipo tFecha mediante:

```
typedef time_t tFecha;
```

Al ser un número entero, la lectura/escritura de la fecha de/en fichero se realiza usando el operador habitual de extracción/inserción de los enteros. En la función de creación de un mensaje nuevo se obtendrá la fecha actual del sistema mediante (será necesario incluir la librería ctime):

```
mensaje.fecha = time(0);
```

Define la siguiente función para mostrar la fecha en formato Año/Mes/Dia, Hora/Mins/Segs:

```
void mostrarFecha(tFecha fecha){
    tm* ltm = localtime(&fecha);
    cout << 1900 + ltm->tm_year << "/" << 1 + ltm->tm_mon << "/" << ltm->tm_mday;
    cout << ", " << ltm->tm_hour << ":" << ltm->tm_min << ":" << ltm->tm_sec;
}
```

2.4. Ficheros de datos

El programa utiliza dos tipos de archivos diferentes: el que contiene la lista de usuarios junto a sus buzones de mensajes (solo hay un fichero de este tipo para todo el programa), y los que contienen la lista de chats de cada usuario (hay un fichero de este tipo por cada usuario). Los ficheros deberán tener exactamente el

formato que se indica y por lo tanto vuestro programa deberá funcionar con los ficheros de prueba que se os faciliten.

2.4.1. Fichero de usuarios: *usuarios.txt*

Al iniciarse la simulación, el servidor del ChatsApp cargará de este fichero los nombres de todos los usuarios registrados. Tras cada nombre de usuario, en la siguiente línea, vendrá un número N indicando el número de mensajes que hay en el buzón asociado al usuario, y tras él vendrán los N mensajes. Cada mensaje vendrá en una línea diferente, empezará con el nombre de usuario de su emisor, tras un espacio vendrá la fecha del mensaje (en formato entero), y tras otro espacio vendrá el texto del mensaje (hasta el salto de línea). Los mensajes no pueden contener por tanto saltos de línea. Este fichero deberá actualizarse al salir de la aplicación. Un ejemplo de fichero sería el siguiente:

```
casillas
1
pepe 1396693594 A ver si Carlo te pone ya en la liga!
dcosta
2
pepe 1396694182 Pero si seras piscinero...
simeone 1396694231 Como va ese golpe?
pepe
0
simeone
0
XXX
```

2.4.2. Ficheros de chats: *<NombreUsuario>_chats.txt*

Cada usuario tendrá un fichero con su lista de chats, incluyendo en cada uno la lista completa de mensajes que se han intercambiado en ese chat. Cada chat tiene como identificador el nombre del otro usuario con el que *<NombreUsuario>* está conversando. Así, el fichero contendrá una lista de identificadores de chat, para cada uno de ellos el número de mensajes en el chat, y después cada uno de los mensajes (en el mismo formato que en el punto anterior). Este fichero se lee cada vez que un usuario accede al sistema y se actualiza cuando se sale. Por ejemplo, el fichero de chats del usuario pepe, con nombre *pepe_chats.txt*, podría tener el siguiente contenido:

```
casillas
3
pepe 1396693523 Nuevo chat iniciado por pepe
casillas 1396693565 Vaya cantadita de Diego...
pepe 1396693594 A ver si Carlo te pone ya en la liga!
dcosta
3
dcosta 1396694038 Nuevo chat iniciado por dcosta
dcosta 1396694056 Casi me lesionas... siempre igual
pepe 1396694182 Pero si seras piscinero...
XXX
```

3. Entrega de la práctica

La práctica se entregará a través del Campus Virtual. Se habilitará una nueva tarea **Entrega de la Práctica 4** que permitirá subir un fichero comprimido .zip que incluya todos los ficheros fuente del proyecto (.h y .cpp). Además de subir el código fuente, se deberá pasar una entrevista con el profesor en la que se hará una demostración de la práctica y se responderá a las preguntas que el profesor realice.

4. Parte opcional

En el WhatsApp original, al entrar en un chat, no se muestran todos sus mensajes, sino solo los M mensajes más recientes, y el usuario puede, pulsando en el botón correspondiente, mostrar mensajes anteriores si lo desea. Se pide implementar el soporte necesario para implementar una funcionalidad similar en nuestro ChatsApp. De esta manera, al estar dentro de un chat, por defecto solo se mostrarán los M mensajes más recientes, y al escribir el mensaje especial \$todos\$ se mostrarán todos los mensajes del chat. Para ello se debe definir (e implementar las operaciones asociadas de) un nuevo tipo `tListaMensajesChat`, representado como un array circular, que mantenga los M mensajes más recientes, de forma que cuando el array esté lleno, el mensaje más antiguo se sobrescriba y se guarde en un fichero histórico asociado al chat, en el cual se guardarían todos los mensajes antiguos que no caben en la lista. El nombre del fichero sería `<NombreUsuario>_historico_<NombreChat>.txt`. Para añadir un mensaje nuevo al final del fichero se deberá abrir el fichero en modo “append” mediante `fichero.open(ios::app)`, y acto seguido simplemente hacer la escritura correspondiente.