Proyecto 1

Implementación Sockets

Profesor: Cesár Azurdia M.
Auxiliares: Hojin Kang, Mauricio Romero, Pablo Palacios, Sandy Bolufe.
Ayudante: Erwin Paillacan.

Quizás la aplicación más conocida de los *sockets* TCP/IP es el chat. En esta tarea se tiene por objetivo crear un chat de una página de compra y ventas de productos, en el cual se puede conversar con otros usuarios y realizar transacciones bancarias con ellos.

Este sistema de chat debe cumplir con ciertos requerimientos, los cuales son:

1. El servidor debe ser capaz de aceptar al menos 6 clientes distintos.
2. Cada cliente debe tener un identificador único [1]
3. Los mensajes recibidos deben ser mostrados en la consola de cada uno de los usuarios, siguiendo el formato:

```
<Identifier>: Message
<Identifer2>: Message2
Me: Message3
...
```

4. Cuando se conecte o desconecte un cliente al chat, se debe mostrar un mensaje en el servidor indicando que el usuario se conectó/desconectó. Sumado a lo anterior se debe mostrar un mensaje en el chat del usuario indicando una conexión/desconexión correcta, y finalmente un mensaje en cada uno de los usuarios que esté conectado en el chat indicando que se ha conectado/desconectado un usuario. Un ejemplo de lo anterior se muestra a continuación:

   Terminal servidor

```
[SERVER] Client: <Identifier2> connected!
[SERVER] Client: <Identifier3> connected!
[SERVER] Client: <Identifier3> disconnected!
[SERVER] Client: <Identifier2> disconnected!
```

[1] En Python, puede usar el método de la librería socket `getpeername()` o puede enviar una cabecera que solo deje conectar si es única.

Terminal cliente 2

```
Welcome to the best chat in the universe!
...
[SERVER] Client: <Identifier3> connected!
<Identifier3>: Hola!
Me: Hola!
[SERVER] Client: <Identifier3> disconnected!
Me: uwu
You have been disconnected from the best chat in the universe!
```

5. Cuando cada usuario ingrese al chat, se debe consultar por el balance que tiene en su cuenta de banco, siguiente el siguiente formato:

```
Welcome to the best chat in the universe!
[SERVER] Client: What is your balance?
50000
[SERVER] Client: The balance of 50000 has been set to your account.
```

   Notar que esta información solo se debe desplegar en el chat del usuario que se está conectando.

6. El servidor debe llevar un registro del balance de cada uno de los usuarios, el cual se irá actualizando según se hagan transferencias a otros usuarios. Notar que no se puede tener balance negativo, ya que en este caso la transacción sería inválida.

7. Como mínimo el chat debe tener los siguientes comandos:

   - `:q` - Permite que el usuario se desconecte del chat.
   - `:p - <Identifier> - Message` - Manda un mensaje privado *Message* al usuario con el identificador *Identifier*. Este solo debe ser desplegado en el chat del usuario que tiene el identificador *Identifier*.
   - `:u` - Despliega los usuarios que se encuentran conectados actualmente al chat.
   - Emojis: El chat debe tener como mínimo los siguientes emojis
     - `:smile` - Despliega la carita feliz `:-)`
     - `:angry` - Despliega la carita enojada `>:(`
     - `:confused` - Despliega la carita confundida `:S`
   - `:funds` - Entrega la cantidad de dinero que le queda al usuario en su cuenta. Esta información solo se debe desplegar en el chat del usuario.
   - `:t - <Identifier> - Amount` - Transfiere la cantidad *Amount* desde su cuenta al usuario que tiene el identificador *Identifier*. Es importante notar que esto implica disminuir la cantidad de su cuenta y aumentarla en la cuenta del usuario con el identificador *Identifier*. Recordar también que no se puede transferir una cantidad mayor al dinero que se tiene en la cuenta. En este caso debería tirar un error como:

```
Me: :funds
[SERVER] Client: You have 3000 dollars.
Me: :t - <Identifier> - 10000
[SERVER] Client: You don't have sufficient funds for the transaction.
```

8. La implementación debe ser en base a *threads*, donde se debe levantar un hilo por cada conexión nueva, en cada hilo debe ejecutarse una función que primero salude al cliente, muestre en la consola que se conectó un cliente nuevo y se mantenga recibiendo los mensajes hasta que se termine la conexión. [2]. Es importante notar que al utilizar *threads* se realizan procesos en paralelo, lo cual podría implicar que más de un usuario actualice los datos de una misma cuenta al mismo tiempo, con efectos inesperados. Se debe tener en consideración este probema. [3]

Una vez implementado el chat, usted debe entregar un reporte que contenga:

1. Introducción.
2. Marco teórico que debe al menos responder:
   - ¿Qué es un socket?
   - ¿Cómo funciona un socket?
   - ¿Qué tipo de socket utilizó en la implementación y cuáles son sus características?
3. Descripción del servidor: Describir la implementación que realizó. Una buena forma para esto es explicar las partes relevantes del código mostrando partes del código mismo.
4. Discusión y conclusiones.

Además deberá entregar los códigos implementados correctamente comentados.

[2] En Python, puede usar la libreria *threading*
[3] Para más información busque sobre secciones críticas.