TD 4 correction

Applications Reseaux

L3 Informatique 2021-22

A. Java Threads

1. Comment utiliser les Threads en Java ? Quelle est la méthode principale pour un Thread ?

- étendre la classe Thread:

```
class Exemple extends Thread {
    public void run(){
        ....
    }
}
```

```
(new Exemple()).start();
```

- implémenter l'interface Runnable:

```
class Exemple implements Runnable {
    public void run(){
        ....
    }

}
```

```
(new Thread(new Exemple()).start();
```

la méthode principale : **run( )**

A. Java Threads

2. *Ecrire un* Class SocketHandler *qui implement* Runnable. *Le constructor de cette* Class *prends un identifiant socket comme parameter et la méthode principale doit lire les donnée de ce socket, ligne par ligne, et les stocker dans un ensemble de* String.

```
class SocketHandler implements Runnable
{
    Socket soc; String[] data;

    public SocketHandler(Socket s)
    {
        this.soc = s;
    }
    public void run()
    {
        BufferedReader in = new BufferedReader(new InputStreamReader(soc.getInputStream()));
        for(int i=0; in.hasNext(); i++)
            data[i] = in.readLine();
    }
}
```

B. Serveur Multi-Thread

1. Comment peut-on construire un serveur (multi-thread) qui lit en parallèle de plusieurs client TCP sockets ? Utiliser la classe SocketHandler du dernier exercice.

```
ServerSocket ssoc = new ServerSocket(port);

while(true){
    new Thread(new SocketHandler(ssoc.accept() ) ).start();
}
```

B. Serveur Multi-Thread

2. Si on créé deux objets de type SocketHandler avec le même identifiant de socket, que se passe-t-il ? Si les deux threads lisent de la même socket client et écrivent sur l'écran chaque ligne de données reçus, qu'allons nous obtenir comme affichage?

problèmes d'interleaving possibles

3. Rappelons que les threads d'une même classe partagent leur mémoire. Si deux threads écrivent sur la même structure de donnée que passe-t-il ? Donner un exemple.

risque d'écrasement

4. Supposons que le système d'exploitation impose une limite de 1000 threads par processus. Pour le serveur multi-thread que vous avez écrit, y a-t-il un moyen d'accepter la 1001e connexion d'un client ?

utiliser un exécuteur

C. Communication avec Java NIO

1. Comment utiliser java.nio.channels pour la communication entre une socket serveur et une socket client ?

```
ServerSocketChannel ssc = ServerSocketChannel.open();
ssc.bind(new InetSocketAddress(port));

channel1 = ssc.accept();
channel1.configureBlocking(false);

ByteBuffer buffer = ByteBuffer.allocate(128);
channel1.read(buffer);
// ... Quand les donnés sont prêt
buffer.flip();
while (buffer.hasRemaining()) {
        b = buffer.get();
        // Traitement de données...
}
```

C. Communication avec Java NIO

2. Quels sont les avantages à utiliser Java NIO pour la communication entre sockets?

- Communication asynchrone (non-bloquant)
- Communication efficace

3. Pourquoi utilise-t-on un Selector pour les communications asynchrones?

On utilise un `Selector` pour pouvoir faire le communication en multiplexage entre plusieurs canals (`sockets`).

C 4. Écrire un serveur Mono-Thread, c'est-à-dire un serveur qui utilise un seul thread pour gérer les connexions avec jusqu'à 10 clients. Comment le serveur peut-il savoir quelle socket client est prête à lire ?

```
public class MonoThreadServer {

    public static int numClients=0;
    public static int MAX=10;

    public static void main(String[] args) throws IOException {

        ServerSocketChannel serverSocket = ServerSocketChannel.open();
        serverSocket.bind(new InetSocketAddress("localhost", 1234));

        Selector sel = Selector.open();
        serverSocket.configureBlocking(false);
        serverSocket.register(sel, SelectionKey.OP_ACCEPT);

        ByteBuffer buffer = ByteBuffer.allocate(256);

        while (true) {
            sel.select();
            Iterator<SelectionKey> iter = sel.selectedKeys().iterator();
            while (iter.hasNext()) {
                SelectionKey key = iter.next(); iter.remove();
                if (key.isValid()){
                    if (key.isAcceptable())
                            handle_New_Client(sel, serverSocket);
                    if (key.isReadable())
                            ReadClientData(buffer, key);
                } else  key.cancel();
            }
        }
    }
```

C 4. Serveur MonoThread…

- La méthode `handle_New_Client(Selector, Socket)` doit fait appel à `accept()` pour recuperer le socket du nouveau client et ensuite l'enregistrer avec le meme Selector, avec l'option **OP_READ**

Utiliser le compteur *numClients* et l'augmenter pour chaque client accepter. Quand le compteur arrive à 10, on ferme le serveur.

- La méthode `ReadClientData(Buffer, SelectionKey)` doit lire les donné envoyé sur le canal qui est prêt à lire et les stocker dans le Buffer. Utliser le méthode `SocketChannel.read(Buffer);`

Pour trouver la socket qui est prête à lire, il faut utiliser

`SelectionKey.channel()`

qui renvoie le canal correspondant à un socket prête à lire.