TP – Systèmes Distribués

Sockets UDP asynchrones

Licence Informatique $3^{ème}$ Année – UPPA

1 Fonctionnement des sockets UDP standards en Java

Le but de ce TP est de modifier le comportement en réception des sockets UDP en rendant la réception des paquets asynchrone : un paquet reçu poura être récupéré ultérieurement et le service de réception n'est pas bloquant. Par défaut, la méthode `receive` est bloquante[1], elle retourne seulement quand un paquet est reçu.

2 Classe `AsynchronousDatagramSocket`

La version asynchrone des sockets UDP sera implémentée par la classe nommée `AsynchronousDatagramSocket` que vous développerez. Cette classe n'est pas une spécialisation de la classe `DatagramSocket`[2].

2.1 Constructeurs

Les deux constructeurs de cette classe à définir sont les suivant et se basent sur ceux de la classe `DatagramSocket` :

- `public AsynchronousDatagramSocket() throws SocketException` : crée une socket et la lie sur un port quelconque déterminé par le système. L'exception `SocketException` est levée en cas de problème de création de la socket.
- `public AsynchronousDatagramSocket(int port) throws SocketException` : crée une socket et la lie sur le port passé en paramètre. L'exception `SocketException` est levée en cas de problème de création de la socket, ce qui est notamment le cas quand une socket est déjà liée sur le port choisi.

2.2 Méthodes

Par rapport à la classe `DatagramSocket`, le nombre de méthodes est fortement réduit. Nous allons uniquement définir des services basiques d'émission/réception. Les services à implémenter sont au nombre de 5 :

- `void send(DatagramPacket dp)` : même fonctionnement que pour la classe `DatagramSocket`, envoie le paquet passé en paramètre au destinataire précisé par le paquet.
- `boolean asynchronousReceive(DatagramPacket dp)` : si un paquet a été reçu précédemment et n'a pas encore été lu, alors retourne le premier paquet reçu non lu en positionnant les données du paquet passé en paramètre. Si aucun paquet n'est disponible, ne modifie pas le paquet passé en paramètre. La méthode retourne `true` si un paquet a été lu et `false` sinon. La méthode n'est pas bloquante, elle retourne immédiatement.
- `void synchronousReceive(DatagramPacket dp)` : si au moins un paquet a été reçu précédemment et n'a pas encore été lu, alors retourne le premier paquet reçu non lu en positionnant les données du paquet passé en paramètre. Si aucun paquet n'est disponible, reste en attente de la réception d'un paquet.
- `boolean available()` : retourne `true` si au moins un paquet a été reçu et n'a pas été lu. Retourne `false` sinon.
- `void close()` : ferme la socket.

Par souci de simplicité, les exceptions ne sont pas gérées. Une version plus évoluée pourrait par contre prendre en compte les problèmes d'émission/réception sur les sockets.

[1] Sauf si un paquet avait été envoyé mais pas encore lu.

[2] Il aurait été possible de spécialiser également cette classe.