TP1 I6

1 Sockets TCP et UDP (3h)

1.1 Un premier client et serveur TCP

Exercice 1 Écrire un serveur TCP qui écoute le port 2345 et qui renvoie le nombre de caractères reçus. Par exemple, si un client se connecte et envoie la chaîne “ABCD” le serveur doit répondre : “4”. Si le client envoie “stop”, le serveur ferme la session.

ATTENTION : Un appel comme `ligne=session.gets()` permet de récupérer des caractères reçus sur la socket, **y compris** un éventuel saut de ligne (caractère ASCII 10). Pour enlever le saut de ligne, il faut appliquer la méthode `chomp!()` sur la chaîne de caractère ; ensuite, la méthode `length()` permet de récupérer la longueur correcte, sans saut de ligne.

Exercice 2 Écrire un programme client qui demande à l'utilisateur de saisir des chaines de caractères au clavier. Le client envoie ces chaines au serveur de l'exercice précédent et affiche la réponse. Si l'utilisateur saisit “stop”, la session est fermée. Indications : voir les exemples `TCPSocket` dans les diapos 13-14 du cours.

Indication : utiliser `netcat` pour tester ces 2 programmes séparément avant de les tester ensemble.

1.2 UDP en diffusion

L'objectif de cette section est de réaliser un groupe de discussion pour tous les étudiants du réseau local. Le protocole UDP permet d'envoyer des segments en diffusion. Le port choisi est : 1234.
Exercice 3 Écrire un serveur et un client UDP. Le serveur UDP doit seulement écouter le port 1234 et afficher chaque segment reçu, ainsi que l'IP de l'émetteur. Le client envoie en diffusion les lignes saisies au clavier par l'utilisateur. Le client s'arrête si l'utilisateur introduit “stop”. Attention : pour le client, **l'adresse destination est l'adresse de diffusion** (disponible via `ifconfig`) ; il faut activer l'option `Socket::SO_BROADCAST`, (diapo 24 “UDP en diffusion” du cours).

1.3 Transfert de fichiers via TCP

Travailler en binôme.

Exercice 4 Écrire un serveur TCP et un client TCP pour transférer un fichier du client au serveur.
Indications :

- Utiliser le modèle client-serveur classique en utilisant les classes `TCPSocket` (pour le client) et `TCPServer` (pour le serveur)
- Exemple de code pour créer un fichier (s'il n'existe pas) et écrire quelques lignes :

```
1 fic=File.open("out.txt","w")
2 ....
3 fic.puts("une ligne")
4 end
```

1.4 Messagerie TCP sans fil d'exécution

Exercice 5 Écrire un serveur et un client TCP (port 3456) pour faire un système de messagerie qui respecte le protocole ci-dessous :

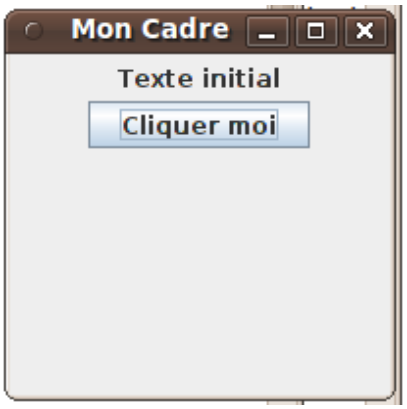
- Le client est celui qui commence à écrire une série de propositions (une par ligne). Pour l'instant, le serveur réceptionne ces lignes et il doit uniquement les afficher.
- Lorsque le client écrit “stop”, le serveur commence à écrire des propositions à la place du client (le client affiche ces propositions dans sur son terminal)

– Quand un des deux écrit “au revoir”, la session est fermée.
Indication : Chaque programme (serveur ou client) peut fonctionner en deux modes : récepteur ou émetteur. Une variable peut être utilisée pour indiquer le mode. Quand cette variable a la valeur `true`, le programme fonctionne comme récepteur (lecture de socket et affichage au terminal) ; sinon, il fonctionne comme émetteur (lecture de terminal et envoi à la socket).

2 Interfaces graphiques et sockets : minimum 1h à réserver

Exercice 6 Écrire un programme qui : 1) construit une fenêtre (cadre) Swing classique de taille 200×200 (avec un bouton “Cliquer moi” et un label “Texte initial” comme dans la figure ci-contre) ; et 2) ouvre le port 7777 à l'écoute afin de pouvoir communiquer par socket. Notre programme devrait recevoir une série de coordonnées (x,y) (une ligne pour chaque coordonné x ou y) : après avoir reçu un x et un y, le programme déplace la fenêtre à la position (x,y). Le programme s'arrête lorsqu'il reçoit la chaîne “stop” (soit pour x, soit pour y).

```
1 # partie 1: construction fenetre
2 require _________
3 require _________
4 import javax.swing.JButton
5 import ....
6
7 monPanel = JPanel.new
8 _________ = JLabel.new("Texte initial")
9 monBouton = JButton.new(_____________)
10 maFenetre = JFrame.new("Mon Cadre")
11
12 monPanel.add(_________)
13 monPanel.___(monBouton)
14
15 maFenetre.setContentPane(________)
16 maFenetre.setVisible(true)
17 maFenetre.setSize(200,_______)
18 maFenetre.setDefaultCloseOperation(JFrame::EXIT_ON_CLOSE)
19
20 #partie 2: socket
21
22 server = TCPServer.open(7777)
23 session = server.accept()
24 ligne1=""
25 ligne2=""
26 while(ligne1.chomp()!="stop"&&ligne2.chomp()!="stop") do
27     ...
28     ...
29     maFenetre.setLocation(x,y)
30     ...
31 end
```



Exercice 7 Écrire un client TCP pour envoyer des coordonnées au programme serveur ci-dessus. Le résultat des deux programmes devrait être une fenêtre qui se déplace de manière circulaire autour de l'écran. Plus précisément, les coordonnées x et y doivent suivre une loi décrite par :

$$x = centerX + r \cdot sin(\theta) \text{ et } y = centerY + r \cdot cos(\theta),$$

où θ représente un angle en radians. En fait, θ est incrémenté graduellement de 0 à 2π (rappel : 2π représente un angle de 360 degrés), voir aussi la figure ci-contre, ou bien les diapos/TD I5. N'oubliez pas l'instruction `sleep(...)`. Exemple de trajectoire :

$(300 + 100 \cdot sin(0 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(0 \cdot \frac{\pi}{12}))$,
$(300 + 100 \cdot sin(1 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(1 \cdot \frac{\pi}{12}))$,
...
$(300 + 100 \cdot sin(12 \cdot \frac{\pi}{12}), 300 + 100 \cdot cos(12 \cdot \frac{\pi}{12}))$.