TD n°1 *Socket, UDP, Client/Server, Diagrammes E/T*

Exercice 1. Sockets.

Quand on ouvre un fichier, on obtient un objet qui nous permet de lire et/ou écrire des données sur le disque : c'est un point de communication entre une application et le système de fichiers.

Une socket est un point de communication entre applications qui leur permet de communiquer au travers du réseau.

1.1 Mode non-connecté.

Pour ce premier TP, nous allons considérer les sockets en mode *non-connecté*, c'est à dire s'appuyant sur le protocole UDP (User Datagram Protocol).

- léger à mettre en œuvre
- modèle `send`/`recv` d'échange de messages
- aucune garantie de livraison !

1.2 Serveur.

On utilisera la bibliothèque `socket` de Python :

```
import socket
```

On crée une socket en mode non-connecté :

```
sock = socket.socket(type=socket.SOCK_DGRAM)
```

Pour que cette socket soit accessible par des applications sur le net, il faut qu'elle soit *publiée* sur une adresse publique. Dans l'internet, l'adresse d'une socket est une paire *(adresse IP, port)*. Par exemple, si mon adresse IP filaire est `192.168.82.209`, je peux publier la socket précédente à l'adresse `(192.168.82.209, 5555)` de la manière suivante :

```
sock.bind(("192.168.82.209", 5555))
```

En général, mon ordi possède plusieures interfaces réseau :

- le réseau filaire
- le réseau wifi
- la boucle locale

Pour publier ma socket simultanément sur toutes les interfaces, je peux faire ainsi :

```
sock.bind(("0.0.0.0", 5555))
```

l'adresse `0.0.0.0` signifie conventionellement *"toutes les interfaces."*

Maintenant le serveur doit attendre l'arrivée d'une *requète* en provenance d'un client :

```
BUFSIZE = 1024
data, addr = sock.recvfrom(BUFSIZE)
```

La méthode `recvfrom` bloque jusqu'à l'arrivée d'un message en provenance d'un client ; elle retourne alors une paire `(data, addr)`, où `data` est la charge utile du message, et `addr` est l'adresse de la socket du client. `BUFSIZE` est la taille maximale à utiliser pour le buffer de réception : à dimensioner de manière appropriée.

Le serveur renvoie alors une réponse en direction du client :

```
sock.sendto(b"salut", addr)
```

1.3 Serveur complet.

Bien sûr, puisqu'un serveur est sensé continuer à servir de nouveaux clients, il faudrait mettre une boucle après la publication de la socket. Donnez le code complet du serveur.

1.4 Client.

On crée une socket en mode non-connecté :

```
sock = socket.socket(type=socket.SOCK_DGRAM)
```

Puis on envoie un message au serveur :

```
sock.sendto(b"dring!", ("192.168.82.209", 5555))
```

Cette opération a aussi l'effet de publier notre socket en utilisant un port choisi automatiquement parmi les ports encore dispos. Heureusement, sinon le serveur ne saurait pas à quelle adresse répondre. Puis on attend la réponse du serveur :

```
BUFSIZE = 1024
data = sock.recv(1024)
```

La méthode `recv` est similaire à `recvfrom` mais renvoie juste la charge utile. Enfin on peut afficher la réponse sur le terminal :

```
print(data.decode())
```

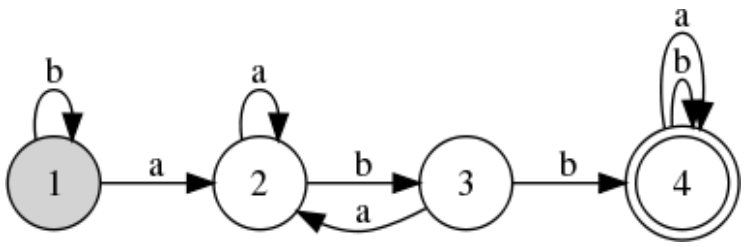
Exercice 2. Diagrammes états/transitions.

La conception de serveurs et clients peut s'appuyer avantageusement sur les diagrammes UML, en particulier les diagrammes états/transitions.

2.1 Serveur tic/tac. Ce serveur n'accepte qu'une requète `get`, et répond alternativement avec `tic` ou avec `tac`. Faites en le diagramme E/T.

2.2 Serveur de compteur. Ce serveur contient un compteur initialisé à 0, et accepte les requètes `incr` et `decr`. La requète `incr` renvoie une réponse `val(c)` avec la valeur courante du compteur, puis incrémente le compteur. La requète `decr` décremente le compteur puis renvoie une réponse `val(c)` contenant la valeur du compteur. Attention : si `decr` devait rendre la compteur négatif, alors `decr` ne le fait pas, le compteur reste à 0 et une réponse `err` est renvoyée. Faites en le diagramme E/T.

2.3 Serveur d'automate. Ce serveur est associé à l'automate déterministe complet suivant :



et se trouve initialement dans l'état initial 1. Il accepte les requètes suivantes :

- **`next(c)`** : `c` est le symbole suivant du mot qu'on exécute sur l'automate. Cette requète ne donne pas lieu à une réponse.
- `ask` : le serveur répond `yes` s'il se trouve dans un état final, `no` sinon.
- `reset` : le serveur bascule dans l'état initial. Cette requète ne donne pas lieu à une réponse.

Faites en le diagramme E/T.