Programmation concurrente et distribuée

Sylvain Conchon

Laboratoire Méthodes Formelles

Université Paris-Saclay, CNRS, ENS Paris-Saclay

sylvain.conchon@universite-paris-saclay.fr

Contenu du cours

- Introduction
- Programmation avec des Threads
- Programmation distribuée avec des sockets
- Quelques problèmes d'algorithmique distribuée : le problème du consensus, la notion d'horloge globale

Évaluation par mini projets

PROGRAMMATION DISTRIBUÉE AVEC DES SOCKETS

Les prises (sockets)

Il s'agit d'une API pour faire communiquer des processus. Ces prises (sockets en anglais) repose sur :

- Une architecture client/serveur
- Une notion d'adresse réseau : IP + port
- Des appels systèmes
- Des protocoles de communications (essentiellement TCP ou UDP)

API normalisée

Une prise est un mécanisme de communication bidirectionel entre des processus.

La manipulation des sockets se fait à travers une API (un ensemble de fonctions ou appels systèmes) normalisée.

Cette API est proposé dans de nombreux langages de programmation (C, C++, Java, Python, etc.) et sur la plupart des systèmes d'exploitation (Linux, Mac OS, Windows).

Documentations

En Python, la manipulation des sockets est réalisée à l'aide du module `socket` dont l'API se trouve à l'adresse suivante :

`https://docs.python.org/3/library/socket.html`

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.
3. La connexion du client a pour effet de créer une nouvelle socket sc chez le serveur de manière à ce que ce dernier puisse communiquer en privé avec le client à travers sc.

Modèle de communication

Les sockets sont des points de communication que des processus peuvent utiliser pour communiquer, i.e. envoyer ou recevoir des données.

Le schéma pour qu'un processus client communique avec un processus serveur est le suivant :

1. Le serveur crée une prise s et la connecte à une adresse IP sur un certain port p. Puis il attend que des clients se connectent à cette prise.
2. Un client crée une prise c et la connecte à l'adresse (IP + port) du serveur.
3. La connexion du client a pour effet de créer une nouvelle socket sc chez le serveur de manière à ce que ce dernier puisse communiquer en privé avec le client à travers sc.
4. La socket s du serveur est toujours disponible et elle peut donc servir à accepter de nouvelles connexions de nouveaux clients.

Appels systèmes

Les fonctions de l'API pour manipuler les sockets se trouvent dans le module `socket`. Une sur-couche pour manipuler des prises sécurisées se trouve dans le module `ssl`.

Les principales fonctions/méthodes sont les suivantes :

Création d'une prise : `socket`

Liaison de la prise à une adresse : `bind`

Ouverture du service : `listen`

Attente de connexion : `accept`

Démarrer une connexion : `connect`

Création

Un appel à

```
socket( family = a,
        type = t   )
```

permet de créer une prise.

- L'argument `a` est la famille d'adresses de la prise. On utilisera principalement `AF_INET` (par défaut, correspond à IPv4).
- L'argument `t` détermine le type de la prise. Plusieurs valeurs sont possibles, mais en pratique on utilisera `SOCK_STREAM` (pour un mode connecté, soit TCP) ou `SOCK_DGRAM` (pour un non connecté, soit UDP).

Il y a d'autres arguments, mais ils ne seront pas utilisés dans ce cours (voir la documentation officielle pour plus de détails).

La fonction renvoie un objet de la classe `socket`.

Liaison

Étant donnée une prise `s`, un appel à

```
s.bind( addr )
```

permet de connecter une prise à une adresse.

Le format de l'adresse `addr` passée en argument dépend de la famille d'adresses lors de la création de la prise. Pour IPV4 (`AF_INET`), il s'agit d'un couple (ip, port) où

- `ip` est une chaîne de caractère contenant une adresse IP comme `142.250.75.227` ou un domaine comme `www.google.com`
- `port` est un entier.

Ouverture

Pour ouvrir une prise `s`, il suffit d'appeler la fonction

```
s.listen(n)
```

Cet appel permet d'attendre des connexions sur la prise `s` en précisant le nombre maximum `n` de requêtes non encore traitées.

Pour une prise IPV4, cela a pour effet que le système enregistre le nouveau service à l'adresse IP et sur le numéro de port indiqués lors de la connexion de la prise.

Un appel `lsof -n -i4TCP` dans le terminal permet de voir ce nouveau service.

Attente

Pour attendre des connexions sur une prise, il suffit d'appeler la méthode

```
s.accept()
```

Un appel à cette méthode bloque le programme en attendant qu'un client se connecte à la prise `s`.

Quand un client se connecte, la fonction renvoie une nouvelle prise pour communiquer avec ce client, ainsi que l'adresse du client.

La prise `s` reste disponible pour d'autres connexions.

Démarrer

Étant donnée une prise `s`, on établit une connexion à l'aide d'un appel à la méthode suivante :

```
s.connect(addr)
```

Cet appel permet de débuter une communication avec un serveur qui attend des connexions à une adresse `addr`

Cet appel est bloquant tant que la connexion ne s'est pas établie.

En interne, l'effet de cette connexion est de brancher la prise `s` à une adresse sur la machine locale (choisie par le système).

Une fois connectée, on peut envoyer ou recevoir des données sur la prise `s`.

Fermer

Pour fermer une prise, il suffit d'appeler

```
s.close()
```

L'appel à cette méthode désalloue toutes les ressources systèmes attribuées à une prise. Cela peut prendre du temp. Pour fermer rapidement la prise, il faut appeler la méthode

```
s.shutdown(how)
```

où `how` permet de fermer finement la prise, en lecture (`SHUT_RD`), en écriture (`SHUT_WR`) ou les deux (`SHUT_RDWR`).

Pour éviter d'attendre que le système permette la réutilisation d'un couple (ip, port), on peut configurer la socket (après sa création) à l'aide de la fonction `setsockopt`. Par exemple,

```
s.setsockopt(SOL_SOCKET,SO_REUSEADDR, 1)
```

permet la réutilisation immédiate d'une adresse.

Échange de données

Deux méthodes (de bas niveau) sont utilisées pour échanger des données sur une prise `s` :

```
s.send(d)
```

```
s.recv(n)
```

Les données `d` envoyées par `s.send(d)` sont nécessairement des chaînes d'octets (objets *bytes* en Python). La fonction renvoie le nombre d'octets envoyés. En cas de problème, elle renverra 0.

Les valeurs reçues par `s.recv(n)` sont aussi des *bytes*. L'entier `n` passé en argument indique la taille maximale des données pouvant être reçues en un appel. Il est nécessaire de faire plusieurs appels à cette fonction si la quantité de données à recevoir est plus grande que `n`.

Chaînes d'octets

En Python, les chaînes d'octets sont représentées comme des chaînes de caractères auxquelles on ajoute un préfixe avec la lettre `b`, comme ci-dessous :

```
b'Hello World!'
```

Une autre manière de produire une chaîne d'octets est d'utiliser la méthode `encode` :

```
s = 'Hello World!'
s.encode('utf8')
```

De la même manière, une chaîne d'octets pourra être interprétée comme une chaîne de caractères en utilisant la méthode `decode`.

```
b.decode('utf8')
```

I/O avec des sockets

Python fournit le module `select` pour une gestion très fine des entrées-sorties, en particulier avec les prises. La fonction `select` de ce module permet de monitorer les prises qui sont prêtes en lecture ou en écriture. Étant donnée une liste de prises `p` , l'utilisation la plus courante de `select` est la suivante :

```
r, w, e = select(p,[],[],to)
```

Cet appel renvoie trois sous-listes de `p` qui contiennent :

- `r` = prises sur lesquelles il y a des données à lire (qui sont déjà dans le buffer)
- `w` = les prises qui ont de la place dans leur buffer et donc prêtent pour recevoir des données
- `e` = les prises qui ne sont pas opérationnelles.

Le paramètre optionnel `to` permet d'attendre jusqu'à une limite de temps (exprimée en secondes avec un nombre flottant).