Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Protocoles & Prog. Réseau

TP n°1

Introduction à la programmation réseaux

Fonctionnement du modèle « client/serveur » sous TCP

Vous trouverez les explications nécessaires à partir de la page 74 du support Python

1 – Écrire un programme **serveur** (utilisant `accept`) :
- ▷ attendant sur le port 8080 ;
- ▷ affichant sur la sortie standard (l'écran) les **lignes** qu'il reçoit depuis le client qui lui est connecté.

Optimisation

Lorsque votre programme se termine, il restitue le numéro de port qu'il utilisait.
Néanmoins, pour des raisons de protection (éviter que des anciens messages en cours de transit n'arrivent sur un port qui a été redonné à un nouveau programme), un **temps d'attente assez long** est mis en place par le système d'exploitation.
Pour pouvoir **récupérer immédiatement** l'usage du même numéro de port dans la phase de développement du programme serveur, il est possible de désactiver cette temporisation en configurant la socket :

```
ma_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

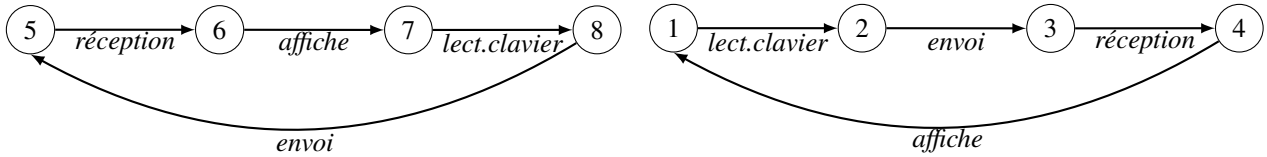
Vous pourrez tester votre programme avec l'outil socat (voir fin de la fiche pour la documentation).

2 – Écrire un programme **client** (utilisant `connect`) envoyant tout ce que l'on tape au clavier vers le serveur (envoi de lignes). *Vous testerez ce programme avec celui de la question 1.*

3 – Écrire deux programmes, l'un client et l'autre serveur, permettant le dialogue entre deux utilisateurs par alternance (« chat » élémentaire).
L'échange se déroule de la manière suivante :

1) *Le client lit une ligne au clavier*
2) Le client transmet une ligne au serveur
3) Le client attend une ligne du serveur
4) *Le client affiche la ligne reçue à l'écran*
5) Le serveur attend une ligne du client
6) *Le serveur affiche la ligne reçue à l'écran*
7) *Le serveur lit une ligne au clavier*
8) Le serveur envoie une ligne au client

Chaque programme réalise une boucle sans fin en « cyclant » entre les différentes lignes de l'algorithme :



Que peut-il arriver si on commence **différemment** le travail du client et du serveur ?

Utilisation de processus exécutés de manière concurrente

Vous trouverez les explications nécessaires à partir de la page 71 du support Python

4 – Sous Python, un nouveau processus peut être créé à partir du processus courant à l'aide du « `fork` ».

Les deux processus obtenus après l'exécution de l'instruction fork, sont exécutés de manière concurrente et partage l'ensemble des entrées/sorties du processus initial, **ainsi que les sockets**, ce qui permettra à chacun de ces processus de s'occuper **d'un sens de la communication**.

Question :
Améliorer les programmes de client et de serveur des questions 2 & 3 de manière à ce que chaque interlocuteur puisse transmettre et recevoir une ligne de données à l'autre dès que possible.

Gestion simultanée de plusieurs clients

5 – Réfléchir (sans programmer !) à une *nouvelle version* du serveur de la question 4, qui permettrait à plusieurs clients de discuter simultanément entre eux :

i. Le serveur est connecté simultanément à plusieurs clients ;
ii. Le serveur ne sert plus à communiquer directement (pas de saisie de ligne au clavier).
iii. Le travail du serveur consiste à rediriger les différents messages échangés entre tous les clients (une ligne reçue d'un client est renvoyée à tous les autres clients).

a. Dans un premier temps, on considérera qu'au lancement du serveur, le nombre d'interlocuteurs attendus est connu (par exemple 3 interlocuteurs).
Est-ce facilement programmable ?
b. Dans un second temps, que se passe-t-il si l'on veut gérer un nombre inconnu de clients ?

Gérer plusieurs connexions simultanées avec des événements : « Être notifier »

Vous trouverez les explications nécessaires à partir de la page 81 du support Python

6 – Écrire le serveur de la question 5, pour traiter un nombre variable de clients dans n'importe quel ordre d'intervention et de connexion, en utilisant soit l'instruction `select`.
Remarques :

- *Il n'est pas nécessaire d'utiliser de « `fork` » dans cette question pour la programmation du serveur.*
- *Le client est le même que celui écrit à la question 4 (il peut être remplacé par la commande `socat`).*

Documentation « socat »

`Socat` est une version améliorée de la commande « netcat », le « *couteau suisse des réseaux* », qui vous permet de :

- ⋆ communiquer en mode client ou serveur à l'aide de **TCP** ou **UDP**, en IPv4 comme en IPv6 ;
- ⋆ communiquer avec des « sockets unix » ;
- ⋆ utiliser du chiffrement pour les communications ;
- ⋆ *etc.*

La syntaxe de la commande `socat` est la suivante `socat [options] <address1> <address2>` :

- ◇ ouvrir une connexion TCP en tant que client ;

```
$ socat STDIO TCP:192.168.0.1:80
```

À partir de ce moment, il envoie ce qui est lu au clavier (entrée standard) et affiche, sur la sortie standard, ce qu'il reçoit (`Ctrl-C` pour arrêter).

- ◇ attendre une connexion TCP en tant que serveur ;

```
$ socat TCP-LISTEN:60567,reuseaddr STDIO
```

Attend une connexion sur le port `60567`, puis envoie l'entrée standard et affiche sur la sortie standard.

- ◇ Envoyer le contenu d'un fichier au travers d'une connexion TCP ;

```
$ socat OPEN:mon_fichier TCP:192.168.0.1:6789
```

- ◇ recevoir du contenu au travers d'une connexion TCP et le rediriger vers la sortie standard ou un fichier ;

```
$ socat TCP-LISTEN:60567 OPEN:mon_fichier,create
```

- ◇ envoyer des paquets UDP depuis le contenu d'un fichier ;

```
$ socat OPEN:contenu_paquet UDP-SENDTO:agate.unilim.fr:60567
```

- ◇ recevoir des paquets UDP dans un fichier ;

```
$ socat udp-recvfrom:60567 OPEN:contenu_paquet,create
```

- ◇ suivre ce que fait socat :

```
$ socat -d -d ...
```

- ◇ En cas de possibilité de blocage : connexion TCP au serveur pas garantie, inactivité du protocole, on peut mettre l'option `-T 1` pour indiquer à `nc` d'attendre au plus 1 seconde avant de quitter :

```
$ socat -T 1 TCP-LISTEN:60567 STDIO
```

Dès que le serveur accepte une connexion, il attend de recevoir des données pendant 1 seconde seulement.