ENSEIRB-MATMECA

PROGRAMMATION RESEAU : LES SOCKETS

Patrice KADIONIK
http://kadionik.vvv.enseirb-matmeca.fr/

TABLE DES MATIERES

1. BUT DES TRAVAUX PRATIQUES

Le but de ces TP est de maîtriser la programmation réseau par *sockets* en langage C sous UNIX.

2. UTILISATION DES COMMANDES UNIX

On n'oubliera pas qu'à tout moment on peut avoir des informations en ligne sur une commande via la commande *man*.

Exemple :

```
% man  socket
```

2.1. Utilisation des commandes d'analyse réseau

Dans cette partie, on étudie les principales commandes utilisateur d'analyse réseau.

1. Commande *netstat* :
   En s'aidant du manuel en ligne, préciser le rôle de la commande *netstat*. A l'aide de l'option :

   ```
   % netstat –nr
   ```

2. A l'aide de l'option :

   ```
   % netstat –a
   ```

   Retrouver les services UDP et TCP actifs sur le PC ainsi que la liste des connexions TCP en cours. Remarquer la notation `host:numero_port` et l'état courant d'une connexion TCP (voir automate d'états d'une connexion TCP donné en annexe).

3. Commande *telnet* :
   En s'aidant du manuel en ligne, préciser le rôle de la commande *telnet*. On peut utiliser la commande *telnet* autrement que pour se connecter au service *telnet* par défaut en précisant un numéro de port :

   ```
   % telnet host numero_port
   ```

   Se connecter au service *ftp* de `brahmane` par *telnet*. Dans une deuxième fenêtre de terminal, à l'aide de la commande *netstat*, remarquer une entrée supplémentaire correspondant à la connexion *ftp* en cours.

2.2. Etude de quelques services Internet

On utilisera à chaque fois la commande *telnet* pour se connecter au service désiré en précisant le bon numéro de port.

4. Service *ftp* :
   Se connecter par *telnet* au service *ftp* de brahmane. On essayera d'utiliser quelques commandes du protocole FTP :
   USER toto
   PASS toto
   HELP
   QUIT
   A l'aide de la commande *netstat*, on retrouvera les paramètres de la connexion en cours (numéros de port utilisés et évolution de l'état de la connexion).

5. Service *Web* :
   Se connecter par *telnet* au service web de la machine www. Une fois connecté, envoyer le caractère "RETURN". Que se passe-t-il ? Même chose en envoyant les caractères ESPACE puis "RETURN". Que vous renvoie le serveur web ? Quel est le type des données renvoyées par le serveur ? Le protocole HTTP utilisé par un serveur web est structuré sous forme de commandes ASCII dont la structure générale est donnée ci-après (RFC1945) :

```
COMMANDE action HTTP/1.0
Autres infos passées au serveur
Un RETOUR CHARIOT (RETURN) →
Un RETOUR CHARIOT →
Données de l'utilisateur
```

La commande HTTP peut être GET, PUT, POST et HEAD suivant l'action demandée (généralement GET et l'action étant alors le nom d'un fichier HTML du serveur web).

Un exemple de commande envoyée au serveur web est celui-ci (récupération de la page d'accueil) :

```
GET / HTTP/1.0
→
→
```

Le serveur en retour renvoie un code d'erreur dont les principaux sont :

```
200 : OK
204 : No content
400 : Bad request
403 : Forbidden
404 : Not found
408 : Request timeout
```

Un exemple de données retournées par le serveur web est :

```
HTTP/1.0 200 OK
Date: Wed, 07 Nov 2012 13:05:03 GMT
Server: Apache/2.2.14 (Ubuntu)
...
Content-Length: 4406
Content-Type: text/html; charset=utf-8
```

En vous aidant de l'exemple précédent et en utilisant *telnet*, récupérer le fichier HTML *index.html*. Quel est le code de retour ?

3. SEANCES 1, 2 ET 3 : PROGRAMMATION RESEAU PAR SOCKETS

1. Dans son répertoire de travail, se créer un répertoire de travail `tp_reseaux` et s'y placer :

   ```
   % cd
   % mkdir tp_reseaux
   % cd tp_reseaux
   ```

2. Y recopier tous les fichiers de `~kadionik/pub` :

   ```
   % cp ~kadionik/pub/* .
   ```

3. Analyse d'un programme client TCP :
   Editer le fichier `myftp0.c` et analyser le code source. Quel type de *socket* utilise-t-on ? Retrouve-t-on l'enchaînement classique des appels systèmes dans ce cas ? Que fait ce programme ? Compiler ce programme. On utilisera le script maison `cur` :
   ```
   % ./cur myftp0
   ```

   Exécuter le programme et le tester avec le serveur `ftp` de `brahmane`.

4. Client TCP *ftp myftp* :
   Copier le fichier `myftp0.c` dans le fichier `myftp.c`. Modifier le code source `myftp.c` pour créer l'équivalent de la commande *"telnet brahmane 21"*. Compiler et tester. Dans une fenêtre de terminal, à l'aide de la commande *netstat*, remarquer une entrée supplémentaire correspondant à la connexion en cours.

5. Client TCP *mydateTCP* :
   Utiliser la commande *telnet* pour tester le service *daytime* de `brahmane`. A l'aide des fichiers de configuration, retrouver le numéro de port et le protocole de transport Internet à utiliser.
   Copier le fichier `myftp.c` dans le fichier `mydateTCP.c`. Modifier le code source `mydateTCP.c` pour pouvoir récupérer par TCP la date du serveur de *daytime*. On passera en argument au programme le nom du serveur.
   Compiler et tester en prenant comme service *daytime* celui de `brahmane`.

6. Client UDP *mydateUDP* :
   Copier le fichier `myftp.c` dans le fichier `mydateUDP.c`. Modifier le code source `mydateUDP.c` pour pouvoir récupérer par UDP la date du serveur de *daytime*. On passera en argument au programme le nom du serveur.
   Compiler et tester en prenant comme service *daytime* celui de `brahmane`.

7. Analyse d'un programme serveur TCP :
   Editer le fichier `pingserveurTCP0.c` et analyser le code source. Quel type de *socket* utilise-t-on ? Retrouve-t-on l'enchaînement classique des appels systèmes dans ce cas de serveur ? Que fait ce programme ? Compiler ce programme.
   Exécuter le programme en utilisant comme programme client *telnet* en choisissant un numéro de port supérieur à 1024, par exemple le numéro de port 22222. Dans une fenêtre de terminal, à l'aide de la commande *netstat*, remarquer une entrée supplémentaire correspondant à la connexion TCP en cours.

8. Serveur TCP *pingserveurTCP* :
   Copier le fichier `pingserveurTCP0.c` dans le fichier `pingserveurTCP.c`. Modifier le code source `pingserveurTCP.c` pour que le serveur renvoie vers le client tout ce qu'il a reçu de sa part (équivalent d'un écho/ping). Compiler et tester en utilisant comme programme client *telnet* en choisissant le numéro de port 22222. Dans une fenêtre de terminal, à l'aide de la commande *netstat*, remarquer une entrée supplémentaire correspondant à la connexion TCP en cours. Peut-on prendre n'importe quelle valeur de port ?

9. Serveur UDP *pingserveurUDP*, client *TCP* *pingclientUDP :*
   Copier le fichier `pingserveurTCP.c` dans le fichier `pingserveurUDP.c` et le fichier `mydateUDP.c` dans le fichier `pingclientUDP.c`. Modifier le code source `pingserveurUDP.c` mais en utilisant ici le protocole UDP. Modifier le code source `pingclientUDP.c` mais en utilisant ici le protocole UDP. Mêmes questions que précédemment.

10. Serveur *wwwserveur* :
    Quel protocole utilise-t-on quand on accède à un serveur web ? Créer le fichier `wwwserveur.c` (serveur TCP) qui renvoie vers le client une page d'accueil HTML. Aucun test ne sera fait au niveau du respect du protocole HTTP par le serveur. Compiler et tester avec comme programme client *Firefox*. L'URL à rentrer est :
    `http://localhost:numero_port/`

11. Serveur TCP *lotoserveur* :
    Créer le fichier `lotoserveur.c` (serveur TCP) qui renvoie vers le client une chaîne de caractères contenant 6 numéros de loto (entre 1 et 49) tirés aléatoirement (voir l'appel système `rand()`). Compiler et tester comme précédemment avec `telnet`.

ANNEXE : AUTOMATE D'ETATS TCP

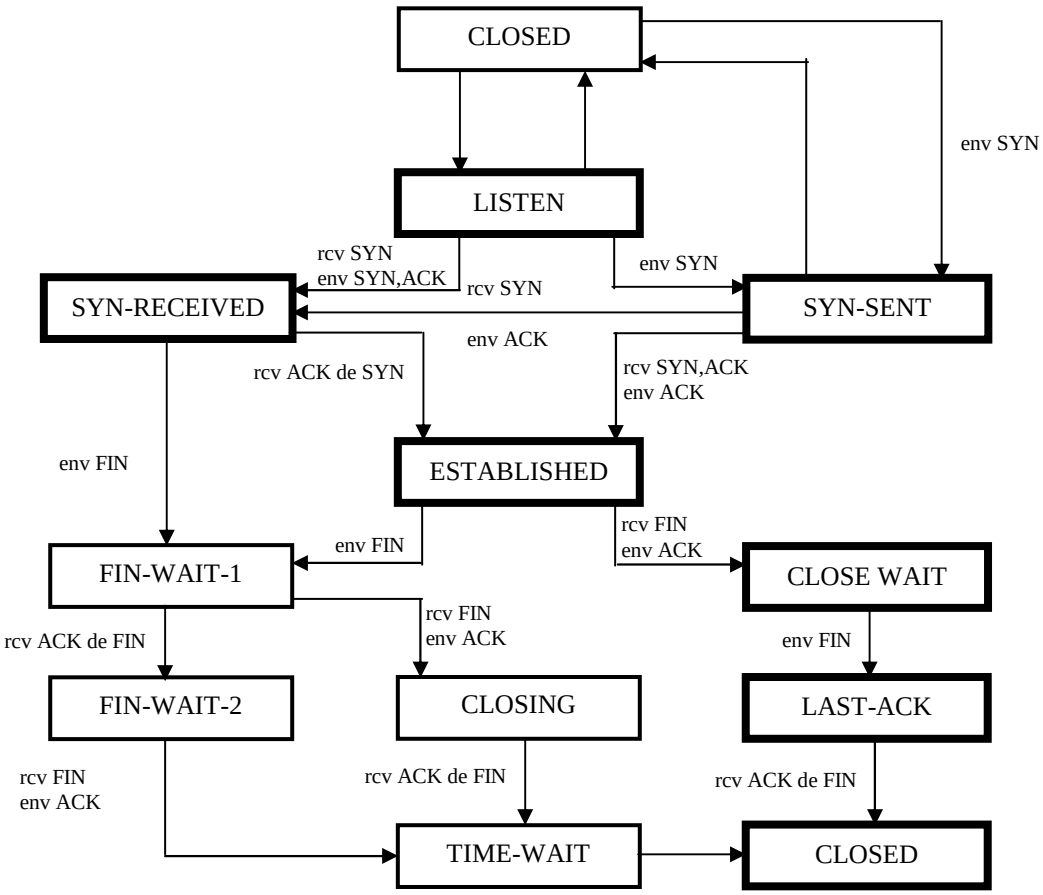
Une connexion TCP connaît plusieurs états durant sa durée de vie. Les états définis sont : LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, et CLOSED.

CLOSED est dit fictif car il correspond à une situation où la connexion elle-même n'existe plus.

Voici un descriptif des états TCP :

- **LISTEN** : la connexion reste en attente d'une requête de connexion externe par un TCP distant. Cet état est atteint après une demande de connexion passive.
- **SYN-SENT** : la connexion se met en attente d'une requête de connexion, après avoir envoyé elle-même une requête à un destinataire.
- **SYN-RECEIVED** : les deux requêtes de connexion se sont croisées. La connexion attend confirmation de son établissement.
- **ESTABLISHED** : la connexion a été confirmée de part et d'autre et les données peuvent transiter sur la voie de communication. C'est l'état stable actif de la connexion.
- **FIN-WAIT-1** : sur requête de déconnexion émise par l'application, la connexion demande la confirmation d'une requête de déconnexion qu'elle a elle-même émise vers le distant.
- **FIN-WAIT-2** : la connexion se met en attente d'une requête de déconnexion par le distant, une fois reçue la confirmation de sa propre requête.
- **CLOSE-WAIT** : la connexion se met en attente d'une requête de déconnexion émise par l'application.
- **CLOSING** : la connexion attend la confirmation de sa requête de déconnexion par le TCP distant, lequel avait auparavant émis sa propre requête de déconnexion.
- **LAST-ACK** : la connexion attend la confirmation de sa requête de déconnexion, émise suite à une requête similaire à l'initiative du distant.
- **TIME-WAIT** : un temps de latence avant fermeture complète du canal, pour s'assurer que toutes les confirmations ont bien été reçues.
- **CLOSED** : la connexion n'existe plus. C'est un pseudo état.

La figure suivante montre l'enchaînement des états et les différentes trames émises. Il occulte par contre le traitement des fautes ainsi que tous les autres événements qui ne sont pas en relation avec les changements d'état.



Machine d'états d'une connexion TCP