`socklab`

Laboratoire d'étude des *sockets* Internet

Guide d'utilisation

Guide de référence

Accès aux sources pour installation :
https://github.com/drakkar-lig/socklab

Revision : 386

6 septembre 2022

Table des matières

Chapitre 1

Introduction

Au sein d'une même machine, les processus disposent d'un certain nombre d'outils pour communiquer entre eux : fichiers, *pipes*, signaux, files de messages, etc. mais ces outils ne sont valables qu'à la condition que les processus concernés s'exécutent sur la même machine.

Deux processus s'exécutant sur deux machines différentes ne pourront communiquer que si ces machines sont reliées par réseau. Mais ils devront utiliser ce que l'on appelle une interface de programmation réseau : il s'agit d'un ensemble de primitives dont l'objectif est de mettre en relation des processus sur des machines différentes.

Il existe plusieurs interfaces de programmation réseaux sur les systèmes UNIX ; la plus utilisée est actuellement les *sockets* BSD (*Berkeley Software Distribution*). Le terme socket sert d'abord à désigner les points d'accès que les processus doivent manipuler pour pouvoir envoyer ou recevoir des données. Par extension, ce terme désigne toute l'interface de programmation (on parle alors des sockets).

L'utilisation de cette interface de programmation n'est pas aisée : il est rare que deux processus (même situés sur une même machine) arrivent à communiquer du premier coup par l'intermédiaire de sockets. En effet, les primitives sont généralement mal employées, ou les structures de données mal utilisées, si bien que les premières tentatives sont souvent des échecs (sans parler des erreurs qui surviennent directement lors de la compilation des programmes).

`socklab` est un utilitaire vous permettant de manipuler les sockets, sans avoir à compiler la moindre ligne de code. Ce laboratoire d'étude des sockets se présente sous la forme d'un interpréteur de commandes, où à chaque commande est associé l'appel d'une primitive de l'interface (parfois plusieurs primitives). À chaque fois que vous lancez une commande, `socklab` exécute la primitive correspondante et vous donne un compte rendu de cette exécution.

Si vous découvrez les sockets, `socklab` vous aidera à comprendre le rôle des différentes primitives, et la façon dont elles doivent être utilisées (ce qui peut être fait, ce qui est interdit, etc.). Si vous connaissez déjà les sockets, `socklab` pourra vous être d'une grande utilité pour étudier leur fonctionnement ou leurs limites dans des situations particulières.

Chapitre 2

Présentation générale du laboratoire

2.1 Classification des commandes

Les commandes que `socklab` met à votre disposition sont classées en trois catégories :

1. les commandes de *contrôle* des sockets, permettant de les créer, les supprimer ou modifier leur mode de fonctionnement (manipulation d'options) ;
2. les commandes d'*exploitation* des sockets, permettant d'envoyer ou de recevoir des messages à travers les sockets ;
3. les commandes *spéciales* de `socklab` qui n'ont pas un lien direct avec la manipulation des sockets (par exemple : l'affichage de l'état des sockets créées, l'affichage de la liste des commandes disponibles).

2.2 Modes de fonctionnement de `socklab`

`socklab` peut être utilisé dans trois modes de fonctionnement différents. Le choix d'un de ces trois modes détermine la liste des commandes de contrôle et d'exploitation disponibles pendant l'utilisation de `socklab` (dans les trois cas, la liste des commandes spéciales est invariante) :

- En mode standard (mode par défaut), une commande est associée à chaque appel système permettant de manipuler les sockets (c'est à dire, les primitives telles que `socket()`, `bind()`, `send()`). En exécutant une liste de commandes, vous pouvez observer les résultats que vous obtiendriez si vous exécutiez un programme C dans lequel serait enchaînée la liste des primitives équivalentes.
- En mode TCP, la panoplie de commandes proposées est orientée vers une utilisation exclusive des sockets avec le protocole TCP. Les primitives n'ayant aucun rapport avec le protocole TCP ne sont pas représentées : les commandes disponibles sont donc moins nombreuses qu'en mode standard. Par ailleurs, de nouvelles commandes sont introduites afin de simplifier l'utilisation de `socklab` (ces commandes réalisent par exemple l'enchainement de plusieurs primitives).
- De façon similaire, en mode UDP, la liste des commandes proposées est orientée vers une utilisation exclusive des sockets avec le protocole UDP.

Le mode standard est normalement utilisé pour étudier les primitives de l'interface de programmation des sockets, comprendre leur fonctionnement et découvrir leurs limites.

Alors que les modes UDP et TCP sont normalement utilisés pour étudier ces protocoles respectifs (au niveau service ou au niveau fonctionnement, mais dans le dernier cas, l'utilisation conjointe d'un analyseur de traffic tel que `snoop`, `tcpdump` ou `ethereal` est cependant nécessaire).

2.3 Principe d'utilisation

Quel que soit le mode de fonctionnement utilisé, le but de `socklab` est de fournir une interface pour créer, exploiter et supprimer des sockets. Au cours d'une même session avec `socklab`, vous pouvez manipuler plusieurs sockets simultanément. Cela signifie qu'entre chaque commande exécutée, `socklab` tient à jour la liste des sockets que vous avez créées. À tout moment vous pouvez choisir n'importe quelle socket précédemment créée, et l'utiliser pour envoyer ou recevoir des données grâce aux commandes appropriées.

2.4 Lancement de `socklab`

`socklab` figure normalement dans le répertoire `/usr/local/bin/` des stations. Ce répertoire est normalement inclus d'office dans la variable d'environnement `PATH`.

Une fois lancé, `socklab` affiche une bannière de présentation, suivie du prompt de l'interpréteur de commandes :

```
sparc1 # socklab
socklab - laboratoire d'etude des sockets INTERNET
------------------------------------------------------------------------
socklab>
```

Par défaut, `socklab` fonctionne en mode standard. Pour l'utiliser en mode UDP ou TCP, vous devez précisez respectivement l'option `udp` ou `tcp` sur la ligne de commande. Le prompt de `socklab` est alors modifié pour indiquer le mode utilisé. Par exemple :

```
sparc1 # socklab tcp
socklab - laboratoire d'etude des sockets INTERNET
------------------------------------------------------------------------
socklab-TCP>
```

Chapitre 3

Exemple d'utilisation de `socklab`

Dans cette section nous allons illustrer l'utilisation de `socklab` en mode standard. Commencez donc par lancer le laboratoire dans ce mode en exécutant `socklab` sans option :

```
sparc1 # socklab
socklab - laboratoire d'etude des sockets INTERNET
------------------------------------------------------------------------
socklab>
```

Dans ce mode, à chaque primitive de l'interface de programmation des sockets est associée une commande. Toutes les commandes portent donc le même nom que la primitive qu'elles manipulent. Par exemple, pour créer une socket, l'appel système s'appelle `socket()`. Demandez l'exécution de la commande `socket` :

```
socklab> socket
Protocole ? [tcp/udp]:
```

La primitive `socket()` a normalement trois paramètres : le domaine dans lequel la nouvelle socket doit être créée, son type et le protocole à utiliser dans le type donné. Le laboratoire ne permet d'étudier que les sockets du domaine Internet. Deux types sont seulement représentés (STREAM ou DATAGRAM), et à chaque type ne correspond qu'un seul protocole (TCP ou UDP respectivement). C'est pourquoi `socklab` vous demande seulement le protocole que vous désirez utiliser avec la nouvelle socket. Choisissez UDP.

`socklab` dispose alors de tous les renseignements nécessaires pour appeler la primitive `socket()`. Si l'appel n'échoue pas, `socklab` vous retourne l'*identificateur* de la socket qu'il vient de créer (il s'agit du numéro de descripteur que la primitive `socket()` retourne). C'est par cet identificateur que vous pourrez désormais manipuler cette socket :

```
socklab> socket
Protocole [tcp/udp] ?: udp
La socket est identifiee par l'identificateur 3

socklab>
```

Afin de ne pas rendre l'utilisation de `socklab` trop fastidieuse, les noms des commandes peuvent être donnés en abrégé. De plus, quand une commande possède des paramètres (par exemple, le type de protocole pour la commande socket), ils peuvent être directement saisis sur la ligne. Ainsi, pour créer une nouvelle socket TCP, vous pouvez le demander de différentes manières : `socket tcp`, `sock tcp`, `s tcp` ou `s t` sont des ordres équivalents. Comme le paramètre est directement passé sur la ligne, `socklab` vous rend le prompt dès qu'il a exécuté la primitive `socket()` :

```
socklab> sock udp
La socket est identifiee par l'identificateur 4

socklab>
```

Depuis le début de cette démonstration, deux sockets ont donc été créées. Pour le vérifier, exécutez la commande `status` :

```
socklab> status
 Id      Proto       Adresse             Connexion      Type   RWX ?
 ------------------------------------------------------------------------
 3       UDP         -                   -               ipv4   .W.
>4       UDP         -                   -               ipv4   .W.

socklab>
```

Cette commande permet d'afficher tous les renseignements rattachés aux sockets qui ont été créées (la forme abrégée de la commande status est `=`). Pour chaque socket affichée, `Id` est sont identificateur, `Proto` est le type de protocole utilisé, `Adresse` est l'adresse qui lui a été éventuellement affectée (dans l'exemple précédent, ce n'est pas le cas), `Connexion` indique si une connexion est établie, `Type` indique le type de la socket (`ipv4` ou `ipv6`) et `RWX` indique ce qu'il est possible de faire (`R` = read, `W` = write, `X` = eXception, cf. **??**). Notez la présence du signe `>` à gauche de la deuxième socket. Ce signe indique quelle est la socket qui est a été dernièrement créée ou manipulée.

Vous allez maintenant utiliser la commande `bind`, qui fait appel à la primitive du même nom pour affecter une adresse à une socket. Dans le domaine Internet, une socket est identifiée par une *adresse Internet* et par un *numéro de port*.

Exécutez la commande `bind` sans argument. `socklab` va vous demander de saisir ses paramètres un par un, à savoir : l'identificateur de la socket à traiter, l'adresse Internet et le numéro de port à affecter. Pour certains arguments, `socklab` vous propose des valeurs par défaut ; c'est le cas de l'identificateur de la socket à traiter : par défaut, `socklab` vous propose l'identificateur de la dernière socket créée ou manipulée (celle qui apparait avec le signe `>` avec la commande `status`). Si `socklab` vous propose des valeurs par défaut pour des arguments, elles apparaissent entre parenthèses. Il suffit alors d'appuyer sur `Enter` pour les valider.

Dans l'exemple nous affectons à la socket d'identificateur 3 l'adresse de la station `sparc1` : remplacez-là par l'adresse ou le nom de la station sur laquelle vous travaillez. Quant au numéro de port, nous choisissons arbitrairement le numéro `5000` :

```
socklab> bind
Id. socket (4) ?: 3
Host ?: sparc1
Port ?: 5000

socklab>
```

Quand une station est connectée sur plusieurs réseaux, elle a plusieurs adresses Internet. La primitive `bind()` peut alors accepter une adresse Internet spéciale notée `INADDR_ANY`, et qui signifie "n'importe quelle adresse de la machine" (il s'agit en fait d'une adresse générique). La notation à utiliser avec la commande bind pour désigner cette adresse particulière est l'astérisque, `*`.

En ce qui concerne le port, vous pouvez laisser au système d'exploitation le choix d'un numéro de port libre (si dans l'opération précédente le port `5000` était déjà utilisé, la primitive `bind()` aurait échoué). Pour ce faire, indiquez le numéro de port `0` ; après avoir exécuté la primitive `bind()`, `socklab` vous affichera le numéro de port choisi par le système.

Pour illustrer ces deux derniers points, nommez la deuxième socket UDP :

```
socklab> bind 4 * 0
Le port 1043 a ete affecte a la socket

socklab> =
 Id      Proto       Adresse             Connexion           Type RWX ?
 ------------------------------------------------------------------------
 3       UDP         sparc1(5000)        -                   ipv4 .W.
>4       UDP         *(1043)             -                   ipv4 .W.
socklab>
```

Vous pouvez remarquer que les deux sockets possèdent maintenant une adresse.

Jusqu'ici vous n'avez utilisé que deux commandes de contrôle, `socket` et `bind`, et une commande spéciale, `status`. Pour terminer, vous allez échanger un message entre les deux sockets UDP créées.

La primitive permettant d'envoyer un message avec le protocole UDP s'appelle `sendto()`. Les paramètres de la commande `sendto` sont dans l'ordre : l'identificateur de la socket à utiliser, la machine destination, le numéro de port destination sur cette machine et le message à envoyer. Si le message est constitué de plusieurs mots, vous devez utiliser des guillemets, sinon tous les mots seront considérés comme autant de paramètres supplémentaires.

À l'aide de la commande `sendto`, envoyez un message depuis la première socket créée, Id 3, vers la deuxième, `sparc1`, port `1043` :

```
socklab> sendto 3 sparc1 1043 "Voici un message"
16 octet(s) envoye(s)

socklab>
```

Sur la deuxième socket UDP (identifiée par 4 dans l'exemple), demandez la réception de 20 octets par la commande `recvfrom`. Cette commande n'a que deux paramètres : le numéro de la socket sur laquelle le message doit être lu, et le nombre d'octets à lire :

```
socklab> recvfrom 4 20
Un message de 16 octet(s) a ete recu de sparc1 (5000).
Message=<Voici un message>

socklab>
```

Voici maintenant ce qu'il se passe lorsque `socklab` rencontre un problème en exécutant une commande. Essayez d'utiliser la commande `accept` sur la dernière socket UDP créée. Cette commande est réservée au protocole TCP pour récupérer une demande de connexion sur une socket passive. Elle n'a donc aucun sens pour le protocole UDP :

```
socklab> accept 4
accept(): Operation not supported on transport endpoint

socklab>
```

Lorsque l'appel d'une primitive aboutit à un échec, un message décrivant le nom de la primitive appelée et l'erreur constatée est toujours affiché en vidéo-inversée (ce message est accompagné d'un signal sonore).

Notez que l'utilisation de `socklab` a été illustré sur une seule machine ; mais vous pouvez évidemment utiliser le laboratoire sur deux machines différentes, connectées par réseaux, et réaliser des communications entre ces deux laboratoires par l'intermédiaire de sockets TCP ou UDP.

À tout moment vous pouvez demander la liste des commandes disponibles, tapez `help` ou `?` :

```
socklab> help
LISTE DES COMMANDES DISPONIBLES:
    s socket            r read              w write           ? help
    k socket6          v recv              d send           = status
    b bind             f recvfrom          t sendto         q quit
    l listen              x exit
    a accept
    c connect
    k close
    h shutdown
    o options

socklab>
```

Chaque commande apparait avec la lettre qui peut être employée à sa place comme abréviation minimale.

Chapitre 4

Guide de référence de `socklab`

4.1 Utilisation

Le laboratoire d'étude des sockets se lance en tapant la commande `socklab` à partir d'un shell. En précisant le paramètre `udp` ou `tcp`, vous pouvez orienter la manipulation des sockets vers le protocole UDP ou TCP respectivement.

Une fois le laboratoire lancé, vous avez accès à un interpréteur de commandes qui vous permet de créer, sélectionner, manipuler ou supprimer les sockets.

Le laboratoire gère en permanence une table dans laquelle est mémorisée la liste des sockets que vous avez créées.

La plupart des commandes demandent un ou plusieurs paramètres : vous pouvez les préciser en tant qu'arguments sur la ligne de commande ; sinon ils vous sont automatiquement demandés lors de l'exécution de la commande.

Quelques commandes possèdent aussi des options, par exemple les commandes `send` ou `recv` : ces options ne peuvent être précisées que sur la ligne de commande. Si vous les omettez, elles ne vous seront pas demandées lors de l'exécution de la commande (les options sont des mots qui commencent toujours par un tiret `-`).

Vous avez le droit d'abréger les noms des commandes, des paramètres ou des options, pourvu que l'interpréteur puisse les identifier sans ambiguïté. Par exemple, vous pouvez taper `so` à la place de la commande `socket`. Par contre vous ne pouvez pas taper `se` à la place de `send` parce que la commande `sendto` commence aussi par `se`. Cependant, à chaque commande est associée une lettre unique qui peut être utilisée à la place du nom complet.

À tout moment vous pouvez interrompre le déroulement d'une commande en appuyant sur `Ctrl-C` : vous revenez alors automatiquement à l'interpréteur de commandes du laboratoire.

Nous allons maintenant décrire séparément toutes les commandes reconnues par l'interpréteur de `socklab`. Pour chacunes d'elles nous indiquons d'abord la lettre pouvant être utilisée comme raccourci, puis ses éventuels paramètres ou options.

4.2 Commandes spéciales

Les commandes spéciales sont celles qui ne s'appliquent pas à une socket en particulier. Elles sont toujours disponibles, quel que soit le mode de fonctionnement du laboratoire.

4.2.1 `Q|X - quit|exit`

Description : ces deux commandes sont équivalentes : elles permettent de quitter le laboratoire pour revenir au shell. Une confirmation est toujours demandée (toutes les sockets créées sont supprimées lors de la terminaison de `socklab`).

4.2.2 `? - help` *`<cmd> ...`*

Description : sans paramètre, cette commande affiche la liste de toutes les commandes disponibles avec la lettre qui peut être utilisée comme raccourci. Si des noms de commandes sont passées comme paramètre, la commande affiche seulement leur syntaxe.

Paramètre :

`<cmd>` nom d'une commande dont la syntaxe doit être affichée.

4.2.3 `= - status`

Description : affichage de l'état de toutes les sockets créées. La dernière socket créée ou manipulée est indiquée par la marque `>` au début de la ligne qui la décrit. Pour chaque socket la commande affiche :

1. son identificateur ;
2. le protocole utilisé : TCP ou UDP ;
3. son adresse locale si elle en possède une (avec son numéro de port) ;
4. l'adresse de la socket distante si une connexion a été établie ;
5. le type de la socket : `ipv4` ou `ipv6` ;
6. les opérations réalisables :
   - `R` (read) lecture de données ou récupération d'une connexion ;
   - `W` (write) écriture de données ;
   - `X` (exception) lecture de données urgentes.

4.3 Commandes en mode standard

Dans ce mode, toutes les commandes correspondent à une primitive particulière de l'interface de programmation des sockets (c'est le mode de fonctionnement du laboratoire par défaut). Le nom de chaque commande est donc celui de la primitive manipulée.

À part `socket`, toutes les commandes ont pour premier paramètre l'identificateur *`<id>`* de la socket sur laquelle elles doivent s'appliquer. Pour ce paramètre vous pouvez préciser un point `.` pour indiquer que vous désirez utiliser la dernière socket créée ou manipulée.

4.3.1 S - socket [tcp|udp]

Description : création d'une socket dans le domaine Internet, de type STREAM ou DATAGRAM, selon la valeur du paramètre.

Paramètre :

`tcp` création d'une socket de type STREAM (utilisation du protocole TCP) ;

`udp` création d'une socket de type DATAGRAM (utilisation du protocole UDP).

4.3.2 B - bind *<id> <host> <port>*

Description : affectation d'une adresse à la socket `<id>`.

Paramètres :

<host> adresse Internet à affecter à la socket. Si la machine sur laquelle vous travaillez est connectée à plusieurs réseaux, elle possède plusieurs adresses. Ce paramètre vous permet alors d'en chosir une précise. Vous pouvez taper directement l'adresse (*dot notation* : `xxx.xxx.xxx.xxx`) ou vous pouvez donner un nom symbolique figurant dans le fichier `/etc/hosts` (`.` est un raccourci pour le nom officiel de la machine sur laquelle vous travaillez).

Vous pouvez spécifier l'adresse particulière `INADDR_ANY` avec l'astérisque `*`. L'adresse `INADDR_ANY` est généralement utilisée pour les sockets TCP passive, pour pouvoir recevoir des requêtes de connexion sur n'importe quelle adresse de la machine.

<port> numéro de port de la socket. Vous pouvez taper directement le numéro du port ou vous pouvez donner le nom d'un service figurant dans le fichier `/etc/services`. En fournissant la valeur `0`, le système choisit automatiquement un numéro de port disponible.

4.3.3 L - listen *<id> <nb>*

Description : mise en écoute de la socket *<id>* (socket TCP uniquement).

Paramètre :

<nb> nombre de connexions pouvant être maintenues en attendant que ces dernières soient récupérées par la commande `accept` (par défaut, `nb = 5`).

4.3.4 A - accept *<id>*

Description : attente ou récupération d'une connexion sur la socket *<id>* (socket TCP uniquement).

4.3.5 C - connect *<id> <host> <port>*

Description : connexion de la socket *<id>* sur une socket distante. Si la connexion a pu être établie, une nouvelle socket est créée.

Paramètres :

<host> nom ou adresse de la machine distante (cf. `bind`) ;

<port> numéro de port ou nom de service sur la machine distante.

4.3.6 K - close *<id>*

Description : fermeture de la socket *<id>*.

4.3.7 H - shutdown *<id>* [in|out|both]

Description : fermeture de la connexion précédemment établie sur la socket *<id>*.

Paramètre :

`in` fermeture en entrée seulement ;

`out` fermeture en sortie seulement ;

`both` fermeture dans les deux sens.

4.3.8 W - write *<id>* *<msg>*

Description : écriture d'un message dans la connexion précédemment établie sur la socket *<id>*.

Paramètre :

<msg> message à envoyer. Si vous voulez envoyer plusieurs mots, vous devez les limiter par des guillemets. Pour envoyer un nombre d'octets précis, sans tenir compte du contenu du message, utilisez la notation `#nnn` où `nnn` est le nombre d'octets à envoyer (le message est alors constitué d'astérisques `*`).

4.3.9 R - read *<id>* *<nb>*

Description : lecture d'un message dans la connexion précédemment établie sur la socket *<id>*.

Paramètre :

<nb> nombre d'octets à lire (100 par défaut).

4.3.10 D - send *<id>* *<msg>* [-oob] [-dontroute] [-loop]

Description : écriture d'un message dans la connexion précédemment établie sur la socket *<id>*. Cette commande diffère de `write` uniquement par les options qu'elle propose.

Paramètres :

<msg> message à envoyer (cf. `write`).

Options :

`-oob` option *Out Of Band*, pour envoyer le message en mode *urgent* ;

`-dontroute` option *Don't Route*, pour que le message ne puisse pas franchir un routeur ; autrement dit, pour qu'il ne puisse pas sortir des limites du réseau sur lequel la machine est connectée ;

`-loop` cette option ne figure pas dans la primitive `send()` standard. En la précisant, `socklab` envoie le message jusqu'à ce qu'une erreur intervienne ou que vous appuyez sur `Ctrl-C`.

4.3.11 `V - recv <id> <nb> [-oob] [-peek]`

Description : lecture d'un message dans la connexion précédemment établie sur la socket *<id>*. Cette commande diffère de `read` uniquement par les options qu'elle propose.

Paramètre :

<nb> nombre d'octets à lire.

Options :

`-oob` option *Out Of Band*, pour lire des données urgentes ;

`-peek` option *Peek*, pour lire les données sans les retirer du buffer de réception ; cette option permet de relire plusieurs fois les mêmes données.

4.3.12 `T - sendto <id> <host> <port> <mesg> [-oob] [-dontroute] [-loop]`

Description : envoi d'un message à une socket destination précise. La socket *<id>* est utilisée comme source du message (socket UDP uniquement).

Paramètres :

<host> adresse de la machine destination ;

<port> numéro de la socket sur la machine destination.

Options :

`-oob` option *Out Of Band* (cf. `send`) ;

`-dontroute` option *Don't Route* (cf. `send`) ;

`-loop` option *Loop* (cf. `send`).

4.3.13 `F - recvfrom <id> <nb> [-oob] [-peek]`

Description : réception sur la socket *<id>*, d'un message envoyé par une socket distante (socket UDP uniquement).

Paramètre :

<nb> nombre d'octets à lire.

Options :

`-oob` option *Out of Band* (cf. `recv`)

`-peek` option *Peek* (cf. `recv`)

4.3.14 `O - options <id> <option> ...`

Description : consultation et modification des options de la socket *<id>*. Utilisée sans paramètre, cette commande passe en revue toutes les options de la socket : elles sont affichées une par une, avec leur valeur courante. Si vous précisez une ou plusieurs options particulières par paramètre, la commande ne traite que celles-ci.

Pour ne pas changer la valeur d'une option, tapez directement sur `Enter` ; l'option suivante est alors affichée.

La majorité des options sont booléennes : vous devez répondre par `o` ou `n` pour les modifier.

L'option `asynchrone` permet d'être averti quand un message ou une demande de connexion arrive sur la socket.

L'option `nodelay` permet de rendre les primitives non bloquantes (par défaut, un `read` est bloquant quand aucun message n'est prêt).

Toutes les autres options sont celles que l'on peut consulter ou modifier par les primitives `getsockopt()` et `setsockopt()` (toutefois, toutes les options normalement accessibles ne sont pas représentées dans `socklab`).

4.4 Commandes en mode TCP

Le mode TCP est le mode obtenu en lançant le laboratoire avec le paramètre `tcp`. L'objectif de ce mode est d'étudier le protocole TCP, et non pas les sockets elles mêmes. Les commandes disponibles sont moins nombreuses mais plus simples à utiliser, car elles réalisent en général l'enchainement de plusieurs primitives manipulant les sockets.

À part `passive` et `connect`, toutes les commandes ont pour premier paramètre l'identificateur `<id>` de la socket sur laquelle elles doivent s'appliquer. Pour ce paramètre vous pouvez préciser un point `.` pour indiquer que vous désirez utiliser la dernière socket créée ou manipulée.

4.4.1 `P - passive`

Description : création d'une socket passive, prête à recevoir des demandes de connexion. L'adresse `INADDR_ANY` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket tcp`, `bind . * 0` et `listen . 5` en mode standard).

4.4.2 `A - accept` *<id>*

Description : identique à la commande `accept` en mode standard.

4.4.3 `C - connect` *<host> <port>*

Description : création d'une socket active et puis tentative de connexion sur une socket distante. L'adresse `INADDR_ANY` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket tcp`, `bind . * 0`, `connect .` *<host> <port>* en mode standard).

Paramètre :

<host> nom ou adresse de la machine distante ;

<port> numéro de port de la socket sur la machine distante.

4.4.4 K - close *<id>*

Description : identique à `close` en mode standard.

4.4.5 H - shutdown *<id>* [in|out|both]

Description : identique à `shutdown` en mode standard.

4.4.6 W - write *<id>* *<msg>*

Description : identique à `write` en mode standard.

4.4.7 R - read *<id>* *<nb>*

Description : identique à `read` en mode standard.

4.4.8 D - usend <id> <msg>

Description : écriture d'un message urgent dans la connexion précédemment établie sur la socket *<id>* (équivalent à `send` *<id>* *<msg>* `-oob` en mode standard).

Paramètre :

<msg> message à envoyer (cf. `write`)

4.4.9 V - urecv *<id>* *<nb>*

Description : lecture d'un message urgent dans la connexion précédemment établie sur la socket *<id>* (équivalent à `recv . <nb> -oob` en mode standard).‘

Paramètre :

<nb> nombre d'octets à lire.

4.4.10 O - options *<id>* *<option>* ...

Description : identique à `options` en mode standard.

4.5 Commandes en mode UDP

Le mode UDP est le mode obtenu en lançant le laboratoire avec le paramètre `udp`. L'objectif de ce mode est d'étudier le protocole UDP, et non pas les sockets elles mêmes.

À part socket, toutes les commandes ont pour premier paramètre l'identificateur *<id>* de la socket sur laquelle elles doivent s'appliquer. Pour ce paramètre vous pouvez préciser un point . pour indiquer que vous désirez utiliser la dernière socket créée ou manipulée.

4.5.1 S - socket

Description : création d'une socket UDP. L'adresse `INADDR_ANY` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket udp` et `bind . * 0` en mode standard).

4.5.2 K - close *<id>*

Description : identique à `close` en mode standard.

4.5.3 T - sendto *<id> <host> <port> <msg>*

Description : identique à `sendto` en mode standard, les options en moins.

4.5.4 F - recvfrom *<id> <nb>*

Description : identique à `recvfrom` en mode standard, les options en moins.

4.5.5 O - options *<id> <options> ...*

Description : identique à `options` en mode standard.

4.6 Cas particulier des sockets IPV6

Il est possible de manipuler des sockets `AF_INET6` associées donc à des adresses IPV6.

4.6.1 Mode `standard`

En mode `standard` il faut alors utiliser la commande `socket6`. La manipulation de cette socket se fait ensuite avec les mêmes commandes que les sockets IPV4.

`k - socket6 [tcp|udp]`

Description : création d'une socket dans le domaine Internet `AF_INET6`, de type STREAM ou DATAGRAM, selon la valeur du paramètre.

Paramètre :

`tcp` création d'une socket de type STREAM (utilisation du protocole TCP) ;

`udp` création d'une socket de type DATAGRAM (utilisation du protocole UDP).

4.6.2 Mode `udp`

En mode `udp` il faut alors utiliser la commande `socket6`.

`N - socket6`

Description : création d'une socket IPV6 UDP. L'adresse `::` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket6 udp` et `bind . * 0` en mode standard).

4.6.3 Mode `tcp`

En mode `tcp`, on peut utiliser les commandes `passive6` et `connect6`. Les autres commandes sont les mêmes qu'en IPV4.

`V - passive6`

Description : création d'une socket passive IPV6, prête à recevoir des demandes de connexion. L'adresse `::` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket6 tcp`, `bind . * 0` et `listen . 5` en mode standard).

`E - connect6` *<host> <port>*

Description : création d'une socket active IPV6 et puis tentative de connexion sur une socket distante. L'adresse `::` lui est automatiquement affectée, et un numéro port libre est choisi par le système (équivalent à `socket6 tcp`, `bind . * 0`, `connect .` *<host> <port>* en mode standard).

Paramètre :

<host> nom ou adresse de la machine distante ;

<port> numéro de port de la socket sur la machine distante.