Networking – part II

Luis D. Pedrosa & Jean-Cédric Chappelier 2017

Cette semaine vous allez implémenter les fonctionnalités nécessaires à la gestion de plusieurs serveurs, mais dans un scénario d'utilisation encore simple dans lequel tous les serveurs stockent tous toutes les données ($W{=}N{>}1$, $R{=}1$).

Vu qu'il y a plusieurs serveurs, nous ne pouvons plus nous contenter d'une seule adresse IP (codée « en dur » dans le code) ; les adresses des différents serveurs devront être lues depuis un fichier local nommé `servers.txt` (utiliser `PPS_SERVERS_LIST_FILENAME` de `config.h`). Ce fichier contient l'adresse et le port de chaque serveur, un par ligne (UNIX line termination), l'adresse et le port étant séparés par une espace. Par exemple :

```
1 127.0.0.1 1234\n
2 127.0.0.1 1235\n
3 127.0.0.1 1236\n
```

(où `\n` ne doit être écrit explicitement, mais symbolise un retour à la ligne).

I. `node_list_t`

La première chose à faire est de définir le type `node_list_t` (voir le fichier `node_list.h`), et ses fonctions associées, servant à représenter une liste de serveurs (puis une liste de nœuds dans la version finale du projet).

Complétez pour cela le fichier `node_list.h`, puis, dans un fichier `node_list.c`, définissez les fonctions (voir le fichier `node_list.h` pour une description plus détaillée) :

- `node_list_new()` : qui crée une nouvelle liste de serveurs vide (ou retourne `NULL` en cas d'erreur) ;
- `get_nodes()` : qui crée une nouvelle liste de serveurs et la remplit à partir du fichier `PPS_SERVERS_LIST_FILENAME` (défini dans `config.h`), de format décrit plus haut (notez qu'une adresse ip est d'une chaîne d'au plus 15 caractères (p.ex. 123.122.121.229) et qu'un port est un entier) ; elle retourne `NULL` en cas d'erreur (quelque soit la source de l'erreur) ; un numéro de port négatif ou nul, ou supérieur strictement à 65535 sera considéré comme une erreur ;

- `node_list_free()` : qui vide une liste de serveurs, tout en les fermant proprement au préalable ;
- `node_list_add()` : ajoute un serveur/nœud *déjà initialisé* à la liste de serveurs ; cette fonction n'est pas nécessaire avant la semaine 11 (anneaux des nœuds) et il n'est donc pas strictement nécessaire de la faire cette semaine ci, mais cela peut s'avérer plus efficace de le faire maintenant vu que vous êtes en ce moment sur ce sujet (libre à vous de vous organiser) ; cette fonction doit retourner `ERR_BAD_PARAMETER` en cas de problème avec ses paramètres, `ERR_NOMEM` en cas de problème d'allocation mémoire (et bien sûr `ERR_NONE` s'il n'y a pas d'erreur).

II. Modification du code client existant

Il vous faut maintenant adapter et étendre le code des clients écrit jusqu'ici. Commencez par modifier le type `client_t` pour traiter une liste de serveurs ; puis adaptez le code de `client.c` en conséquence.

Ensuite (et surtout), il faut modifier les protocoles de communication du client (`network_get()` et `network_put()` du fichier `network.c`) de sorte à prendre en compte tous les serveurs et non plus un seul.

La commande « *put* » contacte chacun des serveurs l'un après l'autre (dans l'ordre donné par le fichier `PPS_SERVERS_LIST_FILENAME`), en attendant un acquittement (d'écriture) ou un « timeout » avant de passer au suivant.
Si n'importe lequel des serveurs échoue dans l'écriture, la commande « *put* » devra échouer (et donc afficher **"FAIL\n"** à la fin), **MAIS** elle doit néanmoins chercher à écrire sur chacun des serveurs, c.-à-d. continuer à (essayer) d'écrire sur les autres serveurs (W=N ici) avant d'afficher, au final, son échec.

De son coté, la commande « *get* » doit contacter les serveurs les uns après les autres(dans l'ordre donné par le fichier `PPS_SERVERS_LIST_FILENAME`), mais s'arrêter dès le *premier* succès de lecture (R=1 ici).

Si l'une quelconque de ces deux commandes n'arrive pas à lire le fichier `PPS_SERVERS_LIST_FILENAME`, elle doit immédiatement s'arrêter en affichant **"FAIL\n"**.

III. Modification de `pps-launch-server`

La dernière modification à faire consiste à adapter `pps-launch-server` afin de pouvoir spécifier l'adresse IP et le port sur lequel lancer le serveur. Pour cela, on ajoutera tout au début de ce programme la lecture de ces deux informations sur l'entrée standard, au format précisé en introduction (l'adresse et le port sont séparés par une espace).

Pour montrer que l'on attend quelque chose, le programme commencera par écrire "IP port?" (avec une espace en fin de chaîne mais sans retour à la ligne).

Et n'oubliez pas, bien sûr, de modifier l'adresse et le port utilisés en fonction de ceux lus sur l'entrée standard.

V. Conseil et rendu

Concernant le rendu, comme les semaines précédentes, le travail de cette semaine ne sera pas évalué séparément, mais devra faire partie du premier rendu intermédiaire du projet (délai : le dimanche 22 avril 23:59). Néanmoins, nous vous conseillons à nouveau de travailler régulièrement et faire systématiquement des tags hebdomadaires (si votre travail correspondant est opérationnel) afin de profiter de nos comptes-rendus de tests. Le tag correspondant à cette semaine est `week06`.

Pour cette semaine, il faudrait mettre à jour tous les fichiers nécessaire (sans oublier le `Makefile`) et ajouter le fichier `node_list.c` au répertoire `done/` de votre dépôt GitHub (de groupe).