Systèmes et réseaux
Projet Réseaux de Kahn

Axel Davy
École normale supérieure

Baptiste Lefebvre
École normale supérieure

26 mai 2013

1 Installation

Une fois l'archive `davy-lefebvre.tgz` récupérée et les fichiers extraits placez-vous dans le répertoire `davy-lefebvre`. Dans ce répertoire la commande :

- `make` : compile notre programme

Pour la désinstallation la commande :

- `make clean` : efface tous les fichier qu'une des commandes précédentes a pu engendrer et ne laisse dans le répertoire que les fichiers sources

Si vous n'avez pas réussi à récupérer l'archive vous pouvez également récupérer le code source à l'aide de la commande

- `git clone https://github.com/KahnProcessNetworks/KahnProcessNetworks`

en vérifiant à bien avoir installé au préalable le système de contrôle de version `git`.

2 Commentaires

2.1 Implémentation Pipe

Pas de commentaires particuliers

2.2 Implémentation Network

La toute première implémentation de Network que l'on a réalisé était assez peu performante : On avait considéré que les put et les get étaient des requêtes à un serveur représentant le channel. Le problème de performance était au niveau de l'implémentation : chaque `put` et `get` utilisait `connect` pour se connecter au serveur, puis une fois le put et get terminé, on fermais le socket. Des problèmes sont apparus dûs au fait que l'on réalisait un grand nombre de connections par secondes. Les problèmes ont été résolus en paramétrant les options des sockets, mais l'implémentation était vraiment lente.

Finalement au lieu d'améliorer cette implémentation en gardant les connections actives, et donc en évitant de refaire des connections à chaque put et get, nous sommes partis sur un tout autre concept permettant d'utiliser vraiment plusieurs machines du réseau, au lieu d'utiliser le réseau local de notre ordinateur.

Nous avons cherché à faire une implémentation vraiment complète à ce niveau là : au départ plusieurs machines sont en attente d'ordres et uen machine commence le programme. Chaque `doco` fait appel à des machines en attentes pour réaliser les taches, puis met la machine ayant fait le `doco` en attente elle-même. Les machines ayant reçu un ordre font un fork pour avoir une instance du programme qui attend des ordres et une autre qui exécute, ce qui fait que toutes les machines sont susceptibles à tout moment de recevoir des ordres.

Nous avons rencontré un certain nombre de problèmes dont le plus gros étant Marshal qui ne supportait pas les types abstraits (que l'on utilisait pour définir les connections ouvertes).

2.3 Implémentation Sequential

Notre première approche a été de considerer qu'un processus pouvait soit terminer, soit rendre la main sans avoir terminé. Dans cette implémentation, les processus avaient le type `'a process = ('a -> unit) -> Status` avec `Status` indiquant si le processus avait terminé, ou s'il fallait le rappeler. `doco` devait donc appeler un à un les processus et en fonction du résultat renvoyé, savait s'il fallait rappeler le processus plus tard ou pas. Le problème de cette implémentation se situait au niveau de `bind` : `bind e e'` devait au premier appel exécuter le processus `e` et conserver son résultat, puis au second appel exécuter `e'` avec le résultat conservé, mais surtout aux appels suivant conserver les modifications appliquées à `e'` (qui à chaque appel pouvait évoluer). Par exemple si l'on execute le processus `integers` de l'exemple donné, après n appels au processus, on se retrouve à n appels de `bind` imbriqués. La complexité de l'implémentation était en $O(n^2)$, ce qui n'était pas satisfaisant.

Pour améliorer l'implémentation, le retour des fonctions a été remplacé par une exception à lever si le processus devait être rappelé. Le programme s'exécutait plus rapidement (un facteur 3 environ), mais la complexité était toujours en $O(n^2)$.

Finalement cette dernière implémentation a été améliorée pour renvoyer lors de l'exception la fonction permettant d'exécuter la suite du processus. `doco` faisait ensuite appel à cette fonction et non pas au processus initial qui avait été modifié. Il n'y a donc plus le problème d'appels imbriqués et la complexité est linéaire.

Lors de nos tests, la vitesse de cette implémentation était très proche de celle de pipe.