Systèmes et réseaux
Projet Réseaux de Kahn

Axel Davy
École normale supérieure

Baptiste Lefebvre
École normale supérieure

26 mai 2013

1 Choix techniques

Aucun.

2 Difficultés rencontrées

Aucune.

3 Éléments non réalisés

Aucun.

4 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` ou `make thread` : compile notre programme avec l'implémentation naïve qui utilise la bibliothèque de threads d'OCaml
- `make pipe` : compile notre programme avec l'implémentation reposant sur l'utilisation de processus Unix communiquant par des tubes
- `make network` : compile notre programme avec l'implémentation s'exécutant à travers le réseau
- `make sequential` : compile notre programme avec l'implémentation séquentielle où le parallélisme est simulé par 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`.

5 Commentaires

5.1 Implémentation Pipe

vide

5.2 Implémentation Network

vide

5.3 implémentation Sequential

Notre première approche a été de considerer qu'un processus pouvait soit terminer, soit rendre la main sans avoir terminer. 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 devait au premier appel exécuter le processus e et conserver son résultat, puis au second appel exécuter e' avec le résultat conservé, mais surtout, il devait aux appels suivant conserver les modifications appliquées à e' (qui à chaque appel pouvait évoluer). Par exemple si l'on execute le process integers de l'exemple donné, après 2n appels au process, 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 programme devait être rappelé. Le programme s'exécutait plus rapidement, 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 pas la suite 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.