Réseaux de Kahn

Thomas Laure

May 2021

Introduction

Projet Réseaux de Kahn réalisé dans le cadre de cours de L3 de Systèmes d'Exploitation au 2nd semestre 2020-21 à l'ENS. Voir le fichier README.md pour la compilation et l'exécution.

Implémentation par des processus unix

Processus

On utilise le module Unix d'Ocaml. Les processus ont le type $unit- >' a$.

run, *return* et *bind* ont un fonctionnement évident identique à celui de l'implémentation par des threads.

Les channels sont implémentées par les pipes Unix, les in-port et out-port sont les in-channel et out-channel crées à partir de entrées et sortie de ces pipes. La fonction new-channel crée ces in-channel et out-channel une seule fois.

put envoie la valeur dans l'out-channel en l'encodant sous forme de bytes grâce au module Marshal.

get lit et renvoie la valeur au sommet de l'in-channel et la décode par Marshal, sauf si la pipe est vide, auquel cas get s'appelle récursivement. Les processus tournant en parallèle, un autre processus doit finir par appeler put de l'autre côté de la pipe.

Parallélisme

Enfin, *doco* utilise la fonction $fork$ d'Unix pour créer un nouveau processus pour tous ses fils. Le père crée tous les fils récursivement, attend leur mort puis continue son exécution, tandis que les fils s'exécutent en parallèle puis terminent avec le code de retour 0.

Implémentation séquentielle

Processus

Les processus ont le type $('a -> unit) -> unit$. Cela représente un calcul qui va déterminer une valeur de type $'a$, l'envoyer comme argument à la fonction qu'il prend en argument, puis terminer.

run crée une référence de type $'a option$, exécute e et donne comme argument la continuation qui prend la valeur de retour de e et la place dans la référence. Ainsi, à la fin de cette fonction, res contient $Some\ v$ où v est la valeur de retour de e, on la renvoie.

return envoie v à sa continuation.

bind exécute son premier argument avec comme continuation la fonction qui prend ce résultat et exécute le b process avec la continuation du bind.

Les channels sont des piles.

Simulation du parallélisme

Le module contient une exception Reschedule d'argument $unit -> unit$, qui représente la continuation du calcul à effectuer. Elle est levée dans deux cas.

D'abord, après avoir effectué l'opération put, on considère que le processus a fini une étape de calcul et doit être mis en attente (ceci pour éviter qu'un processus boucle pour toujours sans laisser la place aux autres et en remplissant indéfiniment un canal sans personne pour le vider. Ainsi, on lève alors l'exception avec en argument la continuation du processus.

Lorsque l'on tente de faire get et que le canal est vide, on considère que l'on doit attendre qu'un autre processus place une valeur dans le canal. Ainsi, on lève l'exception avec en argument le processus get et sa continuation.

Ces exceptions sont rattrapées dans doco dont le fonctionnement est le suivant :

docaux prend en argument une liste de processus, exécute chacun d'eux successivement jusqu'à la fin ou jusqu'à arriver à l'exception reschedule. Dans le premier cas, on ne traite plus le processus, dans le second, on crée le nouveau processus correspondant à la continuation renvoyée par l'exception. La fonction renvoie donc une liste de processus plus petite (au sens large) que la précédente, où chaque processus s'est exécutée jusqu'à ne plus pouvoir le faire.

doco fait des appels successifs à docaux jusqu'à ce que tous les processus soient arrivés à leur terme.