Systèmes et Réseaux : Implémentation des réseaux de Kahn

Joseph Marotte
Lucas Willems

15 mai 2016

Le but de ce projet est de manipuler les réseaux de Kahn. La première étape consiste à implémenter ces réseaux et la seconde étape à les utiliser dans une application concrète.

1 Implémentations des réseaux de Kahn

Comme suggéré par le sujet, nous avons fait 3 implémentations distinctes des réseaux de Kahn :
— Une implémentation reposant sur l'utilisation de processus Unix communiquant par des tubes (avec fork et thread)
— Une implémentation s'exécutant à travers le réseau (avec fork et thread)
— Une implémentation séquentielle
Chaque implémentation a pu poser quelques difficultés et a nécessité des choix techniques.

Implémentation avec des tubes

L'implémentation ici dépends des types choisis pour in_port et out_port puisque cela influencera le choix pour les fonctions new_channel, put et get.

Nous avons choisi d'utiliser des in_channel et des out_channel pour in_port et out_port ainsi que les fonctions associés dans Marshal pour le put et le get : to_channel et from_channel. Pour l'initialisation des ports, il suffit de faire un appel à Unix.pipe puis de récupérer les channels associés.

Nous aurions pu utiliser les types file_descriptor pour les in_port et out_port ainsi que les fonctions associées dans Marshal, mais nous avons trouvé que l'utilisation de in_channel et out_channel était plus simple, la principale difficulté est alors de ne pas oublier de flush les données après le put (comme c'est indiqué dans la documentation de ocaml en fait), ce que nous avions oublié de faire au début.

Implémentation par le réseau

L'implémentation par le réseau est proche de celle par tubes. On utilise les mêmes types, la seule différence se fait lors de l'initialisation d'un nouveau channel (fonction new_channel). Pour cela on utilise les sockets de manière conventionnelle, avec l'initialisation du client et du serveur.

Implémentation séquentielle

L'implémentation du réseau de Kahn séquentiel nous a demandé de bien comprendre le type de la fonction `process` et comment l'utiliser.

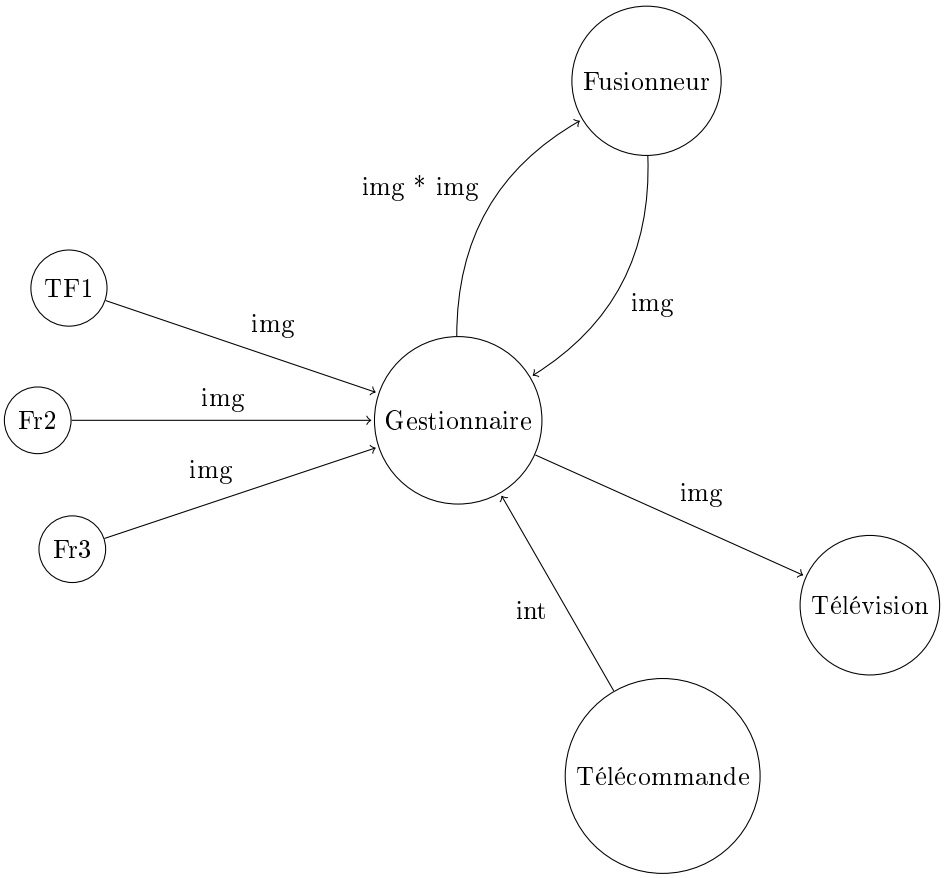
La première difficulté que nous avons rencontré était au niveau de l'implémentation de la fonction `doco`. Comment faire à tour de rôle un pas de calcul pour chacun des programmes ? La solution a consisté à introduire une file `tasks`. Ainsi, à chaque fois qu'un programme est appelé, il effectue une étape de calcul et met la suite dans la file. Une boucle `while` exécute les tâches dans la file et la vide progressivement.

La seconde difficulté a été d'éviter les `out_of_memory` apparaissant lors de l'exécution de notre Picture-in-Picture. Ceci étaient dûs au fait que le nombre d'étape de calculs pour calculer la fonction `manager` (qui enlève 1 image dans chacun des fluxs) correspondait à 10 fois le nombre d'étape de calculs pour calculer la fonction `in_flux` (qui ajoute 1 image dans chacun des fluxs). Ainsi, le nombre d'image en attente devenait vite très grand. Pour ce problème, la solution a été de reporter les `put` dans un canal lorsque celui-ci n'était pas vide.

2 Mise en application avec le Picture-in-Picture

Une fois les implémentations des réseaux de Kahn réalisées, nous avons décidé d'implémenter un système de Picture-in-Picture pour manipuler les réseaux. Ce système consiste à incruster une image dans une image et est beaucoup utilisé dans les télévisions pour visionner deux chaines en même temps. Les réseaux de Kahn permettent ici de séparer et pipeliner les différentes tâches permettant l'incrustation (réception des images, fusion, envoie...).

Pour être clair, voici le réseau de notre Picture-in-Picture qui se veut être le plus fidèle au fonctionnement de la télévision :



Pour réaliser le Picture-in-Picture, nous avons d'abord eu besoin d'implémenter deux modules :

— un module `Image` permettant de facilement manipuler les images et notemment de fusionner deux images prises en paramètre grâce à une fonction `merge`

— un module `Video` permettant de facilement manipuler les vidéos notemment grâce à une fonction `new_frame` pour obtenir la frame d'après dans la vidéo

Une fois ces deux modules implémentés, nous avons pu implémenter le réseau que nous venons de voir en réutilisant nos implémentations des réseaux de Kahn. Voici quelques détails et les difficultés rencontrées pour chaque process :

— Les process TF1, FR2, FR3 ont la même implémentation. Chaque process décompose la vidéo dont le chemin lui est donné en une suite d'image (pour cela il utilise ffmpeg), puis lit les images correspondantes à la vidéo et les transmet au gestionnaire. Lorsqu'il arrive à la dernière image il reprend à la première pour donner l'illusion de la télévision qui a un flux permanent. L'implémentation de ces process est simple.

— Le process télévision reçoit l'image à afficher de la part du gestionnaire et l'affiche, son implémentation est très simple et ne pose pas de problème.

— Le process fusionneur reçoit un couple d'images à fusionner de la part de gestionnaire. Il réduit la taille de l'une des deux images (selon un paramètre donné à la création de ce process) et l'incruste sur l'autre image (à une position donnée en paramètre également). L'implémentation ne pose pas de problème non plus.

— Le process gestionnaire est l'intermédiaire entre les différents process. Il va recevoir les images des différentes chaînes télévisées, et selon les chaînes séléctionnées via la télécommande va : réaliser une fusion (en envoyant les deux images à fusionneur) ou envoyer directement l'image souhaité à télévision. L'implémentation est un peu longue car elle fait le lien entre les autres process.

— Le process télécommande permet de changer de chaîne et de gérer quelle est la chaîne principale (en plein écran) et la chaîne incrustée, qui est optionnel (dans un coin de l'écran). C'est l'implémentation de ce process qui nous a posé le plus de difficultés. Le fonctionnement est le suivant :

— la télécommande envoie au gestionnaire un entier i compris entre 1 et n (où n est le nombre de chaînes) qui correspond à la chaîne souhaitée :
— Si i correspond au numéro de la chaîne principale, alors seulement la chaîne principale i est affichée.
— Sinon si i correspond au numéro de la chaîne incrustée, alors elle devient la chaîne principale et il n'y a plus de chaîne incrustée.
— Sinon si i est un numéro de chaîne valide, la chaîne numéro i devient la chaîne incrustée.
— Sinon, la commande est ignorée.

Pour implémenter ce process, nous avons essayé plusieurs idées :

1. Notre première idée était la suivante : à l'aide de la fonction Unix.select, nous réalisons des entrées directement pendant un certain laps de temps (idéalement 1/24 si l'on veut du 24 images par secondes), si rien n'a été entré par l'utilisateur, alors la télécommande envoie le signal -1 au gestionnaire qui lui indique de ne pas changer de chaîne. Le problème de cette implémentation est que le programme va être exécuté de manière plus ou moins rapide selon le matériel sur lequel il est utilisé, s'il est exécuté trop lentement (par exemple une image par seconde) alors le gestionnaire ne lira qu'une valeur sur le canal (télécommande, gestionnaire) par seconde pendant que 23 autres attendront (car la télécommande met 1 valeur toutes les 1/24 secondes) et alors le changement de chaîne ne se fait pas de manière immédiate ce qui est gênant, pour régler ce problème il faut choisir un laps de temps plus important pour le select mais on peut se retrouver avec le problème inverse ou la télécommande va ralentir le flux général puisque le gestionnaire attends de savoir s'il doit changer de chaîne ou non.
2. Notre seconde idée repose sur les verrous (lock) : on utilise une simple lecture (read_int) pour récupérer la chaîne que l'utilisateur souhaite, cela n'est censé bloqué que le processus de la télécommande puisque l'exécution se fait en parallèle. Lorsqu'une chaîne est rentré, la télécommande va libérer le verrou qui permet au gestionnaire de lire la chaîne. Le gestionnaire lui, essaye de lire le verrou avec un (try_lock) ce qui lui permet de ne pas être bloqué si le verrou n'est pas libre (donc si il ne doit pas changer de chaîne). On utilise également un deuxième verrou pour n'avoir qu'un seul élément dans le canal (télécommande, gestionnaire) afin de ne pas avoir d'empilement de unlock qui pourrait être gênant également (ex : on rentre la chaîne 2 le temps que le gestionnaire prenne le verrou, la chaîne 3 est rentrée et libère le verrou également (qui est déjà libre), la télécommande prend le verrou et fait afficher 2, la prochaine fois que l'utilisateur change de chaîne 1, la chaîne 2 sera affichée et on aura un décalage). Le problème de cet implémentation est qu'elle ne marche pas avec toutes les implémentations :
   — Si l'implémentation utilise des threads alors cela fonctionne.
   — Si l'implémentation utilise des forks alors cela ne fonctionne pas car la mémoire n'est pas partagée : les différents process ont des verrous différents donc le gestionnaire ne recevra jamais l'indication de la télécommande pour changer de chaîne. Il s'agit de trouver un moyen de simuler un système de lock (avec des fichiers par exemple) pour les forks mais nous ne l'avons pas implémenter.
   — Si l'implémentation est séquentiel et simule le parallélisme alors cela marche encore moins (on est bloquant sur le read_int, et les verrous ne sont jamais dévérouillés). La télécommande ne marche pas du tout ; pire, elle empêche la simulation du parallélisme.
3. Notre troisième et dernière idée utilise les fonctions `key_pressed` et `read_key` de la bibliothèque `Graphics`. Elles permettent de savoir si une touche a été pressée (`key_pressed`) et dans ce cas, de la récupérer (`read_key`). Cette nouvelle manière de faire n'est pas bloquante et permet de résoudre tous les problèmes précédents facilement. Toutefois, elle ne fonctionne pas dans les versions avec `fork` car la fenêtre Graphics ne peut être utilisée que par un programme à la fois.