Travail pratique #1 - IFT-2245

Paul Chaffanet et Samuel Guigui

September 12, 2019

1 Bref aperçu du programme

Le programme consiste à récupérer une ligne entrée par l'utilisateur puis à l'exécuter comme le ferait un shell.

On effectue pour cela un prétraitement de cette ligne en la décomposant en une liste composée de sous-listes. Cette liste contient donc toutes les commandes qui sont pipées. Chaque sous-liste représente alors une commande ne contenant aucun pipe, mais éventuellement des redirections. On effectue également les expansions d'arguments en substituant la chaîne contenant le/les caractère/s "*" par les fichiers ou répertoires correspondants durant cette phase de prétraitement.

Après avoir effectué ce prétraitement, la liste de commandes est passée en argument à la méthode cmd_list(cmd_tokens, inp, out). Le cas de base de cette méthode est lorsqu'il n'existe qu'une seule sous-liste à traiter, cela signifie qu'il n'existe pas de pipe à créer. On ne doit créer qu'un seul processus. Il peut également exister des redirections qui sont traitées par la méthode redirect(cmd). Puis on exécute la commande. S'il existe des pipes, alors le processus se charge de créer les pipes et de forker les processus enfants nécessaires à l'exécution de commandes pipées par des appel récursifs à exec_cmd, en modifiant les arguments inp et out lors de l'appel afin de garder trace des files descriptor. Le processus qui était chargé de créer les pipes se chargera également de l'exécution de la dernière commande de liste.

2 Démarche dans la réalisation du programme

Dans un premier temps, nous avons commencer par apprivoiser la puissance du shell POSIX afin de bien comprendre le programme que l'on nous demandait de programmer. Même si on avait déjà été amené auparavant à utiliser des commandes du shell relativement simples, nous n'avions alors pas connaissance des redirections et des pipes dans le shell.

Comprendre toutes les possibilités offertes par le shell a pris un certain temps afin de pouvoir les assimilées. Dans un second temps, nous avons ensuite essayé de comprendre la relation entre shell et processus et nous avons compris que la plupart des commandes shell sont en réalité des petits programmes qui prennent

éventuellement des arguments et qui doivent s'exécuter afin de retourner un résultat. Afin de permettre l'exécution de ces petits programmes, il était donc nécessaire de créer un processus par commande à exécuter. Python est un langage que nous avions déjà utilisé dans le passé, mais nous ne connaissions pas encore le module os de Python, ou du moins l'utilisions tout juste des simples ouvertures et fermetures de fichiers.

On a donc commencé par permettre l'exécution d'une simple commande (type "cat Makefile"). La création d'un seul processus n'est pas très complexe et nous n'avons pas rencontré de difficultés particulières à ce moment là. Cependant, nous avont été surpris du fait qu'il nous était désormais possible d'exécuter notre programme à partir de notre programme.

Nous nous sommes ensuite intéressés à la problématique de l'implantation des pipes (nous n'avons pas suivi l'ordre suggéré par la donnée du travail pratique). Nous avons compris qu'il était possible de rediriger les entrées et sorties des processus à l'aide de la manipulation des files descriptors et que le pipe permettait de retourne une paire de file descriptor utile pour permettre une communication inter-processus. Nous avons donc commencé par créer un pipe de communication entre un processus parent et enfant. Mais la partie la plus ardue fut lorsque que nous avons voulu permettre le chaînage de processus par pipes. Dans ce cas là, nous avons dû repenser notre façon de voir la communication inter-processus et nous avons donc été convaincu de la nécessité de créer un processus enfant qui sera lui même-chargé de la création de pipes. Nous avons rencontrer beaucoup de difficultés afin de bien rediriger les canaux d'entrée et de sortie, mais à force de tests de notre programme, nous avons réussi à établir une communication inter-processus robuste.

Après avoir réussi la communication inter-processus via pipes, les redirections nous ont posé beaucoup moins de problèmes à être implantées. L'implantation des pipes nous avait permis de bien comprendre l'utilisation des files descriptors.

Après raffinement du code, nous avons compris qu'il était possible de tout intégrer dans une seule méthode exec_cmd qui serait récursive en cas de présence de pipes. Cela nous paraissait être une façon de voir les choses plus intéressantes pour implanter de nouvelles fonctions pour le shell, comme l'utilisation du parenthèsage par exemple.

L'expansion d'arguments n'a été ajouté qu'à la fin. Nous n'avions pas exploré cette question en détail, et nous l'avons implanté à la toute fin dans la phase de prétaitement.

3 Limites du programme et possibilités d'améliorations

Nous entrevoyons plusieurs possibilités d'amélioration pour notre programme, qui cependant n'étaient pas spécifiées dans la donnée du travail, notamment:

1. Régler le problème des flèches directionnelles qui ne permettent pas de déplacer le curseur.

2. Permettre de retrouver les commandes utilisées précedemment en utilisant la flèche directionnelle du haut.

3. Intégration complète de toutes les expansions possibles, telles que les expansions arithmétiques $((expression)) ou les expansions avec accolades comme {A,B}, avoir la possibilité d'utiliser les classes de caractères comme [[:upper:]] ou [[:digit:]].

4. Intégration des redirections doubles >>et <<ou encore les redirections 2>&1 , 3>&1, 3>&2 etc.

5. Améliorer la récupération des erreurs avec éventuellement des messages plus précis.

6. De manière plus générale, implanter un interpréteur fonctionnel sur le modèle de la spécification posix.