Compte Rendu TP 4:
Mini Shell

Hussein Elfakharany
Fabien Da Costa Barros

8 Mars 2021
Université Grenoble Alpes

Le but du TP 4 est de réaliser un programme mini-shell qui exécute des commandes UNIX. Un shell consiste à prendre une ligne de commande en entrée, analyser la commande, vérifier si la dernière est une commande de fin, sinon exécuter la commande en prenant compte des redirections et des tubes. Quant à ce TP, il fallait implémenter l'exécution des commandes avec et sans tubes, la redirection des entrées/sorties des commandes vers des fichiers textes, les commandes en arrière-plan ainsi que la gestion des erreurs et des zombies. Pour cela on a créé des fonctions dans le module *cmdexecutions* qui implémentent les fonctionnalités demandées. En ce qui concerne l'analyse syntaxique et lexicale des entrées, des fichiers sources nous ont été fournis.

Le fichier *shell.c* a été modifié pour l'implémentation de la commande *quit* et contient le squelette principal du shell.

Le module *cmdexecutions* contient les fonctions qui exécutent les commandes (simples, avec un seul tube, et avec plusieurs tubes), une fonction qui redirige les entrées/sorties vers des fichiers et un handler qu'on utilise pour intercepter le signal SIGCHLD utilisé pour prendre en compte la mort récente des processus fils.

La fonction *executePipes* prend en argument un pointeur vers la structure cmdline qui contient les données de la ligne de commandes. La fonction est appelée si un ou plusieurs tube sont détectés et elle exécute les différentes commandes en respectant leurs ordres. On a choisi de fixer un nombre de commandes maximal NMAX=10 pour la création d'un tableau statique de taille NMAX-1 qui va contenir nos tubes (on n'a pas jugé utile de créer un tableau dynamique). Chaque tube est représenté par une liste de deux entiers qui correspondent à l'entrée et à la sortie du tube respectivement. Ensuite, grâce à une boucle, on itère dans les différentes commandes et on fork. Dans le cas du fils, on applique les redirections nécessaires et on exécute les commandes à l'aide de la fonction Execvp. En revanche, le père s'occupe de fermer les entrées/sorties inutiles des tubes et à reconnaître la mort des fils avec un wait.

La fonction *executeCmd* éxecute les commandes simples sans tubes et appelle la fonction *redirect* dans le cas où la structure *cmdline* indique que l'entrée ou la sortie sont différentes que l'entrée/sortie standard.

La fonction *redirect* utilise la fonction *dup2* pour rediriger l'entrée ou la sortie si nécessaire vers un fichier texte. Dans le cas d'une écriture, le fichier est créé s'il n'existe pas. Dans le cas d'une erreur, un message est affiché, par exemple dans le cas où les permissions sont manquantes ou si la fonction open a échoué pour une quelconque autre raison.

La fonction *executePipe* est une version simplifiée de la fonction *executePipes* qui était appelée pendant les premières étapes si la ligne de commande contenait seulement un seul tube.

A propos de la gestion du background, on a modifié la structure cmdline pour qu'elle contiennent un flag indiquant si la commande doit être exécutée en background. On a assigné un handler permettant de récupérer tous les fils terminés au début du main. Cela nous permet de ne pas attendre la terminaison des fils. Pour éviter que toutes les commandes tournent en background on ré-assigne le handler par défaut dans le cas où le flag vaut 0 et on n'oublie pas de le remettre à la fin de l'exécution de la ligne de commande.

Nous avons testé le shell avec plusieurs types de test combinant différents aspects ou en les testant séparément comme les pipes, le background, les commandes simples et les options. Les tests se trouvent dans le fichier test et test l'aspect décrit par le nom de fichier.

Dans la gestion des commandes du type : grep k | sort | tail -1 | cat < abc.txt > resabc.txt où la commande comporte plusieurs pipes et une redirection en entrée et en sortie, le shell Linux ne termine pas. Dans notre shell, ce cas est géré différemment car seul la première commande peut prendre une redirection en entrée. Elle est donc interprété de la même manière que : grep k < abc.txt | sort | tail -1 | cat > resabc.txt.