COMPTE RENDU
L3 – SR MINI-SHELL
FROLOV – DO

1. Réalisation des segments shell

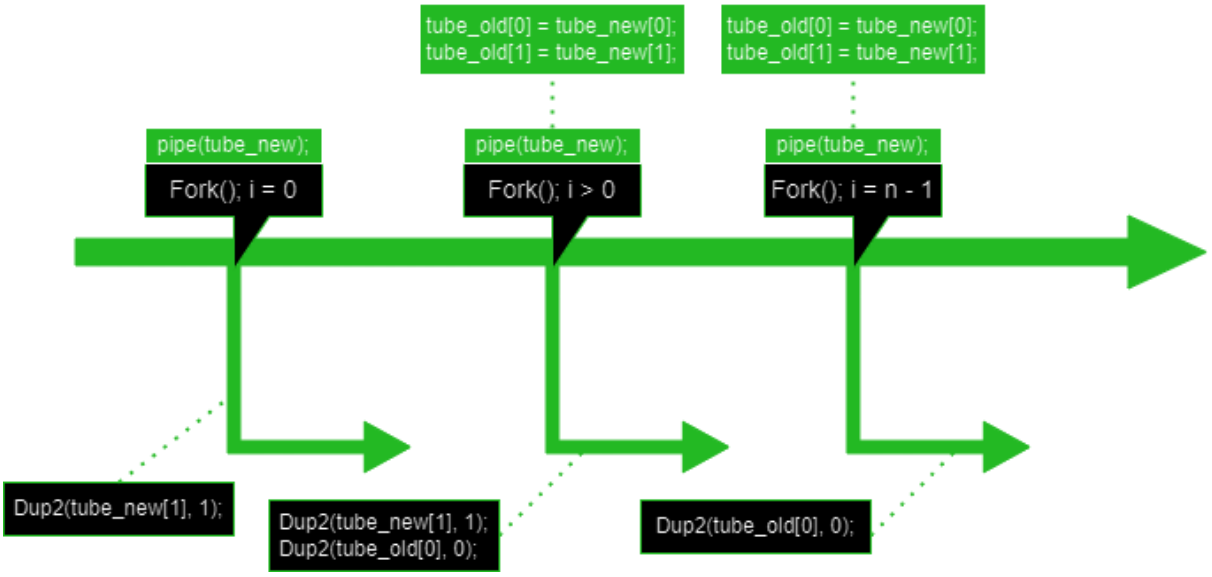
– Redirection:

Nous avons repris le schéma donné dans ***shell.c*** ou nous avons rajouté la gestion des fichiers en les ouvrant avec des droits correspondants pour in/out et en capturant des erreurs éventuelles avec l'ouverture et l'accès.

– Pipes & exécution:

Pour exécuter une seule commande, nous avons écrit une fonction *execute(char **cmd)* où nous utilisons *execvp()* en raison de factoriser l'exécution des commandes pour tube et traiter les erreurs des commandes invalides.

Pour mettre en œuvre l'exécution avec des tubes, nous utilisons deux tableaux pour stocker des pipes *tube_old[]* et *tube_new[]* afin que deux commandes successives puissent gérer la redirection (trivialement, la commande **i** écrit pour la commande **i+1**). Nous échangeons les anciens et nouveaux tubes en rédigeant les in/out avec la primitive *Dup2()* et fermons les descripteurs de fichiers-tubes inutiles au fur et à mesure de notre progression dans le struct *l*. Pour bien les exécuter en parallèle on utilise la primitive *Fork()* et *pipe()*.



Schema de l'exécution pour n pipes

– Gestion des zombies:

Principalement, nous gérons les processus zombies en définissant l'handler *handlerSIGCHLD()* pour un signal ***SIGCHLD***, où nous utilisons une boucle *while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0*) pour suivre les tâches terminées, ne pas perdre le fils et ne pas bloquer le père. Étant donné que la terminaison est directement liée à la mise en œuvre des *jobs*, nous suivons également la condition de fin du travail ici pour gérer l'historique des jobs. Une fois que le travail est terminé dans le gestionnaire, nous incrémentons le compteur *nb_reaped* afin que le processus parent dans ***main()*** attende que tous les travaux soient terminés.

– Premier plan et arrière-plan:

Nous avons d'abord ajouté un champ booléen *foregrounded* dans la structure **cmdline** afin de pouvoir définir si la commande s'exécute en arrière-plan ou en premier plan. Si la commande est en premier plan, nous lui donnons le contrôle du terminal, attendons sa mort et reprenons le contrôle.

– Jobs:

Pour implémenter des jobs, nous avons créé une structure **job_t** qui suit, par un job, son groupe id, son état, son nombre de processus en train d'exécuter et son nombre de processus qui n'arrive pas encore a sa mort. Si tous les processus dans un job attendent ses morts, le job est fini. Si tous les processus sont arrêtés, le job est arrêté.

2. Structure du code générale et organisation du main

– Les modules ajoutés:

– ***gid_tracker.c/.h*** – pour suivre les groupes ids des processus

– ***util.c/.h*** – fonctions diverses qui aident le shell

– ***jobs.c/.h*** – pour définir la structure ***job_t*** et les fonctions associées (exemple: *set_job_state(int number; short state)* pour changer l'état pour un job numéro *number* dans le liste des jobs)

– *shell.c* / main:

– dans ***shell.c*** nous avons défini la fonction *execute(cmd **)* mentionnée ci-dessus, ainsi que la fonction *end_session()* pour libérer la mémoire des structures utilisées. De plus, nous avons défini des gestionnaires de signaux utilisés aussi : *handlerSIGCHLD()* pour ***SIGCHLD***, *handlerPrintNewLine()* pour ***SIGTSTP*** et ***SIGINT***.

– Le programme principal ***main()*** est essentiellement le squelette fourni de la boucle *while(1)* où nous traçons les redirections, gérons les stop et quit, et parcourons la structure **l** pour exécuter des commandes en utilisant toutes les parties définies dans (1) de notre projet. Avant la boucle, nous associons les gestionnaires, initialisant l'historique des tâches, ajoutons quelques fonctionnalités mineures pour l'expérience utilisateur, comme le délai de ligne.

3. Tests

Nous avons créé des fichiers tests pour le script fourni pour tester la redirection, exécution, pipes et le début des *jobs*.

– Redirection:

– *test_red_simple.txt* – test redirection sortie

– *tes_red_plus.txt* – test redirection entrée/sortie.

– *test_red_err.txt* – test redirection sans droit d'accès.

– Exécution simple:

– *test_exec_simple.txt* – test plusieures commandes valides sans pipes.

– *test_exec_invalid.txt* – test commande avec l'argument invalide.

– *test_exec_invalid_cmd.txt* – test commande invalide.

– Exécution avec des pipes:

– *test_pipe_simple.txt* – test seul pipe (*sleep* + commande).

– *test_pipe_en_redirection.txt* – test qui fait redirection pour la commande pipé (ls + wc).

– *test_pipe_multiple.txt* – test trois pipes.

– Jobs:

– *test_arriere_plan.txt* – test de lancement en arrière plan (*sleep* + &).

– *test_jobs.txt* – test de la commande jobs.

En tant que les signaux et fonctionnement des jobs en générale, nous avons testé l'implémentation directement sur le shell écrit, nous avons essayé de voir le comportement de: fg, bg, stop, réaction des processus au signaux,...