Rapport projet Tesh

I) Phase de réflexion

Du 22 octobre au 11 novembre. Nous avons implémenté un simple “doit” et notre propre fonction split pour les chaines de caractères.
Ensuite nous avons réfléchi à une structure de données car nous nous sommes rendus compte qu’une simple liste de chaines de caractères n’était pas très adaptée.

Nous avons donc retenu un choix de structure de données sous forme d’une liste chainée. Chaque élément de la liste contiendrait une commande principale à exécuter ainsi que tous les arguments nécessaires à son exécution. Cette structure nous a paru très adaptative car on pouvait facilement rajouter des attributs à la commande et elle rendait le parsing de la commande relativement simple.

Une fois la structure de données choisie, nous avons regardé plus en détail comment serait structuré notre programme. Nous avons choisi de faire une boucle principale “while(1)”, un tour de cette boucle correspondrait à l'exécution d’une ligne de commande (une ligne de commande = tous les éléments d’une liste chaînée). Dans cette boucle nous avons donc une autre boucle while qui gère l'exécution de la liste chaînée.

Schéma retenu de la structure de commande :

```
struct Commande{

        //chained list
        Commande* next;
        Commande* previous;

        //list of attributes to handle the different tokens
        int end_status;
        int anded; //if it is preceded by “&&”
        … //more attributes for all tokens || , | , < , > ...
}
```

Schéma retenu du main :

```
while(1){
        chaine_caractères = fgets(stdin);
        commande = createStructureCommande();
        parsing(commande, chaine_caractères);

        while(commande.next != NULL){
                pid = fork()
                if(pid == 0){
                    execute(commande);
                }
                else{
                        waitpid(pid);
                }
                commande = commande.next;
        }
}
```

2) Implémentation

Nous avons commencé en codant l'intégralité du projet dans un seul fichier, puis au fur et à mesure des avancées, nous avons introduits de nouveaux fichiers pour mieux se retrouver dans la globalité du projet. Ainsi ont successivement été créés les fichiers StringManagement, qui contient majoritairement la fonction split, qui sert à découper correctement une chaine de caractère qui correspond à une commande, et la fonction contains, qui cherche si un argument est présent ou non dans une liste de chaines de caractères, et CommandStruct dans lequel est défini la structure de commande.

Finalement, la structure de commande est consituée de 13 attributs, qui permettent de connaître le contexte exact de la commande à effectuer, c'est à dire quelles sont les commandes précédentes et suivantes (si il y en a).
Nous avons donc remarqué que la structure s'avère donc être assez lourde, peut être aurait il été plus judicieux de séparer les commandes et leurs arguments de la gestion des tokens.
Nous avons aussi constaté lors de l'implémentation que l'adaptabilité de notre structure a rendu l'implémentation de la gestion de presque tous les tokens très simple (je parle ici de la structure de la fonction execute). Seul le pipe a posé problème. Comme il nécessite une gestion différente du fork() et que nous n'avions pas pensé en amont au fait que les commandes reliées par des pipes sont exécutées en parallèle, la structure du programme n'était pas bien adaptée.

Afin de pouvoir facilement résoudre les problèmes de segfault ou de fils que nous n'aurions pas exit, nous avons mis en place un mode debug (make D=DDEBUG pour compiler en debug). Dans ce mode, de nombreux prints étaient réalisés sur stderr afin d'assurer l'affichage chronologique des informations. chaque print du mode debug indique le pid ainsi que la fonction depuis laquelle le printf était effectué.

3) Phase de tests

Premièrement nous lancions simplement des commandes à la main depuis le mode interactif car nous avons tardé à implémenter le mode non interactif.
Nous comparions alors le résultat de notre commande avec ce qui était attendu.

Une fois le mode non-interactif implémenté, nous avions des fichiers de test contenant des commandes à tester.

Nous lancions les commandes de la manière suivante :

cat "nomdufichier" | ./tesh

Ainsi nous avions une batterie de commandes prêtes à être testées rapidement.

Nous avons aussi mis en place quelques tests simples automatiques, en modifiant le fichier ".gitlab-ci.yml" mis à disposition, ce qui nous a permis de vérifier qu'à chaque modification, le projet ne régressait pas et que l'implémentation de nouvelles fonctionnalités n'avait pas d'influence néfaste sur le bon fonctionnement des fonctionnalités précédemment implémentées.

4) Difficultés rencontrées

Une autre difficulté rencontrée est liée au fait que Mathieu a commencé l'implémentation du projet plus tôt que Valentin, ainsi, au début, Valentin était capable de discuter des avancées du projet en terme de réflexion sur les structures et les fonctions mais pas encore de les implémenter lui même. Il a fallu attendre que Valentin rattrappe son retard sur le code pour qu'il puisse coder lui même de nouvelles fonctionnalités.

Nous avons eu des difficultés liées à des incompréhensions de ce qui était demandé surtout au niveau des différents modes (interactif / non interactif). Cette incompréhension a rendu les résultats des premiers tests blancs inexploitables. ce problème aurait du être évité car toutes les informations étaient données dans la suite du sujet (fonction issaty, lancement des tests via des pipes, test de fgets si EOF est atteint).

Le mode DEBUG n'était pas prévu à la base. Il a en effet été mis en place après de longues heures à tenter de corriger un bug lié à un fils qui ne se terminait pas dans certains cas. En effet, beaucoup de bugs dans ce projet n'étaient pas facilement débogables à cause des processus multiples.

5) Répartition horaire des tâches

Réflexion sur la structure à utiliser : 3h Valentin et Mathieu

Implémentation du premier main exécutant un ”doit” : Mathieu 1h
Implémentation de cd : Mathieu 3h (mauvaises piste avec la variable environ)
Implémentation des fonctions de StringManagement : Mathieu 2h

Implémentation de la fonction de parsing et de la structure de donnée : Mathieu 2h
Première implémentation de la gestion des tokens &&, || et | : Mathieu 3h
Documentation du code et correction de bugs, mise en place du mode DEBUG : Mathieu 6h

Implémentation de issaty, test fgets==NULL et readline (non dll) : Mathieu 3h
Implémentation des redirections de fichier : Mathieu 1h
Correction de bugs et deuxième implémentation de pipe Mathieu 2h

Implémentation du lancement sans erreur et du lancement de script Mathieu 2h
Implémentation de la fonction built in fg : Mathieu 30min

Réalisation de tests et modification de “.gitlab-ci.yml” Valentin 1h
Compréhension du code déjà implémenté Valentin 6h
Chargement dynamique de libreadline.so Valentin 3h
Rédaction du rapport Mathieu et Valentin 2h