**Système d’exploitation :** Maël Tallard, Noah Raimbaud

Rapport : Projet création d’un shell en C

Introduction :

Le 17 mars 2025, nous avons reçu pour projet la création d’un shell en C à l’occasion du cours de système d’exploitation. Ce projet s’est étalé sur 5 semaines et 6 séances au cours desquelles nous avons pu réaliser l’intégralité du projet en remplissant le cahier des charges. Nous avions déjà expérimenté le langage C dans différents TP ainsi que dans nos précédentes années de formations mais jamais dans le cadre de projets qui touchent directement au système. Le défi était important, étant donné que nous n’avions aucune idée du fonctionnement d’un shell, de la gestion de processus, etc… Dans ce rapport, nous allons tout d’abord vous présenter le cahier des charges d’origine. Puis nous expliquerons en détail comment s’est déroulé le projet et enfin nous conclurons.

Cahier des charges :

Il s’agissait de créer en C un shell incluant toutes les fonctionnalités de base sans erreurs, ni warning. Les fonctionnalités sont les suivantes :

* le lancement d’un programme dans un nouveau processus depuis une commande
* la navigation dans les fichiers du pc avec la commande cd
* la gestion des entrées et sorties des processus
* le lancement de plusieurs processus depuis la même commande grâce aux pipes
* la mise en foreground et en background des processus
* l’intégration d’une fonctionnalité de copie des fichiers et des répertoires que nous avons réalisé auparavant

Déroulé du projet :

1 - Les recherches et expérimentations

Comme dit dans l’introduction, nous partions sans aucune connaissance du fonctionnement d’un shell. Il y avait même un écart de connaissances au sein du binôme, l’un était familier avec le fonctionnement d’un terminal zsh, tandis que l’autre ne s’en était quasiment jamais servi. Nous avons donc utilisé la documentation GNU fournie en annexe du sujet. Elle était très complète et nous a permis de progresser rapidement. Les différentes structures nécessaires au projet (processus, job) étaient déjà implémentées. De nombreuses fonctions étaient également définies telles que : init\_shell, find\_job, job\_is\_stopped, job\_is\_completed, launch\_process, launch\_job, put\_job\_in\_foreground, put\_job\_in\_background, mark\_process\_status, update\_status, wait\_for\_job, format\_job\_info, do\_job\_notification, mark\_job\_as\_running et continue\_job.

Ces ressources sont la base même de notre projet et se retrouvent dans notre code source, mais il s’agissait surtout de les comprendre pour pouvoir les utiliser. Dans la première séance, nous sommes partis dans deux directions différentes, avec une personne qui lisait la documentation GNU et l’autre qui faisait des recherches sur le fonctionnement des terminaux présents sur les systèmes UNIX.

Au milieu de la séance nous sommes passés à la programmation, nous avons tout d’abord créé un repository GitHub pour mettre en commun notre travail. Là encore nous avions peu d’expérience avec l’outil, mais son utilisation s’est révélée assez simple.

2 - Implémentation des fonctionnalités

La première étape consistait évidemment à réutiliser le code de la documentation GNU, tout s’est fait très facilement. Nous avons implémenté une fonction main contenant une boucle while(true) demandant en permanence une entrée de l’utilisateur input qui est ensuite analysée.

Il faut distinguer deux types de fonctionnalités dans le terminal, les fonctionnalités intégrées (cd, …) et les fonctionnalités externes (cp, pwd, ls, …). Les fonctionnalités intégrées sont directement implémentées au sein du terminal, et les fonctionnalités externes sont des fonctionnalités qui sont en réalité de petits exécutables stockés dans un fichier /bin.

Lorsque le terminal reçoit une commande, il doit détecter si cette commande appelle un exécutable externe ou une fonctionnalité propre au terminal. L’input va donc passer par différentes boucles “if” et “else if” qui vont analyser le début de la commande pour y trouver les mots clés (exit, jobs, fg, bg cd et cp) et lancer le code correspondant. Sinon, on va lancer un job avec le contenu de la commande.

L’implémentation du cd et de l'exécution de programme ne nous a pas posé de problème particulier et s’est déroulée assez vite, en partie à l’aide de modèles de langage génératifs.

En voyant notre progression rapide, nous avons décidé d’ajouter des fonctionnalités qui n’étaient pas demandées, la principale est sans doute le support des flèches directionnelles et l’ajout d’un historique des commandes grâce à la bibliothèque GNU readline.

Pour gérer les entrées/sorties, nous avons créé une fonction parse\_command dont le but est de découper la commande en input pour y extraire les différents éléments. À l’intérieur de cette fonction se trouvent donc des conditions chargées de vérifier si le chemin d’un fichier se trouve dans la commande et de l’ouvrir en lecture ou en écriture selon la commande.

Nous avons ensuite enchaîné à la séance 3 avec l’implémentation des pipelines qui s’est aussi faite rapidement.

La dernière étape difficile était l’ajout de la mise en background et foreground des programmes, cela s’est fait pendant les vacances et a demandé plus de temps. Il fallait tout d’abord détecter le signe & en fin de commande et lancer le processus en foreground ou background selon cela. Nous avons implémenté la commande jobs pour afficher l’ensemble des processus du terminal, avec leur indice, leur PID et l’état du processus. Puis, pour permettre de remettre en foreground un processus qui avait été lancé en background, il a fallu créer une fonction pour retrouver le processus en fonction du numéro de processus indiqué par l’utilisateur en paramètre de fg. La commande fg supporte donc l’indice du processus donné par la commande jobs, ou bien le PID.

Nous avons terminé par l’implémentation de notre fonction de copie de répertoires et de fichiers, cela s’est fait assez simplement en interceptant la commande cp pour ne pas appeler la méthode présente sur nos PC.

3 - Les difficultés rencontrées et comment nous les avons surmontées

Une des difficultés du projet était la lisibilité du code. Nous avions à la fin du projet un fichier de 900 lignes avec de très grosses fonctions, la solution a été de “découper” notre projet en 4 fichiers avec leur header. Le fichier Main contient le code contenant la boucle logique du terminal (while (true)). Le fichier Terminal contient toutes les fonctions nécessaires au fonctionnement du terminal (lancer le terminal, créer un processus, gérer le background et foreground, …). Le fichier Parse contient la fonction parse\_command qui est la plus longue et complète du projet. Pour finir, le fichier Copy contient la fonction de copie de répertoires et de fichiers.

La plus grosse difficulté que nous avons rencontré était que nous ne travaillons pas sur la même plateforme, nos deux PC étaient de la famille UNIX mais l’un était sur une distribution Linux (celui de Maël) et l’autre tournait sur Mac OS (celui de Noah). Cela nous a posé problème pour l’ajout du passage de background à foreground car cette fonctionnalité ne tournait pas sur Mac OS, nous avons essayé un grand nombre de versions de notre programme. Après différents tests nous n’avons pas trouvé de solution pour cet OS, malgré tout cette fonction tourne bien sur Linux.

Conclusion :

Pour conclure, nous sommes très satisfaits du résultat. Ce projet qui était le premier projet de groupe de l’année a été très enrichissant, nous avons appris à utiliser le système de contrôle de version git et la plateforme GitHub. Nous avons également beaucoup appris sur le fonctionnement d’un terminal et sur la gestion des processus d’un système d’exploitation. De plus l’intitulé du projet nous a beaucoup plu, cela nous a poussé à travailler en dehors des séances pour avancer rapidement et implémenter des fonctionnalités qui n’étaient pas demandées. C’était aussi la première fois que nous travaillions au plus proche du système et cette expérience nous a montré l’intérêt d’utiliser le langage C pour de tels usages.