Nathan ANTONI, François HAEGEL

Rapport de projet : Mini Shell “my_sh”

Rappel du sujet :

Le but de ce projet était de mettre en oeuvre les notions abordées en cours et dans les TPs afin de réaliser un interpréteur de commande simplifié, capable de recevoir et d’exécuter les commandes entrées par l’utilisateur.

Organisation :

Gestion des sources :

Pour réaliser ce projet, nous avons utiliser le gestionnaire de code source **Github**. C’est l’outil que nous maîtrisons et utilisons le plus.

Gestion de projet :

Nous avons utilisé l’outil de gestion de projet **Trello** pour organiser et planifier la réalisation des différentes tâches et fonctionnalités de l’application. Cela nous a permis de toujours avoir une vision globale comme détaillée de l’état d’avancement du projet.
Trello nous a aussi beaucoup aidé lors de la répartition des tâches, en effet, le système d’attribution de cartes se combine très bien avec un découpage en petites tâches, qui nous convient très bien.

IDE :

Concernant les IDEs, nous avons tous les deux utilisé **Visual Studio Code**, là aussi, c’est un outil avec lequel nous sommes à l’aise, qui est bien plus que suffisant pour coder en C.

Description de la solution développée :

Fonctionnalités réalisées :

FM01 – Le binaire est capable d’exécuter une commande simple (ie : ls –l ; ps ; who)
FM02 – Le binaire est capable d’exécuter un sous-ensemble de plusieurs commandes de sorte à prendre en compte :

- Les opérateurs de contrôle : && et ||
- Les redirections de flux simples : |, >, <, >>, <<
- L’exécution en arrière-plan : &

FM03 – L’exécution des commandes internes (fonctionnalités built-in) suivantes :

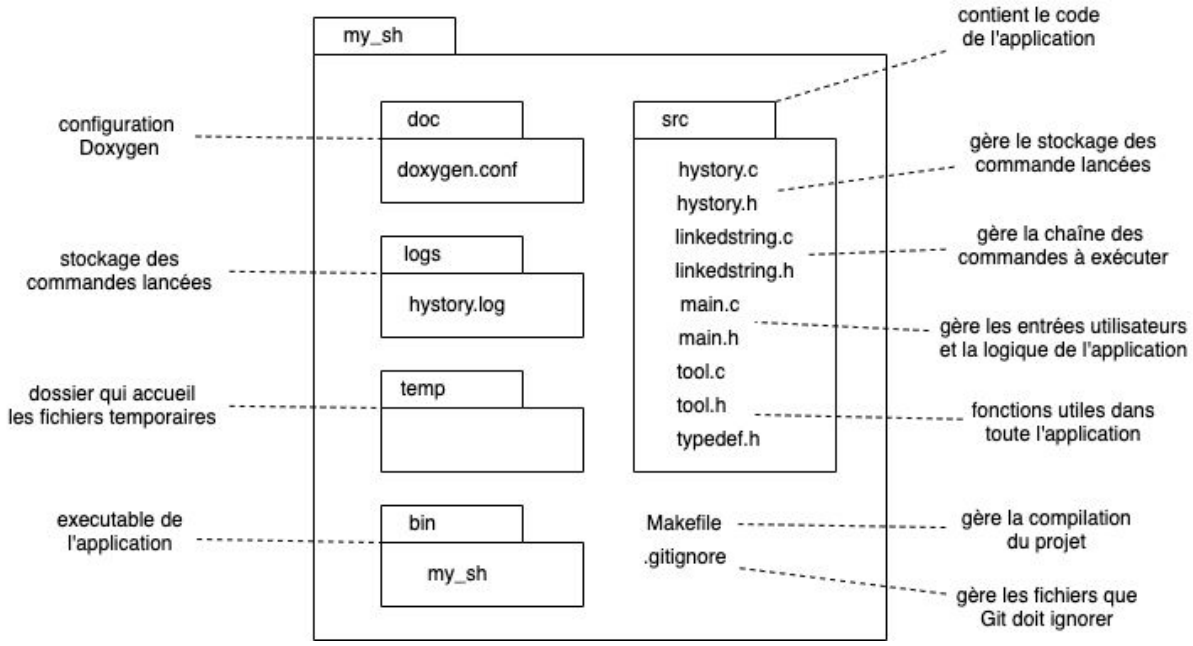
- cd - Permettant de se déplacer au sein d’une arborescence de fichier.
- pwd – Affichant la valeur de la variable contenant le chemin du répertoire courant.
- exit – Permettant de quitter l’interpréteur.
- echo – Permettant d’afficher du texte sur la sortie standard.

FM04 - La persistance des commandes saisie dans un fichier (historique)

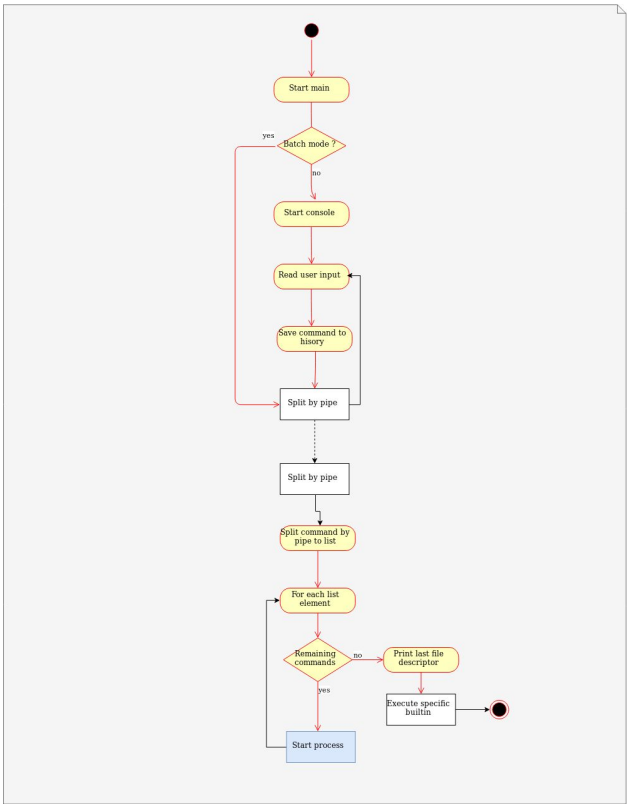
Nathan ANTONI, François HAEGEL

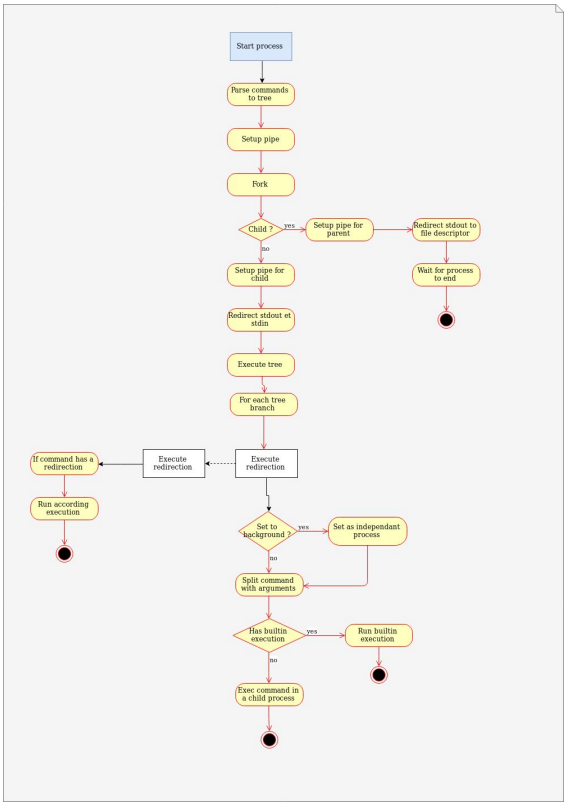
FO01 - La réalisation d'un mode batch (ie : ./my_shell –c « ls –al | grep toto »)

Architecture du projet :



Fonctionnement :





Commandes :

- **$ make** : compilation du projet
- **$ make doc** : génère la documentation Doxygen
- **$ make mrporper** : supprime les fichiers de documentation et les exécutables
- **$ bin/my_sh** : lancement du mini shell

Difficultés rencontrées :

Nous avons été confronté à plusieurs difficultés lors de la réalisation de l'application :

- découper correctement les chaînes passées en arguments
- mettre en place l'opérateur de redirection "<<", dont nous avons d'abord dû comprendre le comportement (ses interactions avec les autres commandes surtout).
- choisir l'architecture à mettre en place pour le gestion des commandes imbriquées
- nous avons essayé de mettre en place la possibilité de parcourir l'historique des commandes avec les flèche UP et DOWN, nous y étions presque parvenu à l'aide de la librairie ncurses, cependant cette dernière redéfinit les paramètre du terminal, et complexifie le formatage de la sortie standard (problème que nous n'avons pas su résoudre dans les temps).
- mettre en place les pipes qui étaient une notion assez complexe pour nous (surtout dans la pratique).

Améliorations possibles :

- utilisation d'alias
- utilisation de variables d'environnement
- pouvoir lancer des commandes qui rafraîchissent la sortie standard (ex: top)
- faire fonctionner le parcours de l'historique des commandes avec les flèches UP et DOWN
- pouvoir se déplacer dans une commande pour pouvoir la modifier avec les flèches RIGHT et LEFT
- rendre les pipes asynchrones (ex: sleep(3) | sleep(5) | sleep(4) -> attend 12 secondes alors qu'un vrai shell attend 5 secondes)