TELECOM NANCY

RAPPORT DU PROJET RÉSEAUX ET SYSTÈMES : MODULE RS

Octobre - Décembre 2021

Projet RS 2021

Étudiants :
Benjamin JURCZAK
Alexandre DHENIN

Numéro Étudiant :
32022703
32024369

Responsable du module :
Lucas Nussbaum

Table des matières

1 Introduction

Le projet du module RS 2021 consistait à implémenter un shell de type bash, nommé "tesh" Le but de ce projet était de mettre en application les notions vues lors du cours de système de la première partie du semestre.
Les fonctionnalités demandées pour tesh étaient listées dans le sujet du projet. Au nombre de 10, elles regroupaient l'ensemble des fonctionnalités de base d'un shell (redirections d'entrée et de sortie, enchainements de commandes, mode non interactif, ...).
Leur implémentation avait pour but d'obtenir en fin de projet un shell fonctionnel, répondant aux besoins classiques d'un utilisateur. Ces besoins ont été modélisés par un ensemble de tests, proposés régulièrement par le responsable du module, Lucas Nussbaum. Les tests ont permis au cours de la réalisation du projet d'apprécier le bon comportement de tesh.

2 Choix de conception

2.1 Généralités

Pour l'implémentation du shell "tesh", nous avons procédé dans l'ordre suggéré par le sujet, en codant chaque partie, les unes aprés les autres.
Les tests blancs ont été d'une importance capitale et ont permis d'adapter le code pour qu'il corresponde aux attendus du sujet, qui reflètaient les commandes essentielles qu'un shell doit être capable éxecuter. Nous nous sommes donc beaucoup basé sur les tests blancs pour avoir une idée de l'exactitude de notre code.
De plus, il est important de préciser que nous avons choisi de travailler avec des fonctions récursives, qui facilitent la tâche lors de l'exécution de commandes à rallonge.

2.2 Fonctionnement de tesh

Dans cette partie, nous expliquons les étapes qui vont mener à l'exécution d'une commande passée au shell. De la récupération de cette commande à son exécution en passant par plusieurs disjonctions de cas selon le type de commande (redirections, instructions simples, présence de backgrounds...), nous allons suivre le cheminement de ce qui se produit en interne.

L'exécution du fichier *tesh.c* permet le lancement de sa fonction main, lors de laquelle on vérifie en premier lieu si des options sont passées en plus de ./tesh (par exemple "-e", "-r", un fichier de commande ou une utilisation en mode non interactif).
Si l'utilisateur souhaite utiliser tesh en mode "prompt" (interactif), ce dernier s'affiche et l'utilisateur peut entrer une commande.
La commande d'entrée est récupérée (fonction fgets ou readline si présence de l'option -r) et stockée dans un tableau *instruction* duquel on supprime le retour chariot à la fin. Ce tableau est ensuite placé en argument de la fonction *parseur*. Cette fonction agit comme un séparateur et "scinde" la commande selon le caractère espace (la cas sans espace n'étant pas à gérer comme précisé dans le sujet). Chaque "morceau" de la commande est alors rangé dans un nouveau tableau de char*.
Ensuite, *parseur* appelle la fonction récursive *instruction_bg* avec ce tableau en paramètre. Dans cette fonction, ont lieu des tests pour savoir si certaines instructions doivent être exécutées en arrière plan. Si il y en a ("&" détecté) on procède aux traitements nécessaires (non détaillés dans ce court

rapport). En l'absence de telles instructions, la fonction *parenthese* est appelée, elle va tester si une commande est constituée de parenthèses et permettre d'exécuter les instructions dans le bon ordre. Cette fonction, est expliquée plus en détails dans la partie suivante.
Si la commande ne contient pas de parenthèse (le cas où elle en contiendrait est évoqué en partie 2.3), la fonction récursive *instruction_simple* est appelée. Dans cette dernière, les arguments sont le tableau de chaîne de caractère contenant la commande, un caractère (un symbole : "&&", "||", ";") donnant la nature de l'enchainement précédent et un entier (0 ou 1) représentant la sortie de la commande précédente (succés / échec). Dans cette fonction, on parcourt la commande et on stocke dans deux tableaux le début (tout ce qui se trouve avant le premier symbole détecté) et la fin de la commande. Ensuite, la fonction distingue les cas selon le symbole trouvé ("&&", "||", ";") et l'état de la sortie précédente. La commande détectée par *instruction_simple* est ensuite passée à *instruction_redi*.
Cette fonction, est aussi récursive et traite toute les situations de redirection ("<", "|", ">", "»") grâce à une utilisation astucieuse des descripteurs. C'est dans cette fonction que se fait l'appel à la fonction exécution, qui va permettre l'exécution finale de la commande, c'est lors de celle-ci que vont se faire des disjonctions de cas telles que détecter si la commande commence par "cd", si on appelle la fonction interne "fg", si le code de retour est faux etc.

Ceci termine la présentation du cheminement du traitement d'une commande envoyée à tesh. Rappelons que par soucis de concision, les explications sur ces traitements internes de la commande faits par notre shell ont été très grandement simplifiées. Beaucoup de précisions, notamment sur les appels récursifs et la récupération des arguments pourraient être apportées et faciliter la compréhension du fonctionnement.

2.3 Possibilité de gérer les parenthèses

La fonction récursive *parenthese* n'a pas été nommée au hasard. Cette fonction permet de gérer les enchainements de commandes même si ceux-ci contiennent des parenthèses. Par exemple, si la commande suivante passée à un shell de type bash :

```
( false && echo a ) || echo b || ( echo c && echo d ) || echo e && echo f
```

Le retour doit afficher :

```
b
f
```

Tandis que sans parenthèses, l'affichage serait le suivant :

```
b
d
f
```

Le fonctionnement est le suivant :

Lorsqu'une commande est passée au tesh, cette dernière est "parsée" dans un tableau *instructions*. La fonction *parenthese* prend (entre autres) ce tableau en argument. Ce tableau est parcouru jusqu'à trouver le symbole séparateurdes deux parties de la commande. Dans l'exemple précédent ce serait le premier "||" : ce qui le précède correspond à la première partie de la commande et la deuxième

partie est constituée de tout ce qui vient après.

Pour trouver ces deux parties, deux compteurs permettent de compter le nombre de parenthèses droites et gauche rencontrées jusque là. La condition d'arrêt de parcourt du tableau est qu'on rencontre un des symboles "", "||" ou ";" et que le nombre de parenthèses ouvrantes soit égale au nombre de fermantes (avec uniquement la première condition, on pourrait être à l'intérieur d'une "sous commande" dans une parenthèse pas encore fermée).
Puis selon les cas, plusieurs possibilités :

- La commande ne contenait pas de parenthèses : on envoie le tableau à la fonction *instruction_simple* ;
- La commande était effectivement composée de parenthèses : On stocke dans deux tableaux les deux parties de cette commande en enlevant les parenthèses englobant la première partie (celles avant le false et après le echo a pour l'exemple précédent). Ensuite, parenthèse s'appelle récursivement diféremment selon les cas (selon le symbole séparateur trouvé) et le schéma se répète. A chaque appel, la commande se réduit jusqu'à l'avoir traitée entièrement, en ayant extrait dans le bon ordre les blocs à exécuter, et qu'on passe à *instruction_simple* pour traitement "normal" de la sous commande.

3 Difficultés

3.1 Pipes multiples

Pour les pipes multiples et les redirections, la difficulté a été de se représenter le problème. Imaginer une solution a été difficile et du fait du caractère assez abstrait du problème, le schéma d'exécution n'était pas évident.
En représentant le cheminement du problème sur une feuille, nous sommes arrivé à une solution utilisant des descripteurs de façon récursive.

3.2 Option "-e"

L'option *"-e"* de tesh a pour but d'arrêter le shell quand une commande a un code de retur différent de 0. L'ajout de cette fonctionnalité a mis du temps à être effective car il a fallu introduire des *"exit"* et donc revoir pratiquement tout le code. Une révision du cours et une familiarisation avec la commande exit a été nécessaire pour y parvenir.

3.3 Background

Les background (commandes en arrière plan) font partie des fonctionnalités à implémenter qui ont le plus posé posé problème.
Dès le début, la compréhension de l'énoncé a demandé refléxion. Puis c'est finalement grâce à une réponse sur le READ.ME du projet proposée par le responsable du module (*2021-12-10 Problème dans l'ordre des sorties sur les tests bg*) que l'implémentation a pu commencer. Il a fallu bien saisir l'enjeu de la fonctionnalité et réfléchir à la manière dont nous allions l'implanter. Pour la commande fg sans argument par exemple, il fallait pouvoir identifier les processus en arrère plan pour que le shell puisse attendre la fin de ces processus (cela implique donc de stocker les processus lancé en arrière plan).

3.4 Coordination d'équipe

La coordination entre les membres du groupe a parfois posé problème. En effet, lors d'un sprint, chacun changeait le script à sa manière et changeait les modifications dans son propre sens, dans le but de faire marcher sa partie. Le problème qui s'est posé était à chaque fois de se familiariser avec le nouveau code, ce qui prenait du temps et ce qui amenait parfois à des modifications sur son propre travail.
La forte dépendance entre les missions de chacun a donc pu mener à certains conflits. Il a fallu prendre du temps lors de chaque réunion (d'où leur longueur) pour se mettre à jour et s'expliquer mutuellement comment le code fonctionnerait dorénavant.

4 Gestion de projet

4.1 Méthode agile

Le type de projet proposé se prêtait grandement à l'utilisation d'une méthode agile : sujet progressif, points séparés les uns des autres, nombre de semaines approximativement égale au nombres de points du sujet (et donc de sprints).

Ainsi, nous avons opté pour une gestion de projet en mode agile avec chaque semaine une mission pour chaque membre qui devait être réalisée pour la semaine suivante. Les sprints duraient donc une semaine et permettait de faire un point hebdomadaire sur les nouvelles fonctionnalités du tesh. En effet, une réunion par semaine minimum avait lieu durant laquelle notamment les problèmes de chacun étaient explorés et une solution cherchée.
Les objets des sprints ont été pris dans l'ordre proposé par le sujet et étaient différents pour les deux membres. Lors de la première semaine par exemple, Benjamin avait pour mission de se familiariser avec le fonctionnement de base et notamment d'implémenter un parseur de commande tandis qu'Alexandre tentait une première élaboration de la commande cd.

4.2 Temps de travail

Le tableau ci-après rend compte des heures approximatives passées sur les différentes étapes du projet par membre du groupe.

	Benjamin	Alexandre
Conception	20	15
Implémentation	15	30
Tests	4	6
Rédaction du rapport	3	1

Table 1 – Temps de travail sur le projet (en heures)

Références

[1] https ://man7.org/linux/man-pages/man3/. Pages des man.

[2] https ://members.loria.fr/lnussbaum/rs.html. Cours de RS.

[3] https ://stackoverflow.com/. StackOverFlow.

[4] https ://www.youtube.com/watch ?v=_kia4d7kq8it=108sab_channel=jacobsorber. Explications pour l'utilisation de dlopen.

[5] http ://ymettier.free.fr/articles_lmag/lmag43_briques_en_c9/ar01s06.html. Explications pour l'utilisation de dlopen.