Projet 2021/2022 : tesh – TELECOM Shell

RS : Réseaux et Systèmes

Deuxième année

L'objectif de ce projet est de réaliser un shell (comme `bash`, `tcsh`, `zsh` ou `dash`). D'un point de vue pédagogique, il permettra de revoir (et de mettre en pratique) les différents concepts vus pendant la partie « système » du module de RS (manipulation de processus et de fichiers avec l'API POSIX, notamment).

1 Logistique

Groupes. Apprendre à travailler en groupe fait partie des objectifs de ce projet. Il vous est donc demandé de travailler en binôme (dont la composition est libre – il est possible de mélanger deux groupes de TD). Il est **interdit** de travailler seul (sauf pour au plus un étudiant, si vous êtes un nombre impair).

Tests automatiques. Une part importante de l'évaluation du projet sera faite à l'aide d'une batterie de tests. Il est donc très important, au cours de votre travail, d'accorder une large part à vos propres tests. Plusieurs «tests blancs» seront également organisés au cours du projet : votre code sera récupéré, compilé, et testé sur des cas de tests qui feront partie de ceux utilisés pour l'évaluation finale, et les résultats de ces tests vous seront communiqués. **Il est donc indispensable de commencer à travailler tôt pour bénéficier de ces «tests blancs» et avoir la garantie que votre projet fonctionne correctement.**

Rapport. Vous devez rendre un mini-rapport de projet (5 pages maximum hors annexes et page de garde, format pdf). Vous y détaillerez vos choix de conception, les difficultés auxquelles vous avez été confronté, et comment vous les avez résolues. Vous indiquerez également le nombre d'heures passées sur les différentes étapes de ce projet (conception, implémentation, tests, rédaction du rapport) par membre du groupe.

Soutenances. Des soutenances pourront être organisées (éventuellement seulement pour certains groupes, par exemple s'il y a des doutes sur l'originalité du travail rendu). Vous devrez nous faire une démonstration de votre projet et être prêts à répondre à toutes les questions techniques sur l'implémentation de l'application.

Plagiat et aide extérieure au binôme. Si, pour réaliser le projet, vous utilisez des ressources externes, votre rapport doit les lister (en expliquant brièvement les informations que vous y avez obtenu). Un détecteur de plagiat [1] sera utilisé pour tester l'originalité de votre travail (en le comparant notamment aux projets rendus par les autres groupes). Toute triche sera sévèrement punie (jusqu'à un 0 au projet).

Concrètement, il n'est pas interdit de discuter du projet avec d'autres groupes, y compris de détails techniques. Mais il est interdit de partager, ou copier du code, ou encore de lire le code de quelqu'un d'autre pour s'en inspirer.

Informations complémentaires. Des informations complémentaires pourront être fournies sur `https://gitlab.telecomnancy.univ-lorraine.fr/Lucas.Nussbaum/rs2021`. Ces informations complémentaires doivent être considérées comme faisant partie du sujet. Il est donc conseillé de surveiller cette page régulièrement.

Questions. Vos questions éventuelles peuvent être adressées à `lucas.nussbaum@univ-lorraine.fr`. Les réponses (et les questions correspondantes) pourront être publiées sur la page du projet.

1. `http://theory.stanford.edu/~aiken/moss/`

2 Description du projet

L'objectif général du projet est de réaliser un interpréteur de commandes, appelé `tesh`, très fortement inspiré des shells Unix classiques comme `sh`, `dash`, `bash`, `tcsh`, `zsh`.

Fonctionnement de base. Une fois lancé, le shell attend que l'utilisateur entre une commande, puis l'exécute, attend sa fin, et attend la commande suivante. Pour simplifier l'implémentation, les paramètres des commandes ne peuvent pas contenir d'espaces (il est donc inutile de gérer des cas comme `./monprogramme "premier argument" 'deuxieme argument'`), et les séparateurs de commandes ou caractères de redirections (`>`, `>>`, `|`, `&&` etc) sont forcément sur des *mots* distincts (inutile de gérer `./monprog>sortie`, ce sera toujours écrit `./monprog > sortie`.

Commande interne : `cd`. La commande `cd` doit être traitée de manière particulière, car, si elle était exécutée dans un processus fils, son effet serait limité à ce processus fils, et n'aurait pas d'effet sur la commande suivante. Il faut donc l'implémenter comme une commande interne du shell (on parle de `built-in`).

Affichage d'une invite de commande (*prompt*). En mode interactif, le shell affiche un *prompt* de la forme "`USER@HOSTNAME:REPCOURANT$ `". `USER` est le nom de l'utilisateur courant, `HOSTNAME` est l'hostname de la machine (cf `gethostname(2)`), et REPCOURANT est le répertoire courant.

Enchainement conditionnel de commandes. Comme dans `bash` : `cmd1 ; cmd2` doit exécuter `cmd1` puis `cmd2`. `cmd1 && cmd2` doit exécuter `cmd1`, puis `cmd2` seulement si `cmd1` a terminé avec le code 0. `cmd1 || cmd2` doit exécuter `cmd1`, puis `cmd2` seulement si `cmd1` a terminé avec un code différent de 0.

Redirections d'entrée et de sortie. Comme dans bash : `cmd > fichier` doit exécuter `cmd` en redirigeant sa sortie vers `fichier` ; `cmd >> fichier` doit exécuter `cmd` en ajoutant sa sortie à la fin de `fichier` ; `cmd < fichier` doit exécuter `cmd` en utilisant `fichier` comme entrée ; `cmd1 | cmd2` doit exécuter `cmd1` et `cmd2` en redirigeant la sortie de `cmd1` vers l'entrée de `cmd2` (pensez à généraliser : `cmd1 < fichier1 | cmd2 | cmd3 >> fichier2`).

Mode interactif et non-interactif. `tesh` doit aussi pouvoir s'exécuter en mode non-interactif, soit si un fichier contenant les commandes à exécuter est passé en paramètre (`./tesh monscripttesh`), soit si l'entrée standard de `tesh` n'est pas un terminal (à tester avec `isatty(3)`). Quand `tesh` s'exécute en mode non-interactif, alors il n'affiche pas de *prompt*.

Sortie sur erreur (`-e`). Quand l'option `-e` est passée à `tesh`, alors `tesh` s'arrête dès qu'une commande termine avec un code de retour différent de 0 (c'est particulièrement utile en mode non-interactif).

Lancement de commandes en arrière plan. Si une commande est suivie du séparateur de commandes `&`, alors elle doit être lancée en arrière plan. Le shell doit simplement afficher son PID, sous la forme `[pid]` (par exemple `[42]`). Ensuite, une commande interne `fg` doit permettre de ramener la commande au premier plan et d'en attendre la fin. Si `fg` est appelé sans paramètre, alors le shell attend la fin d'un des processus en arrière plan. Si un PID est passé en paramètre, alors le shell attend la fin de ce processus. Dans les deux cas, le shell doit afficher le PID du processus en arrière plan qui a terminé, et son code de retour, sous la forme `[pid->retcode]` (par exemple `[42->2]`, si le processus 42 a terminé avec le code de retour 2.

Édition de la ligne de commande du shell avec *readline*. Si l'option `-r` est passée au shell, alors le shell doit utilisée la bibliothèque *readline* pour permettre l'édition interactive de la ligne de commande, et la gestion de l'historique.

Chargement dynamique de la bibliothèque *readline*. Au lieu de lier le programme avec la bibliothèque *readline*, utilisez `dlopen` pour charger dynamiquement cette bibliothèque.

2.1 Conseils de réalisation

La quantité de code à produire est relativement faible (quelques centaines de lignes de code tout au plus). Mais le niveau de difficulté du code à produire est assez important. Il est crucial de programmer de manière prudente, réfléchie, claire et progressive, en testant bien les différents cas d'erreur.

Il est conseillé (mais pas obligatoire) de réaliser votre projet dans l'ordre qui suit.

Dans un premier temps, écrivez un programme qui lit une commande au clavier, l'exécute, attend sa fin, puis lit une autre commande, etc. C'est la base de votre shell. Il y en a pour 40 lignes de C tout compris, et c'est proche de ce qui a été fait en TD/TP.

Ensuite, si vous ne l'avez pas encore fait, c'est une bonne idée d'implémenter l'analyse de la ligne de commande saisie (découpage en mots), et d'implémenter par exemple "`;`". Pour la suite, vous avez vraiment le choix, mais c'est une bonne idée d'implémenter les choses globalement dans l'ordre où elles sont présentées dans le sujet.

3 « Rendu » du projet.

Le « rendu » du projet se fera via l'instance Gitlab de TELECOM Nancy. En dehors de la configuration correcte de votre dépôt, il n'y a rien à faire le jour de la fin du projet. Une page web (dont l'adresse sera communiqué sur le site du cours), et les tests blancs, vous permettront de vérifier la bonne configuration de votre dépôt.

Pour créer votre projet, il faut aller sur
`https://gitlab.telecomnancy.univ-lorraine.fr/projects/new` et choisir "Import project from" ... "Repo by URL", puis rentrer l'URL
`https://gitlab.telecomnancy.univ-lorraine.fr/Lucas.Nussbaum/rs2021-template`
Votre *Project name* doit commencer par `rs2021-` et contenir les noms des deux étudiants du binôme (par exemple `rs2021-dupont-durand`). Son *Visibility Level* doit être *Private*.
Une fois le projet créé, allez dans Settings, Members, et ajoutez *Lucas.Nussbaum* avec *Master* comme *role permission* (pour que le projet puisse être récupéré). Ajoutez également votre binôme.

Votre dépôt Git doit contenir, à la racine du dépôt :

- Un ficher `AUTHORS` listant les noms et prénoms des membres du groupe (une personne par ligne) ;
- Un `Makefile` compilant votre projet en créant un fichier exécutable nommé `tesh` ;
- Un fichier `rapport.pdf` contenant votre rapport au format PDF.

Le dépôt `rs2021-template` que vous avez utilisé comme base contient un fichier `.gitlab-ci.yml` qui permet de lancer des tests à chaque fois que vous *poussez* des modifications sur Gitlab. Il est conseillé (mais pas demandé) d'ajouter vos propres tests pour vous assurer que votre projet ne régresse pas.

4 Calendrier

- La version finale de votre projet sera récupérée le **dimanche 12/12/2021 à 21h00**. Il n'y aura aucune extension, ni possibilité de corriger votre code après cette date.
- Il est très vivement conseillé de commencer tôt, pour profiter au maximum des tests blancs qui vous permettront d'avoir un retour sur le fonctionnement de votre programme (visez d'avoir terminé le projet tôt, et profitez des tests blancs pour détecter les derniers bugs). D'autre part, la semaine 49 (du 06/12) est une semaine chargée en examens.

  Le premier test blanc devrait être lancé avant la fin du mois d'octobre.

  Vous ne serez pas prévenus avant le lancement des tests blancs : il faut donc faire attention à garder la branche *master* de votre dépôt Git fonctionnelle (par exemple, en travaillant dans d'autres branches [2].

5 Informations diverses

Pour demander à Git d'enregistrer vos informations de connexion, vous pouvez utiliser `git config credential.helper store` (voir `git-credential-store(1)`).

2. chapitre 3 de `https://git-scm.com/book/`