Systèmes et Outils

Licence 3 UPEC 2023/2024

Travaux Machine 6

Date limite de rendu : Mercredi 20 Décembre 2023 à 23h59

Voici quelques conseils, en vrac, afin de faciliter votre travail :
— lisez attentivement les consignes et respectez les ;
— commentez vos en-têtes de fonctions ;
— la documentation est dans le manuel en ligne : man(1) (tapez `man 1 man` pour accéder à la documentation de man) ;
— lisez les messages d'erreur et les avertissements du compilateur ;
— testez les valeurs de retour des fonctions ;
— versionnez régulièrement votre code ;
— **du code qui ne compile pas vaudra 0 ;**
— **un dépôt git vide au moment de la date limite de rendu vaudra 0.**

Nous allons, dans ce mini-projet, implémenter la version minimale d'un shell.

Pour rappel, le shell est l'invite de commandes permettant d'interagir avec la plupart des systèmes UNIX. Il suffit en général d'ouvrir une console pour y avoir accès, que ce soit sous Linux, Macos X, ou même sur certains non-UNIX comme Windows (en installant le WSL).

Dans tout l'énoncé, on pourra supposer que la longueur des commandes n'excède jamais 1024 caractères. L'emploi de la fonction `system` est bien évidemment interdit !

1 Instructions

Vous pouvez travailler par groupes de 1 ou 2.

Le rendu de vos programmes se fera sur le gitlab étudiants, dans lequel vous devez créer un dépôt.

Connectez-vous au gitlab étudiants, en vous créant un compte si nécessaire :
`https://git-etudiants.lacl.fr/`

La création de compte se fait ici :
`https://appetu.lacl.fr/profile/login`

Une fois que vous avez accès au gitlab, créez un nouveau projet. Renseignez ensuite l'URL de votre projet dans l'activité prévue à cet effet sur eprel (dans la section évaluation). À la fin de chaque exercice, versionnez vos fichiers.

2 Version dépouillée

Exercice 1. Pour commencer, écrivez un shell qui traite uniquement des commandes ne possédant pas d'arguments.

Le corps de votre shell consistera d'une boucle while infinie, dans laquelle il
— affichera une invite de commande (le caractère '$', suivi d'un espace) sur la sortie standard
— attendra que l'utilisateur ou l'utilisatrice rentre une commande sur l'entrée standard
— se dupliquera pour exécuter cette commande
— attendra la terminaison de ce nouveau processus avant de refaire un tour de boucle.

Si la commande passée est `exit`, alors le shell devra s'arrêter.

Fonctions importantes : `fork(2)`, `exec(3)`, `wait(2)`, `exit(3)`.

3 Traitement des arguments

Nous allons maintenant faire en sorte que notre shell accepte jusqu'à 16 arguments après la commande. Pour ce faire, il nous faut parser la ligne lue sur l'entrée standard.

Exercice 2. Écrivez une fonction `int parse_line(char *s, char **argv [])` qui découpe la chaîne `s`, terminée par le caractère nul, en fonction des espaces qu'elle contient : les mots séparés par des espaces doivent chacun être stockés dans une case différente du tableau `*argv`. La dernière case du tableau devra contenir `NULL` afin de marquer la fin du tableau.

Par exemple, si le chaine `s` est "`commande -v toto tata`", le tableau devra être rempli de la manière suivante :

commande	-v	toto	tata	NULL

Fonctions importantes : `strpbrk(3)` ou `isspace(3)`.

Exercice 3. Modifiez votre programme pour que le processus exécuté après chaque entrée de l'utilisateur ou utilisatrice prenne en compte les éventuels arguments donnés sur l'entrée standard.

4 Redirection de la sortie standard

Exercice 4. On veut maintenant que notre shell soit capable de traiter une redirection de la sortie standard vers un fichier. Pour simplifier, on pourra supposer que si redirection il y a, le caractère '>' sera toujours donné en avant-dernier argument et que le fichier sera donné en dernier argument.

Par exemple, on ne procédera pas à une redirection pour la ligne "`commande > fichier toto`", mais il faudra rediriger la sortie standard sur le fichier `fichier` pour la ligne "`commande toto > fichier`".
Fonctions importantes : `open(2)`, `dup2(2)`, `close(2)`.

5 Tubes simples

On désire implémenter une version simplifiée du chaînage de commandes. Pour éviter d'avoir à modifier notre parseur, on fera l'hypothèque que contrairement aux shells classiques, chaque commande chaînée sera attendue sur une ligne différente, et qu'il y en aura au maximum deux.

Exercice 5. Modifiez votre code de sorte que si le dernier argument d'une commande est le caractère '|', alors plutôt que d'exécuter immédiatement la commande, votre shell attende une deuxième commande, et redirige la sortie standard de la première sur l'entrée standard de la seconde.

Par exemple, la commande "`ls |`" suivie de la commande "`wc -w`" affichera en sortie standard le nombre de fichiers dans le répertoire courant.
Fonctions importantes : `dup2(2)`, `pipe(2)`.

6 Signaux

Exercice 6. Lorsque l'on appuie sur CTRL+C dans la console, un signal SIGINT est envoyé aux programmes qui y sont exécutés. Faites en sorte que votre shell ignore ce signal mais que les programmes qu'il a lancés ne l'ignorent pas.
Fonctions importantes : `sigaction(2)`, `sigprocmask(2)`, `kill(2)`.