mickael.hoerdt@hesge.ch 2017/2018

Mini-projet de programmation Système/Systèmes d'exploitation
Un interpréteur de commandes UNIX simplifié

Objectif

L'objectif de ce projet est d'implémenter un interpréteur de commandes Unix simplifié en C, similaire à `bash`. Il permettra tout de même de :

- Lire et d'écrire des variables d'environnement.
- Naviguer dans le système de fichier et créer des alias.
- Exécuter de manière interactive des commandes relatives à la variable d'environnement `PATH`.
- Rediriger les d'entrées/sorties de commandes exécutées.
- Lire depuis un fichier une suite de commandes à exécuter séquentiellement. Les commandes à exécuter seront séparées par un `\n` ou par un `;`

Cahier des charges

1. Variables d'environnement

Au démarrage, l'interpréteur doit récupérer le répertoire de travail de l'utilisateur ayant exécuté l'interpréteur, lire un fichier appelé `profile` dans ce répertoire et configurer les variables d'environnement spécifiées dans ce fichier, puis utilisées par la suite par l'interpréteur. Par exemple les deux lignes suivantes :

```
PATH=/bin:/usr/bin:/usr/local/bin
HOME=/home/hoerdtm
```

Positionnent le répertoire de travail de l'utilisateur ayant lancé l'interpréteur à `/home/hoerdtm` et positionnent la liste des répertoires dans lesquels rechercher dans l'ordre `/bin`, `/usr/bin`, `/usr/local/bin` pour le lancement d'un nouveau programme indiqué en premier mot de la ligne de commande. Votre interpréteur doit indiquer une erreur si l'une des deux variables d'environnement n'est pas positionnée dans le fichier `profile`. Il doit être possible d'assigner les variables `HOME` et `PATH` dans n'importe quel ordre et de réassigner ces variables en cours d'exécution d'interpréteur de commande. Votre shell devra aussi implémenter l'expansion d'autres variables initialisées pendant la session, en reconnaissant le symbole d'affectation (exemple : `A=2`) et le symbole d'expansion `$` (exemple :`echo $A`).

2. Navigation du shell dans le système de fichier et création d'alias

Votre interpréteur devra implémenter les commande internes `cd`, `pwd`, `alias` qui respectivement change le répertoire de travail de l'interpréteur, affiche le répertoire de travail courant et assigne un alias à une ligne de commande indiquée en paramètre. Voir la commande `help alias` pour comprendre le fonctionnement de cette commande interne à implémenter.

Une fois la ou les commandes commande terminées, l'invite de commande doit être à

nouveau présentée à l'utilisateur, afin qu'il puisse entrer une autre commande. Il devra être possible de quitter votre interpréteur avec un code de sortie donné avec la commande interne `exit`.

3. Exécution de commandes relatives à la variable d'environnement PATH

Lorsque l'utilisateur entre une commande puis appuie sur entrée, l'interpréteur doit lire la ligne entrée. Si le premier mot de la ligne de commande n'est pas une commande interne, celui-ci doit être interprété comme le nom du programme à exécuter, tandis que le reste de la ligne de commande devra être interprété comme les paramètres à donner au programme. Par exemple, si l'utilisateur entre la ligne `ls -l /tmp`, l'interpréteur devra exécuter le programme `ls` avec les arguments `-l` et `/tmp`. La sortie dans le terminal exécution du programme devra être la même que si le programme `ls` avait été exécutée dans `bash`.

L"interpréteur tentera d'exécuter le programme avec le chemin d'accès indiqué (absolu ou relatif). Si aucun chemin d'accès n'est indiqué, c'est la variable d'environnement PATH qui sera utilisée pour chercher le programme à exécuter.

4. Redirection des entrées/sorties des commandes exécutées

Dans une ligne de commande, l'interpréteur devra reconnaître les symboles spéciaux supérieur "`>`", inférieur "`<`" et pipe "`|`" pour respectivement rediriger la sortie standard de la commande exécutée sur un nom de fichier indiqué à droite du symbole ">", rediriger l'entrée standard de la commande exécutée sur un nom de fichier indiqué à droite du symbole "<", rediriger la sortie standard de la commande exécutée à gauche du symbole "|"dans l'entrée standard de la commande exécutée à droite du symbole "|".

Par exemple avec la ligne de commande suivante : `ls -l /tmp > /tmp/toto.txt` l'interpréteur devra exécuter le programme `ls` avec les arguments `-l` et `/tmp`. La sortie standard de l'exécution de cette commande devra être écrite dans le fichier `/tmp/toto.txt` sans modifier le binaire `ls` mais simplement en faisant appels aux services du système `close(2)`, `open(2)` et `dup(2)`/`dup2(2)`.

Avec la ligne de commande suivante : `ps | wc -l` L'interpréteur devra exécuter `ps` et `wc` en parallèle. La sortie standard du programme `ps` devra être redirigée dans l'entrée standard de la commande `wc` par l'intermédiaire d'un pipe, au faisant appel au service du système `pipe(2)`

Pour simplifier, une seule seule redirection d'entrée/sortie par ligne de commande interprétée est obligatoire dans ce projet. Il n'est pas obligatoire d'implémenter plusieurs redirections par ligne de commande comme par exemple dans la commande suivante : `ps | wc > /tmp/res.txt`

5. Lecture d'une suite de commande depuis un fichier.

Il devra être possible d'exécuter votre interpréteur avec un fichier en argument ou redirigé sur son entrée standard.

Votre interpréteur devrait exécuter séquentiellement chaque commande apparaissant dans le fichier, qu'elle soit séparées par un retour à la ligne (\n) ou par un point virgule (;).

Ainsi si votre programme s'appelle monbash, on pourra l'exécuter des deux manières suivante pour qu'il lise le contenu de script.sh et qu'il en exécute séquentiellement le contenu :

```
./monbash script.sh
./monbash < script.sh
```

Modalités de rendu

Le projet est à réaliser par groupe de 2 personnes. L'url gitlab de votre projet devra être communiquée au professeur avant le 1/12/2017 sous peine de pénalité dans la notation.

Les points 1. et 2. du cahier des charges sont à rendre sous la forme d'un rendu intermédiaire contenant un rapport de 5 pages et l'état actuel de l'implémentation la semaine avant les vacances de Noël. Le rapport devra être disponible sur l'adresse du dépôt gitlab fournie au professeur.

Le rendu final devra implémenter l'ensemble du cahier des charges est à rendre au plus tard le 20/1/2018. Il comprendra un rapport de maximum **10 pages** décrivant votre implémentation ainsi que les responsabilités de chacun dans la réalisation du projet. Votre code C devra être **obligatoirement** compilable via un fichier `Makefile` sous GNU/Linux et `gcc`.

Annexe : fonctions utiles pour accomplir le projet

- `getpwuid(3)` et `getuid(2)` pour récupérer des informations relatives aux utilisateurs.
- `fopen(3)` et `fread(3)` pour ouvrir un fichier et en lire une ligne.
- `setenv(3)`, `getenv(3)` et `putenv(3)` pour lire ou écrire des variables d'environnement.
- `open(2)` et `close(2)` pour respectivement ouvrir, fermer un fichier.
- `dup(2)` pour dupliquer un descripteur de fichier et rediriger les entrées sorties.
- `strcmp(3)` `strtok(3)` pour effectuer des traitement de chaînes de caractères
- L'appel système `fork(2)` pour créer le processus exécutant la commande interprétée.
- L'appel système `execve(2)` pour exécuter un programme avec arguments.
- `wait(2)` ou `waitpid(2)` pour attendre la fin de l'exécution d'un processus fils
- `getcwd(3)` et `chdir(2)` pour obtenir le répertoire de travail courant ou bien le changer.

Ces fonctions sont toutes documentées dans les pages de manuel des sections indiquées entre parenthèse, c'est à dire les appels système (section 2) et les appels aux bibliothèques (section 3).