Projet Interface 2037

Sovannara Hak (hak@isir.upmc.fr)

Généralités

Le but de ce projet est de programmer en C un interpréteur de commande basique ainsi que ses commandes associées ayant un comportement similaire aux terminaux futuristes des films des années 70-80 (on prendra l'exemple de l'Interface 2037 du MU-TH-R 182 du film Alien).

L'interpréteur de commande (souvent appelé *shell* en anglais) est le programme qui est lancé lorsque vous ouvrez un nouveau terminal. Il permet entre-autres d'exécuter les programmes disponibles sur votre système, par exemple [1] :

```
1 > ls -l -a
```

On vous demande un minimum de travail d'analyse et d'autonomie : c'est à vous de choisir comment organiser votre programme et votre code pour qu'il soit le plus simple et le plus élégant possible. Néanmoins, il est fortement suggéré de suivre l'ordre d'implémentation suggéré par le sujet.

Remarques importantes

- Il est indispensable de gérer correctement les cas d'erreur (commande introuvable, erreur de syntaxe, etc.) ; c'est souvent la partie la plus difficile de chaque question !
- Sauvegardez ou versionnez. Faites une sauvegarde complète de votre code au moins à chaque fois que vous avez fini et testé une question (ou utiliser un logiciel de gestion de version comme `git`).
- Dans une commande shell, le nombre d'espaces ne compte pas.

Quelques critères de notations

- Les fuites mémoires seront pénalisées (vérifiez avec Valgrind).
- Les erreurs détectées par Valgrind seront pénalisées (variable non initialisée, lecture invalide, etc.).
- L'absence de warning à la compilation.
- Faites un programme robuste : toute erreur de segmentation impactera fortement votre note finale.
- L'utilisation de variables globales sera pénalisée.
- La propreté de votre code (commentaires, indentation, organisation, ...) sera prise en compte dans la notation.
- Pensez à la gestion des erreurs (avec *errno.h* par exemple).
- Un programme incomplet mais fonctionnel est préférable à un programme ne compilant pas ou cassé (sauvegardez ou versionnez).

1. Dans la suite de ce document, le symbole « > » des exemples correspond à l'*invite de commande* (prompt) de votre programme, et « $ » celui de votre shell. Il s'agit d'une simple chaîne de caractères (variant suivant les interpréteurs utilisés) qui indique que l'interpréteur est prêt à accepter des commandes.

Rendu du projet

- Vous devez envoyer une archive tar.gz contenant :
  - votre code source ;
  - un fichier texte appelé README expliquant rapidement les questions traitées et non traitées, ce que font les fonctions si vous ne l'avez pas mis dans le code et vos choix d'implémentations.
- Le code source doit contenir un fichier Makefile permettant de compiler votre projet, et éventuellement vos programmes de tests de base.
- N'hésitez pas à ajouter des fichiers secondaires contenant une fonction `main` pour tester vos fonctions (par exemple `test/test_fonction1.c`).
- La dernière version de votre projet devra être envoyé par mail à `hak@isir.upmc.fr` le 19/01/2015.

1 Fonctions de base

1.1 Commandes sans arguments

Implémentez en C les commandes de base suivantes :
- `list` : affiche la liste des fichiers et dossiers présent dans le répertoire courant.
- `pwd` : affiche le chemin vers le répertoire courant
- `help` : affiche la liste des commandes disponibles

Implémentez ensuite l'interpréteur de commande. Il s'agit d'un programme qui va lire sur l'entrée standard des chaines de caractères, et va exécuter la fonction associée. On indiquera à l'utilisateur que le programme attend une entrée en affichant un prompt (« > » par exemple). Les commandes reconnues pour le moment sont : `list`, `pwd`, `help` et `exit`. Votre programme doit quitter lorsque l'utilisateur tape la commande `exit`. Pour le reste du projet, vous afficherez le message `Unable to compute` suivit d'une explication courte en cas d'erreur (si la chaine de caractères entrée ne correspond pas à une commande connue...)

Attention, comme dans un shell normal, vous devez attendre la fin de l'exécution de la commande avant de re-afficher l'invite de commande (*prompt*) ;

Exemple :

```
1  >list
2  philip_j_fry
3  ellen_ripley
4  parker
5  ash
6  >pwd
7  working directory: /home/shak/interface2037/profiles
8  >help
9  available commands:
10  list
11  pwd
12  help
13  exit
14 >
```

Fonctions utiles `fgets()`, `opendir()`, `stat()`, `strcmp()`, `readdir()`, `malloc()`, `free()`,`fflush()`,`strerror()`,`getcwd()` ...

1.2 Commandes avec des arguments

On souhaite maintenant pouvoir donner des arguments aux commandes lancées. Pour implémenter ce comportement, vous pouvez utiliser la fonction `strtok_r()` qui permet de découper une chaine de caractères en fonction de *délimiteurs*.

Implémentez les fonctions suivantes :

- `open fichier` : lit `fichier` et l'affiche sur la sortie standard (comme la commande `cat` dans un vrai terminal)
- `cd dossier` : change de répertoire courant
- `echo un texte à afficher` : affiche toute la chaine de caractères passée après `echo`.
- `head n fichier` : affiche au plus les n premières lignes du fichier.
- `tail m fichier` : affiche au plus les m dernières lignes du fichier (dans l'ordre du fichier original).

Puis ajoutez les aux commandes supporté par votre interpréteur. Il faudra vérifier qu'on donne bien en argument un fichier ou un dossier, vérifier que ces arguments existent...

Modifier et mettez à jour la commande `help` pour accepter un argument optionnel. Sans argument, la commande affiche la liste des commandes disponibles. Si un argument est passé, la description de la commande demandée est affiché.

Exemple :

```
1 >help open
2 usage: open file
3  print file on the standard output
4 >
```

Fonctions utiles `strtok_r()`,`strcat`,`chdir()`,`fopen()`,`fclose()` ...

2 Fonctions avancées

Attention, l'implémentation des fonctions avancées nécessitera probablement une réorganisation de votre code et un peu de réflexion de votre part ; pour être sûrs de pouvoir rendre un programme fonctionnel, sauvegardez précieusement une copie de votre avant les modifications (sauvegardez ou versionnez).

2.1 Affichage futuriste

Ce qu'on appelle *affichage futuriste* est un affichage *lent/progressif* : l'apparition des caractères est suffisemment lente pour avoir un effet de défilement du texte. L'implémentation de la gestion de l'affichage reposera sur deux threads, un de lecture et l'autre d'impression.

- Le thread de lecture lit ce qui doit être affiché, puis remplit un buffer partagé de taille limitée. Dans un premier temps, ce buffer pourra contenir 3 caractères.
- Le thread d'impression lit le contenu du buffer, écrit sur la sortie standard (ou plus généralement sur un flux) la totalité du buffer, puis attend un certain temps avant

de pouvoir de nouveau lire le contenu du buffer. Dans un premier temps, le thread attendra 20ms.

Il faudra synchroniser les threads afin de ne pas lancer la procédure d'impression avant que le buffer ne soit rempli, et de ne pas remplir ou écraser le buffer avant qu'il soit vidé. Tout en évitant bien evidemment les accès concurents.

On s'imposera la contrainte suivante : seul le thread d'impression a le droit d'utiliser `stdout`. Par conséquent, il ne sera pas nécessaire de protéger par mutex `stdout`. Vous aurez peut être besoin de modifier les commandes codées afin de ne pas utiliser stdout directement (sauvegardez ou versionnez).

Fonctions utiles `usleep(),tmpfile()...`

2.2 Fichier de configuration

Ajouter à l'interpréteur la possibilité de charger un fichier de configuration passé en paramètre. Ce fichier de configuration contiendra :

- la taille du buffer,
- le délai d'attente du thread d'impression,
- le caractère (ou chaine) qui représentera le prompt

```
$ ./interface2037 ma_config.txt
Loading configuration file ...
cache size: 2
processing delay: 33ms
prompt: >
>
```

2.3 Finitions

Pour rendre l'interpréteur encore plus futuriste, implémentez les fonctionalités suivantes :

- Affichage en vert
- Mettre tout le texte à afficher en majuscule
- Rendre la commande `open` insensible aux majuscules : `open topsecret` doit pouvoir ouvrir le fichier `TopSecret` (on supposera qu'il n'y aura jamais deux fichiers `TopSecret` et `topsecret` dans le même dossier).
- Ajouter un écran de chargement (Exemple : une *Boot Sequence* affichant des instructions obscures, un *log* de tests validés, une barre de chargement...) avant d'entrer en mode *interactif*
- Une commande `delete` qui demande confirmation puis *efface* l'écran (en écrivant via le thread d'impression des espaces après avoir positionné le curseur sur la première ligne du terminal)
- Intercepter un `ctrl-c` pour ne pas tuer l'interpréteur.

Utilisez les macros suivantes pour vous aider :

```
#define COLOR_GREEN   "\x1b[32m"
#define COLOR_RESET   "\x1b[0m"
#define TERMINAL_HOME   "\x1b[H"
#define CLEAR_SCREEN "\e[1;1H\e[2J"
```

```
5
6  printf(COLOR_GREEN);
7  printf("Ici commence l'affichage vert");
8  printf(COLOR_RESET);
9  printf("Retour affichage normal");
10 printf(CLEAR_SCREEN); //effacer tout le terminal
11 printf(TERMINAL_HOME); //positionner le curseur sur la ligne un du
       terminal
```

Fonctions utiles `toupper()` ...

2.4 Questions bonus

Ajoutez les fonctionnalités manquantes qui pour vous, seraient indispensables (comme trier la sortie de `list`, une commande pour calculer la probabilité de survie). Les bonus ne seront acceptés que si toute les autres questions sont traitées.