Séance de TP N°4
04. microshell – Microshell avec redirections

Créez votre répertoire de TP *~/l3i6/04.microshell* et placez-vous dessus. Récupérez dans votre répertoire de TP le fichier *Fourniture04.tgz* sur le site de cet enseignement[1] et déballez-le. Ce répertoire contient :

- ***Makefile.gen*** et le script ***configure*** pour produire un ***Makefile*** adapté à votre système.
- des fichiers sources fournis : ***msh1.c***, ***msh2.c***, ***mshTools.c***, ***mshTools.h***,
- des fichiers sources à compléter : ***msh3.c***, ***msh1b.c***,
- des scripts pour les tests : ***script1.sh***, ***script.dc***, ***t_msh2.zsh***.

Etude du microshell version 1

Construire l'exécutable **msh1** qui produit un programme Shell minimal. Ce shell reconnait des commandes sans argument, comme **date**, **ls** , **who**... Après chaque exécution d'une commande, l'invite du Shell affiche le numéro du processus qui a servi à exécuter la commande et le code de retour.
Testez ce shell sur des commandes sans argument et avec argument. Le nom de la commande peut être un nom de chemin ou un nom simple de fichier exécutable situé dans l'un des répertoires cités dans la variable d'environnement *PATH*. Etudiez le code source du shell et expliquez l'origine du problème pour les commandes avec argument.

Etude et expérimentation du microshell version 2

Etudiez le code de la version ***msh2.c*** du microshell. Cette version traite les arguments des commandes en utilisant une fonction auxiliaire *mk_vect_cmd* qui rend dans un vecteur de chaînes les différents mots qui constituent la commande (un mot est une suite de caractères non blanc, sans espace ou tabulation). Cette fonction est placée dans le module *mshTools.c*. Compilez cette version et testez là avec différentes commandes à arguments, par exemple *ls -ls*, *cp msh1 truc*, *etc*. Le script ***t_msh2.zsh*** effectue quelques tests de ce shell. Affichez ce script et exécutez-le. Interprétez les résultats.

Réalisation du microshell version 3 avec redirections.

On traite maintenant les redirections des voies standard 0 et 1 pour les commandes du microshell **msh2**. La syntaxe choisie pour ces redirections est d'indiquer à la fin de la commande, sur la même ligne, le signe de redirection entouré d'au moins un blanc, suivi du nom de fichier disque utilisé pour la redirection. Le signe de redirection sera '<' pour la voie 0 et '>' pour la voie 1. Une commande peut rediriger 0, 1 ou 2 voies.

Exemples de commandes pour la version 3

```
% ls -ls msh3.c
%   ls  -ls
% ls -ls $>$ toto
% cat toto
% dc # calculatrice de grande précision, mais en notation polonaise postfixée
3
4
+
p
```

[1] http://deptinfo.unice.fr/~lahire/enseignement/SYSL3

```
q
% dc $<$ script.dc # La même chose, mais avec une redirection de la voie 0
% dc $<$ script.dc $>$ resultat
% cat resultat
% msh3 $<$ script1.sh $>$ resultat
% resultat
```

Modifiez le code source de ***msh3.c*** (identique à ***msh2.c***) pour traiter les redirections demandées. On écrira le code de manière incrémentale, en visant d'abord la redirection de la voie 1 seulement, sans chercher à détecter des erreurs de syntaxe. On testera cette version simplifiée, et si elle donne satisfaction, on traitera la redirection de la voie 0. Enfin, si tout va bien, on rendra cette version robuste en traitant les différents cas d'erreurs possibles (syntaxe, trop de redirection, fichier inexistant, *etc*) de la manière appropriée. Faites de nombreux tests de redirections, y compris en appelant comme commande **msh3** lui-même avec le ***script1.sh*** redirigé sur la voie 0.

Microshell version 1b

Modifiez le code de ***msh1b.c*** (identique à ***msh1.c***) pour que la lecture des lignes se fasse directement par la primitive système *read(2)*. Compilez ce programme est testez le. Cela marche-t-il toujours ? Examinez le fichier de commandes ***script1.sh***. Exécutez ce script avec **msh1** en redirigeant la voie standard 0. Cela marche-t-il?
Faites de même avec **msh1b**. Expliquez le phénomème. Suggestion: la sémantique de *read* et *fgets* n'est pas la même, selon que la voie 0 est associée au clavier ou à un fichier. Même si vous n'avez pas compris, passez à la question suivante, on vous expliquera dans le corrigé...

Pour les plus courageux ...

Améliorations de la version ***msh3.c***. Si vous ne l'avez pas déjà fait, allégez le code de la boucle de lecture des commandes en plaçant dans des routines les sections :

- qui analysent les signes de redirection,
- qui réalisent les redirections,
- qui restaurent les redirections.

Ecrivez ces routines de manière générique pour des redirections quelconques, en choisissant la syntaxe classique des shells : '2>', '3>', '3<', '4>', *etc*.

Pour les très courageux *msh4.c* : shell avec commandes maison et variables

Vérifiez que votre variable *PATH* contient bien votre répertoire courant en tête de la liste des répertoires. S'il le faut, corrigez-là.

Commandes maison (built-in)

Pour éviter de créer un processus et de charger un programme pour chaque commande, les shells reconnaissent un certain nombre de commandes qu'ils traitenet directement. Ecrivez un programme **echo** qui affiche ses arguments. Testez-le avec **msh3**.
Définissez un type de descripteur de commande maison qui contient comme champ le nom d'une commande et l'adresse d'une fonction qui l'exécute en interne. Définissez le répertoire des commandes maison comme un tableau de tels descripteurs. Incorporez dans la routine d'analyse d'une ligne de

commande la reconnaissance des commandes maison et traitez comme il convient ces commandes maison. Testez votre programme avec la commande maison **echo**.

Variables shell

Ajoutez deux commandes maison pour définir une variable shell ou une variable d'environnement et leur valeur, avec la syntaxe :

```
% defvar nomVariable valeur
% defenv nomVariableEnvironnement valeur
```

La différence entre une variable shell et une variable d'environnement, est que cette dernière est exportée et devra être communiquée aux processus fils. Modifiez la syntaxe du shell pour qu'il reconnaisse la notation '$nomVariable' de substitution d'une valeur de variable à un argument de commande. Réalisez un dictionnaire de variables shell (par exemple un tableau de descripteurs struct) qui note le nom, la valeur et le statut exporté ou non de chaque variable. Modifiez l'appel système *exec* pour qu'il exporte au processus fils la liste des variables d'environnement déjà accessibles au shell (via *envp*) et les variables exportées de son dictionnaire. Testez cette version avec des commandes de définition de variables exportées ou non et la commande maison **echo**. Ajoutez les commandes maison **listvar** et **listenv** qui listent les variables shell et celles exportées. Testez le tout...