Processus & tubes

Gabriel Hondet
gabriel.hondet@lsv.fr

22 novembre 2020

Exercice 1 : Fichiers louches.

Téléchargez le fichier `http ://www.lsv.fr/~hondet/resources/archos/touch_weird.sh.x` et exécutez le. Trois fichiers (vides) devraient être apparus dans votre répertoire courant. Supprimez-les en utilisant *uniquement* la ligne de commande.

Indication : comment sont identifiés les fichiers de manière générale ?

Exercice 2 : Programme erroné.

Considérons `http ://www.lsv.fr/~hondet/resources/archos/closed_pipe.c`. Le fils est censé imprimer les caractères que lui envoie le père : expliquez le dysfonctionnement et corrigez le programme.

Exercice 3 : Tubes et remplacement de code.

Écrivez un programme (en C) qui télécharge l'archive `http ://www.lsv.fr/~hondet/resources/archos/shell-bootstrap.tar.gz`, et la décompresse sans créer de fichier temporaire. Dit autrement, on veut coder la commande `curl <url> | tar xz` en C. Les programmes `curl` et `tar` seront appelés par `exec` ou dérivé.

Exercice 4 : La fonction d'Hénon, partie I.

On va calculer l'orbite d'un système dynamique de dimension 2. La fonction d'Hénon est définie par le système

$$H_{a,b} = \begin{cases} x_{n+1} = a - by_n - x_n^2 \\ y_{n+1} = x_n. \end{cases} \tag{1}$$

On utilisera un processus pour calculer la suite $(x_n)_n$ et un autre processus pour calculer la suite $(y_n)_n$. Les processus échangeront leurs données via un (ou des) tube(s).

Par la suite, on créera en plus un processus dédié à la sortie : ce processus doit écrire des lignes sous la forme `0.3415 1.2451` où le premier nombre est x_n et le deuxième y_n dans un fichier `henon.dat`.

On pourra tracer la fonction avec la commande `gnuplot henon.p` après avoir téléchargé le script `http ://www.lsv.fr/~hondet/resources/archos/henon.p`. Le fichier `henon.dat` doit être dans le même dossier que `henon.p`.

Pour $a = 1\ .4$ et $b = -0\ .3$, la fonction est chaotique (au sens de Devaney), c'est-à-dire,

- la fonction est sensiblement dépendante des conditions initiales (effet papillon, imprédictabilité) ;
- la fonction est topologiquement transitive (indécomposabilité) ;
- les points périodiques sont denses dans le domaine de définition de $H_{a,b}$ (régularité).

et possède un attracteur étrange.

Pour plus d'informations sur les systèmes dynamiques, voir *An introduction to Chaotic Dynamical Systems*, Robert L. Devaney, Westview.

Exercice 5 : Coquille vide.

Vous trouverez le squelette de base du code C que l'on va utiliser pour recoder un shell. Pour compiler le projet, utilisez la commande `make`. Par défault, le shell ne peut pas faire grand chose. On va essayer de le compléter pas à pas. On va d'abord s'intéresser à la fonction `execute` :

Le cas de base, correspond au cas `C_PLAIN`. Donner un exemple de commande qui une fois parsée retourne un objet `cmd` tel que `cmd->type == C_PLAIN`. En utilisant `ps`, observer ce qui se passe dans un terminal lorsque vous lancez une commande.

Pour le moment, toute commande de base sera tout simplement exécutée. Pour exécuter une commande, la librairie `glibc` offre un panel de fonctions dont on peut avoir un aperçu en utilisant la commande

```
man exec
```

Selon vous, quelle fonction serait la plus appropriée dans notre cas (justifier) ? En utilisant toutes ces observations, remplir le trou `C_PLAIN`.

Quel est le symbol pour l'opérateur de séquence en bash ? Donner un exemple de commande où la séquence se comporte différemment de l'opérateur *et* logique.

Implémenter le cas `C_SEQ`.

Implémenter les cas `C_AND` et `C_OR`.

Il est possible en bash d'écrire une commande de la forme (cmd1 && cmd2 | cmd3 ...) 2>/dev/null

Quel est le rôle des parenthèses dans la commande ci-dessus ? Donner un exemple d'une commande qui utilise (de façon non triviale) ces parenthèses. Implémenter le cas `C_VOID`.

Que se passe-t-il si vous faites `CTRL+C` dans notre shell ? Proposer une solution pour récupérer la main après que l'utilisateur a entré `CTRL+C`.

Que se passe-t-il lorsque que vous lancez la commande

```
ls > dump
```

Pour corriger ce problème, je vous invite à lire les pages de manuel `man stdin` et `man dup`. En utilisant toutes ces informations, implémenter la fonction `apply_redirections` puis modifier votre implémentation pour que la commande ci-dessus se comporte comme prévu.

Il nous reste finalement à implémenter le cas `C_PIPE`, pour cela je vous invite à regarder le manuel de `man pipe`. Donner un exemple qui met en évidence pourquoi on ne peut pas simplement utiliser `dup2` pour réimplémenter le pipe ? En utilisant la fonction `pipe`, réimplémenter le cas `C_PIPE`.

À ce stade, nous avons implémenté un *shell* très rudimentaire, cependant il est possible de l'étendre de bien des manières. Voici quelques possibilités d'extensions qui peuvent vous rapporter des points bonus :

- Réimplémenter les commandes `ls`, `cat` ou `cd`
- Implémenter l'extension des wildcards : `ls *.pdf`
- Implémenter les processus de fond via les commandes : `jobs`, `bg`, `fg`, …