

Projet « Systèmes et Compilation »

Université Paris 7 — IUP 1

Novembre 97

Ce projet est à réaliser par groupe de deux ou trois étudiants. Il fera l'objet d'une soutenance. Un rapport court (typiquement trois à cinq pages), synthétique, illustré d'exemples devra être fourni. Le but du projet est d'écrire en C, au moyen de `lex` et `yacc`, un interpréteur de commandes (un *shell*). Mis à part pour les expressions, la signification des différents opérateurs et commandes internes est la même qu'en `tcsh`. Par ailleurs, il n'est pas demandé que le *shell* réalise la complétion de nom de fichier.

Le *shell* doit utiliser, au moins en ce qui concerne la variable `PATH`, un environnement que l'utilisateur pourra changer (par la commande interne `setenv`). Il permet de lancer des commandes internes (connues par lui et directement interprétées, sans exécuter de binaire) ou externes (lancées par exemple via un `exec`). Les commandes externes correspondent soit à des fichiers binaires soit à des *shell-scripts*. Le *shell* doit aussi permettre l'interprétation directe de *scripts* (sans lancer de sous-shell), via la commande interne `source`. Enfin, il doit pouvoir synchroniser les processus lancés, et fournir des moyens de communication via des redirections (`<`, `>` ou `>>`) ou des tubes (`|`).

La fonction de librairie `system` est bien sûr interdite. Il s'agit d'écrire soi-même un *shell* (donc en utilisant des appels comme `fork`, `execlp`, ou `dup`), et non de faire interpréter par un shell ce qui est entré par l'utilisateur.

La grammaire suivante décrit la forme des instructions à traiter (ε désigne le mot vide). La grammaire ne détaille pas ce qu'est une expression entière. D'autre part, certains lexèmes sont laissés à définir (*identificateur* ou *entier*, nécessaire pour *expression-entière* par exemple). Les lexèmes sont notés en fonte *courier*, les non-terminaux en *italique*.

```
instruction →  $\varepsilon$ 
            | commande liste-arguments
            | instruction ; instruction
            | instruction || instruction
            | instruction && instruction
            | (instruction)
            | instruction &
            | instruction | instruction
            | instruction > identificateur
            | instruction < identificateur
            | instruction >> identificateur
commande → identificateur
            | if { test } then commande
            | if { test } then commande else commande
liste-arguments →  $\varepsilon$ 
                  | liste-arguments identificateur
test → expression-entière comparaison expression-entière
        | expression-chaîne == expression-chaîne
        | expression-chaîne != expression-chaîne
comparaison → == | != | < | > | <= | >=
expression-chaîne → identificateur
                    | $identificateur
```

En plus de la commande `if`, il devra y avoir comme commandes internes les suivantes: `alias`, `builtins`, `cd`, `echo`, `exit`, `kill`, `nohup`, `printenv`, `setenv`, `source`, `unalias`, et `unsetenv`, avec la même signification qu'en `tcsh`.

En fonction du temps restant, il sera bien sûr possible de réaliser quelques extensions...