

Programmation d'un Interprète Shell

travail individuel à rendre sous moodle

Présentation

Un embryon de shell vous est fourni dans le répertoire `fork-exec`. Le programme `Shell.c` est, en l'état, capable d'analyser les lignes de commande qui lui sont soumises et d'afficher le résultat de son analyse. Pour réaliser cette analyse ce programme utilise une fonction d'*analyse syntaxique* qui renvoie l'arbre syntaxique associé à la ligne de commande¹. C'est un arbre binaire qui décrit comment sont combinées les commandes contenues dans la ligne de commande donnée. Voici la définition de la structure de données utilisée :

```
typedef enum expr_t {  
    VIDE,           // Commande vide  
    SIMPLE,         // Commande simple  
    SEQUENCE,       // Séquence (;)  
    SEQUENCE_ET,    // Séquence conditionnelle (&&)  
    SEQUENCE_OU,    // Séquence conditionnelle (||)  
    BG,             // Tache en arriere plan  
    PIPE,           // Pipe  
    REDIRECTION_I,  // Redirection entree  
    REDIRECTION_O,  // Redirection sortie standard  
    REDIRECTION_A,  // Redirection sortie standard, mode append  
    REDIRECTION_E,  // Redirection sortie erreur  
    REDIRECTION_EO, // Redirection sorties erreur et standard  
} expr_t;  
  
typedef struct Expression {  
    expr_t type;  
    struct Expression *gauche;  
    struct Expression *droite;  
    char **arguments;  
} Expression;
```

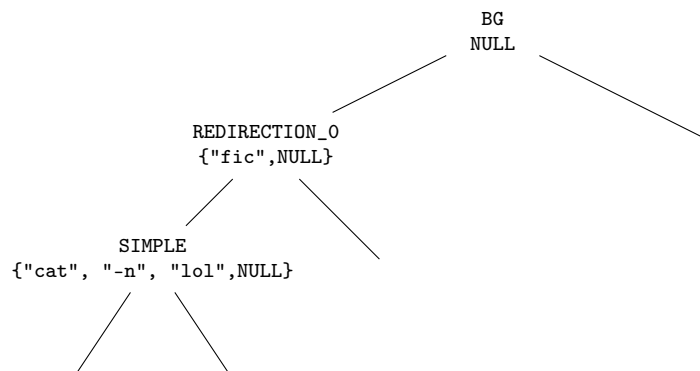


FIGURE 1 – Arbre associé à la commande `cat -n lol > fic &`

Compilez le programme et testez-le en entrant la commande `./Shell`. Consultez la fonction `afficher_expr()` pour voir comment on explore récursivement la structure de donnée associée à une expression.

1. NB. cette fonction retourne `NULL` en cas de ligne syntaxiquement incorrecte.

Plan de travail

Modifier à votre guise le module `Evaluation.c` afin de :

1. exécuter la commande interne `echo` qui affiche ses arguments ;
2. exécuter une commande externe simple (telle que `ls -al`) ;
3. évaluer une séquence de commandes (telle que `ls -al ; cat -n lol`) puis avec les opérateurs `&&` et `||` ;
4. mettre en arrière plan une commande (telle que `xeyes &`) - on proposera une méthode pour éliminer les zombies à chaque appel de la fonction `evaluer_expr()` ;
5. effectuer une redirection de la sortie standard (telle que `ls -al > output`) ou de l'entrée standard (telle que `cat -n < input`) ;
6. effectuer les redirections simultanées de la sortie standard et de l'entrée standard (telle que `cat -n < input > output`) ;
7. mettre en place toutes les redirections vers les fichiers ;
8. exécuter la commande interne `source` ;
9. évaluer tout pipeline de deux commandes (tel que `ls -al | cat -n`) ;
10. évaluer des pipelines de plusieurs commandes ;
11. mettre en place l'élimination des zombies en utilisant le traitement des signaux.

Naturellement on pourra enrichir son travail en ajoutant des commandes internes (telle que `cd,...`) ou encore des traitements plus ambitieux comme le traitement des jobs.