Polytech Paris-Sud Systèmes
Année 2012-2013

TP6 Micro-shell

Le but de cet exercice est la réalisation d'un petit shell. Notre shell sera une simple boucle d'interaction qui affiche une invite de commande `-->`, lit une ligne entrée au clavier (`fgets(3)`), l'exécute, et recommence. Le shell quitte quand l'utilisateur tape `Ctrl-D` à la place d'une commande.

Un squelette de programme est disponible :

`https://www.lri.fr/~mandel/systemes/tp-06.tgz`

Question 1 *Dans un premier temps, on suppose que la commande est simplement un nom de programme à exécuter dans un processus séparé. Le shell doit attendre (*`waitpid(2)`*) que le programme se termine avant de rendre la main à l'utilisateur.*

On propose une fonction `decoupe` permettant de découper une ligne en mots :
- L'entrée de `decoupe` est la chaîne à découper, contenue dans le paramètre `ligne`.
- La sortie de `decoupe` sera une liste de pointeurs vers le début de chaque mot. La liste est stockée dans le tableau `elems` de `maxelems` pointeurs et est terminée par le pointeur `NULL`.

Voici le code source :

```
#include <ctype.h>

void decoupe(char *ligne, char** elems, int maxelems) {
  char* debut = ligne;
  int i;
  for (i=0; i<maxelems-1; i++) {
    /* saute les espaces */
    while (*debut && isspace(*debut)) debut++;
    /* fin de ligne ? */
    if (!*debut) break;
    /* on se souvient du début de ce mot */
    elems[i] = debut;
    /* cherche la fin du mot */
    while (*debut && !isspace(*debut)) debut++; /* saute le mot */
    /* termine le mot par un \0 et passe au suivant */
    if (*debut) { *debut = 0; debut++; }
  }
  elems[i] = NULL;
}
```

Lors de son parcours de `ligne`, la fonction `decoupe` transforme les espaces et retours à la ligne (voir `isspace(3)`) en fin de mots en caractère `\0` et fait pointer `elems` à l'intérieur de `ligne`. L'avantage de cette méthode est qu'il n'est pas nécessaire d'allouer de la mémoire supplémentaire pour stocker les mots isolés de `ligne`.

Question 2 *On suppose maintenant que la ligne de commande peut contenir un nombre arbitraire d'arguments, séparés par un ou plusieurs espaces.*

Question 3 *Que se passe-t-il lorsque vous tapez la séquence de commandes suivantes :*

```
--> cd /tmp
--> pwd
--> ls ~
```

Question 4 *Modifiez votre shell pour que, si une commande est en cours d'exécution, un appui sur* `Ctrl-C` *interrompe la commande et redonne la main à l'utilisateur sans toutefois quitter le shell. On utilisera pour cela* `sigaction(2)` *avec* `SIG_IGN` *et* `SIG_DFL`*.*

On propose maintenant une fonction `decoupe2` qui permet de séparer un tableau de la forme du tableau `elems` précédent en deux parties selon le séparateur `sep`.

- Les paramètres sont le tableau `elems` de chaînes de caractères et la chaîne de caractère `sep`.
- La valeur de retour est l'adresse du tableau qui se trouve après le séparateur.

Voici le code source :

```
char** decoupe2(char** elems, char* sep) {
  char** debut = elems;
  char** res = NULL;
  while (NULL != *debut) {
    if ( 0 == strcmp(*debut, sep) ) {
      res = debut + 1;
      *debut = NULL;
      return res;
    }
    debut++;
  }
  return res;
}
```

Question 5 *Ajoutez à votre shell la redirection de la sortie standard. On utilisera la fonction* `decoupe2` *avec* `>` *comme séparateur.*

Question 6 *Ajoutez à votre shell les opérateurs de composition* `&&` *et* `||`*. On supposera qu'il y a en même temps au plus un opérateur ou une redirection.*

Question 7 *Ajoutez à votre shell la création d'un tube anonyme.*

Question 8 *Modifiez votre shell pour que toutes les commandes soient lancées en tâche de fond (on ne veut pas de processus zombies). Que se passe-t-il si une commande en tâche de fond essaye de lire l'entrée standard (e.g.,* `cat`*) ?*