Examen

Durée : 2 heures
Documents de cours, TD et TP autorisés

1 Question de cours

Question 1 *Quels sont les comportements des programmes JoCaml suivants :*

1. ```
   def f1(x, y) = (print_endline x; 0) & (print_endline y; 0)
   let main = spawn (f1 ("1", "2"))
   ```
2. ```
   def f2(x) & f3(y) = print_endline x; 0
   let main =
     spawn (f2 "1");
     spawn (f2 "2");
     spawn (f3 "3")
   ```
3. ```
   def f4(x) & f5(y) = print_endline y; 0
    or f4(x) & f6(y) = print_endline y; 0
   let main =
     spawn (f4 "1");
     spawn (f5 "2");
     spawn (f6 "3")
   ```

2 du – Disk Usage

La page de manuel de la commande `du` commence de la façon suivante :

```
du affiche la quantité d'espace disque utilisée par chacun des arguments, et
pour chaque sous-répertoire des répertoires indiqués en argument.  Toutes les
mesures contiennent l'ensemble de la hiérarchie des sous-répertoires dans le
système de fichiers.  Sans argument, du mesure l'espace occupé par le
répertoire courant et ses descendants.

L'espace est mesure en blocs d'un Ko par défaut, à moins que la variable
d'environnement POSIXLY_CORRECT soit configurée, auquel cas les blocs
représentent 512 octets.
```

Voici un exemple de l'invocation de la commande `du` :

```
--> du cours-01 cours-02
104 cours-01/exemples
22560 cours-01/Fig
35912 cours-01
136 cours-02/exemples
672 cours-02/Fig
2728 cours-02
```
```

Le but de cet exercice est d'écrire en C un programme qui a le même comportement que cette commande.

Question 2 *Donner le code d'une fonction*

```
off_t size(char *file);
```

qui calcule la taille du fichier `file`*. En cas d'erreur la fonction doit retourner la valeur* `0`*.*

Question 3 *Donner le code d'une fonction*

```
void du(char *file);
```

qui a le même comportement que la commande `du file`*.*

Question 4 *Donner la définition du programme principal.*

Question 5 *Modifier votre code pour que la commande* `du` *suive maintenant les liens symboliques.*

Question 6 *Quel est le comportement de votre programme vis-à-vis des liens physiques vers un même fichier ? Quel autre comportement pourrait être désiré ? Expliquer en quelques mots comment l'implanter.*

Question 7 *Programmer maintenant une version qui crée un nouveau thread pour le parcours de chaque sous-répertoire.*

Question 8 *Le nombre de threads qui peuvent être crées par le système est limité. Que se passe-t-il si le nombre de sous-répertoires est supérieur à cette limite ?*

Proposer une implantation qui borne le nombre de threads créés à une constante `N`*.*

3 Opérateur strict

En shell, l'opérateur de conjonction booléenne `&&` est un opérateur paresseux : il exécute d'abord son argument de gauche puis son argument de droite seulement si la première évaluation a été un succès. Nous voulons ici programmer en C une commande `and` qui va implanter une version stricte de l'opérateur `&&` : les deux arguments sont toujours exécutés.

Si on exécute la commande `and cmd1 cmd2`, les deux commandes `cmd1` et `cmd2` sont exécutées en parallèle. L'exécution de `and` termine lorsque `cmd1` et `cmd2` ont terminé leur exécution. Le code de retour de `and` est la conjonction des codes de retour de `cmd1` et `cmd2`.

Dans l'exemple suivant, le message `OK` est affiché et la commande `and` termine sur un code d'échec.

```
--> and false "echo OK"
```

Question 9 *Implanter une première version de la commande* `and` *qui attend exactement deux arguments. Vous pouvez utiliser la fonction* `decoupe` *rappelée en annexe pour séparer le nom de la commande et ses arguments dans les arguments de* `and`*.*

Question 10 *Implanter une seconde version de la commande* `and` *qui peut traiter un nombre arbitraire d'arguments.*

Question 11 *Ajouter une option* `-k` *qui, lorsqu'elle est présente, tue tous les processus encore en cours d'exécution dès qu'un des processus termine sur un échec.*

A La fonction decoupe

```
/* découpe ligne en mots
   fait pointer chaque elems[i] sur un mot différent
   elems se termine par NULL
 */
void decoupe(char *ligne, char** elems, int maxelems) {
  int i;
  char* debut = ligne;
  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;
}
```