Programmation des Systèmes

Devoir surveillé intermédiaire

17 octobre 2014

Durée : 1h15
Documents autorisés

Vous expliquerez votre code de sorte que votre intention (et pas seulement votre implémentation) soit claire. La gestion des erreurs pourra rester très simple, par exemple en interrompant le programme si nécessaire.

1 Questions de cours

Q 1. Donnez les définitions de « chemin absolu » et de « chemin relatif ».

Q 2. Donnez la définition de « i-nœud ». Listez en particulier quelques informations qui sont attachées à l'i-nœud, quelques informations qui n'y sont pas attachées. Expliquez la suppression d'un i-nœud.

2 Recherche de liens symboliques

Voici la trace d'une session exécutée dans le répertoire `/tmp/exemple` :

```
$ ls -al
total 20
drwxr-xr-x  3 ray  mond   4096 oct  1 16:45 .
drwxrwxrwt 15 root root  12288 oct  1 16:45 ..
-rw-r--r--  1 ray  mond      0 oct  1 16:44 fich
lrwxrwxrwx  1 ray  mond      4 oct  1 16:44 lien -> fich
lrwxrwxrwx  1 ray  mond      6 oct  1 16:44 lien2 -> cassé
lrwxrwxrwx  1 ray  mond     17 oct  1 16:44 lien3 -> /tmp/exemple/fich
drwxr-xr-x  2 ray  mond   4096 oct  1 16:45 rép

$ ls -al rép
total 8
drwxr-xr-x 2 ray mond 4096 oct  1 16:45 .
drwxr-xr-x 3 ray mond 4096 oct  1 16:45 ..
-rw-r--r-- 1 ray mond    0 oct  1 16:45 fich_bis
lrwxrwxrwx 1 ray mond    7 oct  1 16:45 lien_bis -> ../fich
```

L'objet de cet exercice est d'écrire une fonction `liens_symboliques` qui affiche tous les liens symboliques dans une arborescence. Ainsi `liens_symboliques("/tmp/exemple")` affichera :

```
/tmp/exemple/lien -> fich
/tmp/exemple/lien2 -> cassé
/tmp/exemple/lien3 -> /tmp/exemple/fich
/tmp/exemple/rép/lien_bis -> ../fich
```

Q 3. Donnez en français, en les numérotant, les étapes à réaliser pour la fonction `liens_symboliques`.

Q 4. Proposez une fonction

```
void affiche_lien(const char *chemin)
```

qui, étant donné un chemin correspondant à un lien symbolique, affiche le chemin, une flèche et la cible.
Ainsi, dans l'exemple ci-dessus,

```
affiche_lien("/tmp/exemple/lien");
```

affichera la ligne

```
/tmp/exemple/lien -> fich
```

Q 5. Proposez une fonction récursive

```
void liens_symboliques(const char *chemin)
```

qui, pour un nom de chemin
— affiche le lien, s'il s'agit d'un lien symbolique,
— traite toutes ses entrées récursivement de la même manière s'il s'agit d'un répertoire.

3 Relativité générale

Les liens symboliques peuvent être facilement « cassés » (c'est-à-dire ne plus pointer vers un fichier) si les fichiers sont déplacés. Les liens symboliques utilisant un chemin absolu comme cible sont particulièrement sensibles : ainsi, dans l'exemple initial, en déplaçant le répertoire `/tmp/exemple` dans `/home/ray/test`, on casse `lien3`.

Pour essayer de mitiger un peu ce problème, on se propose de réécrire les liens symboliques absolus en liens relatifs.

3.1 Première version

Une façon simple de transformer un chemin absolu en un chemin relatif est de lui rajouter en préfixe un chemin menant à la racine.

Q 6. Proposez une fonction

```
void affiche_relatif(const char *chemin, const char *chemin_racine)
```

qui, pour un `chemin` et un chemin relatif qui mène de `chemin` à la racine :
— si `chemin` est un lien symbolique absolu, affiche le chemin relatif correspondant,
— si `chemin` est un répertoire, traite toutes ses entrées récursivement.
Sur l'exemple initial, `affiche_relatif("/tmp/exemple", "..")` affichera juste :

```
/tmp/exemple/lien3 -> ../../tmp/exemple/fich
```

Vous vous contenterez d'indiquer les différences avec la fonction `liens_symboliques`.

Q 7. Modifiez la fonction `affiche_relatif` en une fonction `relativise_absolus` qui modifie la cible des liens symboliques absolus pour en faire des liens relatifs.
Vous utiliserez pour cela les fonctions `symlink` et `unlink`.

Pour les dernières fonctions à écrire, vous commencerez par soigneusement décrire en français leurs étapes de calcul.

Q 8. Proposez une fonction

```
char *chemin_racine(const char *chemin_origine)
```

qui retourne un chemin de la forme « `../../..`, etc. » menant de `chemin_origine` à la racine quand `chemin_origine` est un chemin absolu.

3.2 Seconde version

La première version fonctionne bien si on déplace toute l'arborescence (par exemple en montant la partition dans un sous-répertoire), mais ne permet pas encore de déplacer `/tmp/exemple` dans `/home/ray/test`. Dans cet exemple, on voudrait que `lien3` pointe tout simplement vers `fich`, puisqu'il est dans le même répertoire.

Q 9. Proposez une fonction

```
char *court_relatif(const char *depuis, const char *vers)
```

qui retourne le chemin relatif le plus court pointant vers la cible « `vers` » depuis le répertoire « `depuis` ».
« `depuis` » et « `vers` » seront donnés par des chemins absolus.