R3_05 - Programmation système - **1h40**

Formulaire A4 manuscrit recto-verso autorisé

1. **[5 pts]** On dispose de 3 caches de taille 8 octets, C_1, C_2, C_3, contenant respectivement des lignes de 1, 2 et 4 octets. On suppose qu'un défaut de cache coûte 25 cycles, C_1 a un temps d'accès de 2 cycles, C_2 de 3 cycles, et C_3 de 5 cycles. On accède séquentiellement aux adresses de la première colonne du tableau, qu'on vous demande de compléter. L'unité d'adressage est un octet. (h : hit, m : miss)

	C_1		C_2		C_2	
Adresse (binaire)	n° ligne	h/m	n° ligne	h/m	n° ligne	h/m
1 (00000001)						
134 (10000110)						
212 (11010100)						
1 (00000001)						
135 (10000111)						
213 (11010101)						
162 (10100010)						
161 (10100001)						
2 (00000010)						
44 (00101100)						
41 (00101001)						
221 (11011101)						

— Combien y'a-t-il de defauts pour les 3 caches ?
— Quels sont les temps d'accès (en cycle) pour les 3 caches ?

2. **[4 pts]** Soit la mémoire cache du niveau 1 caractérisée par :

```
number_of_sets  64
ways_of_associativity  8
coherency_line_size 64 (octets)
```

Considérons le fragment de code en C suivant :

```
unsigned long long int x,y,z; // size == 8 octets

x=x+y;
y=y+z;
z++;
```

Les adresses ont été attribuées aux variables comme suit :

```
addr(x)=0x007fff80c09ea8
addr(y)=0x007fff80c09fb0
addr(z)=0x007fff80c0afa0
```

(a) Quelle est la capacité du cache ?
(b) Quelles entrées du cache correspondent aux adresses des variables `x,y,z` ?
(c) Combien d'accès au cache effectue-t-on pendant l'exécution du code ci-dessus ? Combien d'accès avec succès (*hit*), combien avec échec (*miss*) ?

3. **[4 pts]** Soit une image `image.ext3` d'un système des fichiers de type `ext3`. La commande `dumpe2fs image.ext3` donne (entre autres) :

```
Filesystem OS type:       Linux
Inode count:              256
Block count:              2048
Free blocks:              944
Free inodes:              241
Block size:               1024
Inode size:               128
```

Le système est monté sur `/mnt/aux`. La commande `ls -l /mnt/aux/` donne :

```
drwxr-xr-x 2 dm    staff  1024 Mar 25 13:29 archives/
drwxr-xr-x 2 dm    staff  1024 Mar 25 13:30 bin/
drwxr-xr-x 2 dm    staff  1024 Mar 25 13:29 home/
drwx------ 2 root  root  12288 Mar 25 13:28 lost+found/
-rw-r--r-- 1 dm    staff 13312 Mar 25 13:38 trash.dat
```

On supprime le fichier `trash.dat` avec la commande

```
rm /mnt/aux/trash.dat
```

et on démonte le système des fichiers.

Qu'affiche maintenant la commande `dumpe2fs image.ext3` ?

4. **[3 pts]** Le système de fichiers qu'on utilise sur une machine est `Ext3`. Voici une suite de trois commandes consécutives :

```
dm@portable:~/OS/Exos/filemanip$ ls -l
drwxr-xr-x    2 dm    staff        4096 Dec 11 05:42 Bar/
-rw-r--r--    1 dm    staff           1 Dec 11 05:38 foo

dm@portable:~/OS/Exos/filemanip$ mv foo Bar/

dm@portable:~/OS/Exos/filemanip$ ls -l
drwxr-xr-x    2 dm    staff        8192 Dec 11 05:43 Bar/
```

Expliquez ce qu'il s'est passé avec le répertoire `Bar`.

5. **[5 pts]** Vous avez ci-dessous 6 codes "fragmentaires" en C. Chaque code contient un paramètre `k`. Analysez chacun et répondez aux questions suivantes :

   (a) Combien de processus en fonction de `k` engendre ce code ?

   (b) Combien d'entre eux seront morts ?

   (c) Combien vont devenir zombies ?

   (d) Combien vont tourner éternellement ?

   (e) Dessinez le graphe (comme vu en td) pour `k=3`.

`ff-1.c`

```
for (i=0;i<k;i++) {
  x=fork();
  if (x==0) break;
}
while (1) sleep(2);
```

`ff-2.c`

```
for (i=0;i<k;i++) {
  x=fork();
  if (x>0) break;
}
while (1) sleep(2);
```

`ff-3.c`

```
for (i=0;i<k;i++) {
  x=fork();
  if (x==0) exit(0);
}
while (1) sleep(2);
```

`ff-4.c`

```
for (i=0;i<k;i++) {
  x=fork();
  if (x>0) exit(0);
}
while (1) sleep(2);
```

`ff-5.c`

```
for (i=0;i<k;i++) {
  x=fork();
  kill(x,SIGINT);
}
while (1) sleep(2);
```

`ff-6.c`

```
for (i=0;i<k;i++) {
  x=fork();
  kill(getppid(),SIGINT);
}
while (1) sleep(2);
```