Examen de programmation système

Juliusz Chroboczek

17 mai 2018

La durée de l'examen est de 2 heures 30 minutes. Les documents sont autorisés, le matériel électronique est interdit. Le sujet est composé de 2 pages.

La gestion des erreurs sera prise en compte dans la notation.

Exercice 1. Quels sont les contenus possibles du fichier `a` après l'exécution de chacun des fragments de code suivants, en supposant qu'aucun appel système n'a échoué ? (S'il y a plusieurs résultats possibles, il faut les donner tous.)

1.
```
fd = open("a", O_RDWR | O_CREAT | O_TRUNC, 0666);
write(fd, "toto", 4);
close(fd);
```
2.
```
fd = open("a", O_RDWR | O_CREAT | O_TRUNC, 0666);
fd2 = open("a",  O_RDWR | O_CREAT | O_TRUNC, 0666);
write(fd, "toto", 4);
write(fd2, "titi", 4);
close(fd);
close(fd2);
```
3.
```
fd = open("a", O_RDWR | O_CREAT | O_TRUNC, 0666);
fd2 = dup(fd);
write(fd, "toto", 4);
write(fd2, "titi", 4);
close(fd);
close(fd2);
```
4.
```
fd = open("a", O_RDWR | O_CREAT | O_TRUNC, 0666);
pid = fork();
if(pid > 0)
    write(fd, "toto", 4);
else
    write(fd, "titi", 4);
close(fd);
```

Exercice 2. Alice est consultant informatique. Son client a des problèmes de performance avec un programme écrit en C. Alice lance son outil de déboggage préféré `strace` qui affiche la suite des appels système effectués par le programme :

```
write(4, "L", 1)                        = 1
write(4, "a", 1)                        = 1
write(4, " ", 1)                        = 1
write(4, "f", 1)                        = 1
write(4, "i", 1)                        = 1
...
```

1. Expliquez *en une phrase* quel est le problème identifié par Alice.
2. Expliquez *en deux phrases au plus* comment résoudre le problème.

Exercice 3. Bernard fait de la recherche en bioinformatique, et il manipule un énorme fichier contenant des séquences de bases :

```
GTTAAGGCGGGAAA
GTTGCGAGGACA
...
```

Bernard doit compter le nombre d'instances de la suite « `GCGA` » dans son fichier. Il a écrit un prototype en Python, mais ça prend vraiment trop de temps.

Écrivez un programme C qui lit le fichier à l'aide de `mmap` puis qui parcourt la zone de mémoire ainsi obtenue et affiche le nombre d'occurrences de la chaîne « `GCGA` ». Vous pourrez vous servir de la fonction `memmem` :

```
void *memmem(const void *haystack, size_t haystacklen,
             const void *needle, size_t needlelen);
```

qui retourne un pointeur sur la première occurrence de `needle` (de longueur `needlelen`) dans la zone de mémoire `haystack` (de longueur `haystacklen`).

Points en plus si votre programme exploite les huit cœurs de la machine de Bernard (en utilisant des processus ou des *threads* POSIX), mais attention alors aux occurrences qui se trouvent à la frontière entre les différentes zones de mémoire. Points en moins si vous ne gérez pas les erreurs correctement.

Exercice 4. Chloé fait de la recherche en physique, et elle fait de gros calculs :

```
FILE *f = fopen("sortie.text", O_WRONLY | O_CREAT | O_TRUNC, 0666);
if(f == NULL) {
    perror("Eek");
    exit(1);
}
for(int i = 0; i < 100000; i++) {
    double y = groscalcul(i);
    fprintf(f, "%d: %lf\n", i, y);
}
flose(f);
```

Son programme prend énormément de temps, et elle voudrait savoir où il en est. Modifiez le programme de Chloé pour qu'il affiche la valeur de `i` à chaque fois qu'il reçoit le signal `SIGUSR1`. (On rappelle que l'opération `i++` n'est pas atomique vis-à-vis des signaux asynchrones, il faudra donc soit utiliser un type atomique soit effectuer une conversion asynchrone vers synchrone. On rappelle aussi que la fonction `printf` n'est pas *async signal safe*.)