Examen de programmation système

Juliusz Chroboczek

26 mai 2016

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 3 pages.

Exercice 1. Les services centraux de Paris-Diderot ont des problèmes de performance avec leur logiciel de comptabilité, notamment lors des entrées-sorties. Ils demandent à Alice de passer voir. Alice lance son outil de déboggage préféré *strace*, et remarque le fragment suivant :

```
...
read(3, "L", 1)                         = 1
write(4, "L", 1)                        = 1
read(3, "a", 1)                         = 1
write(4, "a", 1)                        = 1
read(3, " ", 1)                         = 1
write(4, " ", 1)                        = 1
read(3, "f", 1)                         = 1
write(4, "f", 1)                        = 1
read(3, "i", 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.
3. (hors barême) Alice a résolu le problème en 20 minutes, mais pour pouvoir payer son loyer, il faut qu'elle facture une journée entière. Que peut-elle faire pour faire croire au client qu'elle a travaillé toute la journée?

Exercice 2. Alice profite de son passage à Paris-Diderot pour rendre visite aux copains qui font une thèse au laboratoire de physique. Bernard a écrit un programme qui fait huit gros calculs et affiche leur somme :

```
extern double groscalcul(int);

int main() {
```

```
    double s = 0.0;
    for(int i = 0; i < 8; i++)
        s = s + groscalcul(i);
    printf("lf\n", s);
    return 0;
}
```

Bernard est un peu embêté, car il a demandé à son patron de lui acheter une machine avec huit cœurs, et son programme est toujours aussi lent que sur l'ancienne. Alice lui explique que son programme est séquentiel : il s'exécute sur un seul cœur. Mais ça tombe bien, une réduction, ça se parallélise facilement.

1. Écrivez un programme C qui crée une zone de mémoire partagée *anonyme* (pas besoin de `open` ou `shm_open`) contenant un tableau `double valeurs[8]`. Il engendre ensuite huit processus numérotés de 0 à 7 ; le processus numéro i calcule `groscalcul(i)`, stocke le résultat dans `valeurs[i]`, puis termine. Le programme principal attend la mort de ses huit fils, puis affiche la somme des huit valeurs.
2. Expliquez *en une phrase au plus* pourquoi il n'est pas nécessaire d'utiliser un sémaphore ou un *mutex* pour protéger la zone de mémoire partagée.

Exercice 3. Alice passe ensuite chez Chloé, qui calcule une approximation du point fixe d'une fonction contractante :

```
const double epsilon = 1.0E-6;
extern double f(double);

int main() {
    int n = 0;
    double x = 0.0;
    while(1) {
        double y = f(x);
        if(fabs(x - y) < epsilon)
            break;
        x = y;
        n++;
    }
    printf("Point fixe %lf, obtenu en %d itérations.\n", x, n);
    break;
    return 0;
}
```

On rappelle que la fonction `fabs` calcule la valeur absolue d'un paramètre de type `double`.

Chloé sait que son calcul n'est pas parallélisable, mais elle aimerait savoir où il en est. Modifiez le programme de Chloé pour qu'il affiche la valeur de n lorsqu'il reçoit le signal `SIGUSR1`. Vous pourrez soit supposer que n est suffisamment petit pour tenir dans une variable de type

`sig_atomic_t`, soit utiliser une variable globale pour signaler un événement à la boucle principale. Votre programme devra se comporter de façon raisonnable s'il reçoit plusieurs signaux de suite.

On rappelle les définitions suivantes :

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};

int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

Exercice 4. Consultant indépendant, c'est bien payé, mais un peu triste. Alice a donc décidé de prendre un emploi de programmeur à temps plein. À la réunion de leur équipe, Alice et Damien présentent le code qu'ils ont écrit pour compter le nombre d'instances de l'entier 42 dans une (grande) matrice carrée. Alice a écrit le code suivant :

```
int a[n][n];
...
int count = 0;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        if(a[i][j] == 42)
            count++;
    }
}
```

Damien, par contre, a décidé de parcourir le tableau par colonnes :

```
int count = 0;
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        if(a[j][i] == 42)
            count++;
    }
}
```

Alice dit que son code est plus efficace, que c'est par lignes qu'il faut parcourir la matrice. Damien dit que pas du tout, c'est par colonnes que ça ira plus vite. Éloïse, qui a toujours été très forte en algorithmique, dit que ça n'a pas d'importance car les deux algorithmes sont en $O(n^2)$.

Dites qui a raison, et expliquez *en un paragraphe au plus* pourquoi. Indication : en C, les tableaux sont stockés en mémoire par lignes, la case `a[i][j]` est suivie de la case `a[i][j+1]`.