Série d'exercices 2 : Fils d'exécution

2.1 Un serveur Web multi-fil a été programmé. Si on considère utiliser seulement des appels systèmes bloquants, est-ce que des fils d'exécution en mode usager sont une bonne solution?

2.2 L'information associée à un fil d'exécution contient une pile, les registres et un ensemble d'information maintenues par le noyau (numéro, table de page, etc). Nous savons que chaque fil a sa propre pile dans l'espace adressable. Par contre, le processeur ne contient qu'un ensemble de registres, pourquoi donc en avoir une copie par fil?

2.3 La plupart des processeurs Intel supportent la technologie HyperThread. Un processeur réel apparaît alors comme deux processeurs logiques et peut exécuter simultanément deux fils d'exécution. Les instructions des deux programmes sont exécutées en alternance. L'intérêt est que le processeur passe d'un fil à l'autre, et si un défaut de cache se produit sur un fil d'exécution, il peut continuer de faire progresser l'autre fil. Que cela demande-t-il comme ressources additionnelles dans le processeur? Est-ce que cela pose problème pour un système temps réel?

2.4 Vous faites la conception d'un serveur de fichier et hésitez entre une architecture mono ou multi-fil d'exécution. Le système possède un seul processeur. Chaque requête demande 15ms de traitement du CPU lorsque le bloc désiré est en cache d'E/S, ce qui arrive deux fois sur trois. Autrement, il s'ajoute 75ms de temps d'accès au disque pendant lequel le fil d'exécution est bloqué. Quelle sera la performance en requêtes servies par seconde, avec un ou plusieurs fils d'exécution?

2.5 Dans un système avec des fils d'exécution en mode noyau, tel que Linux, il faut une pile par fil. Est-ce différent lorsque les fils d'exécution sont gérés en mode usager?

2.6 Pendant le développement d'un nouveau processeur, un modèle est usuellement construit afin de le simuler. Le processeur est alors simulé une instruction à la fois, séquentiellement, même s'il s'agit en fait d'un processeur multicœur. Les conditions critiques (courses) qui se produisent normalement sur un processeur multicœur pourront-elles être simulées et détectées dans un tel contexte où la simulation procède séquentiellement?

Les questions suivantes sont associées à un code source.

2.7 Quelle est la valeur finale de a? Combien de millions de fois la variable a est-elle incrémentée?

```c
#define MAX 100000000 // 100 millions
static uint64_t a = 0;

void *count(void *arg) {
    volatile uint64_t *var = (uint64_t *) arg;
    volatile uint64_t i;
    for (i = 0; i < MAX; i++) {
        *var = *var + 1;
    }
    return NULL;
}

int main(int argc, char **argv) {
    int p;
    int i;
    pthread_t t1;
    pthread_t t2;

    pthread_create(&t1, NULL, count, &a);
    pthread_create(&t2, NULL, count, &a);
    count(&a);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("pid=%d a=%" PRId64 "\n", getpid(), a);
    return EXIT_SUCCESS;
}
```