

## Système - TP2

On souhaite écrire un programme calculant la somme des entiers de 1 à  $N$  à l'aide de plusieurs threads. Chaque thread va réaliser la somme d'un sous-ensemble de ces entiers et la somme globale est obtenue en réalisant la somme de ces résultats intermédiaires.

On répartit uniformément les entiers entre les threads de cette manière (exemple avec 3 threads) :

- Thread 1 : 1, 4, 7, ...
- Thread 2 : 2, 5, 8, ...
- Thread 3 : 3, 6, 9, ...

### Exercice 1

Écrire une fonction qui calcule la somme des entiers affectés à un thread :

```
int somme(int N, int id, int total);
```

Cette fonction prend en paramètres le dernier entier de la série  $N$ , le numéro du thread  $id$  ainsi que le nombre total de threads `total`.

### Exercice 2

Écrire un programme qui calcule la somme des entiers de 1 à  $N$  à l'aide de 4 threads. Ce programme doit lancer 4 threads, attendre qu'ils se terminent, faire la somme des résultats intermédiaires et afficher le résultat.

Chacun des threads doit récupérer son identifiant ainsi que le nombre total de threads, utiliser la fonction de l'exercice 1 pour calculer la somme et placer ce résultat dans un tableau global.

### Exercice 3

On souhaite que le programme principal n'ait pas à faire la somme finale lui-même, les threads doivent donc calculer directement la somme dans une variable globale. Modifier le programme suivant de manière à ce que chaque thread ajoute lui-même son résultat intermédiaire dans une variable globale plutôt que de le stocker dans un tableau.

### Exercice 4

Le programme suivant réalise un tri rapide :

```

int partitionner(int *tab, int p, int r) {
    int pivot = tab[p], i = p-1, j = r+1;
    int temp;
    while (1) {
        do {
            j--;
        } while (tab[j] > pivot);
        do {
            i++;
        } while (tab[i] < pivot);
        if (i < j) {
            temp = tab[i];
            tab[i] = tab[j];
            tab[j] = temp;
        } else {
            return j;
        }
    }
}

void quickSort(int *tab, int debut, int fin) {
    int med;
    if (debut < fin) {
        med = partitionner(tab, debut, fin);
        quickSort(tab, debut, med);
        quickSort(tab, med + 1, fin);
    }
}

```

On peut voir que les deux appels récursif à la fonction quickSort sont indépendants, on souhaite donc les paralléliser. Modifier ce programme de manière à ce que les tri récursifs soient fait en parallèle dans des threads différents.