

# TP 1 – Threads

## Exercice 1 – threads indépendants

Soit un programme qui prend en argument 2 valeurs  $n$  et  $p$  et démarre  $n$  threads. Chaque thread reçoit la valeur  $p$  ainsi que le numéro  $t \in [1, n]$  du thread et calcule  $u_t = \sum_{i=1}^p ((t-1)p + i)$ . Le thread principal affiche la somme des valeurs  $u_t$  calculées. Rappel : le résultat final de votre programme doit être égal à  $np(np+1)/2$ .

Rédigez le programme selon la spécification initiale (somme des  $u_t$ ). Vous prendrez soin de n'utiliser aucune variable globale. La valeur  $u_t$  retournée par chaque thread doit être placée dans une zone mémoire distincte des arguments.

## Exercice 2 – barrières

Un calcul « data-parallèle » consiste à ce que  $n$  threads réalisent des calculs similaires en utilisant ou produisant des données distinctes. En général, le calcul parallèle est suivi d'une partie séquentielle réalisée par l'un des threads.

Dans cet exercice, le calcul est simplifié à l'extrême : on considère une chaîne de  $n$  caractères. Chaque thread  $i$  (avec  $0 \leq i < n$ ) remplit la case d'indice  $i$  de cette chaîne avec alternativement le caractère « # » ou « - ». Lorsque les  $n$  threads ont écrit leur caractère, l'un d'eux (n'importe lequel) se charge d'afficher la chaîne ainsi calculée, puis on commence une nouvelle phase de calcul, pour un total de  $p$  itérations.

L'algorithme de chaque thread  $i$  est donc constitué d'une boucle parcourue  $p$  fois : à chaque tour, le thread remplit la case  $i$  de la chaîne avec « # » ou « - », puis attend que les  $n-1$  autres threads aient rempli leur propre case, et enfin l'un des threads affiche la chaîne une fois l'attente terminée.

La sortie du programme (avec  $n = 20$  et  $p = 2$ ) sera :

```
##### (une ligne de #)
----- (une ligne de -)
##### (une ligne de #)
----- (une ligne de -)
```

Votre programme prendra  $n$  et  $p$  en arguments. Vous n'utiliserez bien sûr aucune variable globale.