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 deux (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.

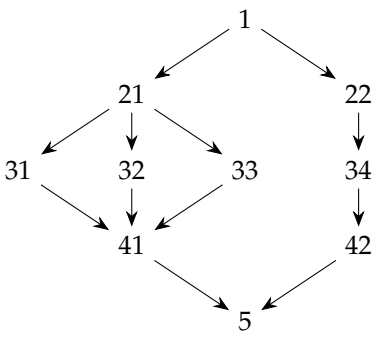
La sortie du programme (avec $n = 20$ et $p = 2$) sera :

```
####################   (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.

Exercice 3 – threads, create-join et dépendances

Un graphe de dépendances est un graphe orienté dont les sommets sont des « tâches » et les arcs représentent des contraintes d'ordre entre les tâches. Pour certains types de graphes de dépendances (les graphes dits « série-parallèle »), le parallélisme peut être maximal et les dépendances peuvent être satisfaites en utilisant seulement la création et la jonction de threads (c'est-à-dire `pthread_create()` et `pthread_join()`), sans avoir à utiliser d'autre mécanisme de synchronisation.



Vous trouverez ci-contre un tel graphe, où l'on voit par exemple que les tâches 21 et 22 doivent être effectuées après la tâche 1, la tâche 21 devant être faite avant les tâches 31, 32 et 33, qui peuvent elles-mêmes être exécutées en parallèle.

Écrivez un programme exécutant les tâches de ce graphe en respectant les dépendances. Le code de chaque tâche est un appel à une fonction `void tache(int i)` unique, avec en argument le numéro de la tâche (cette fonction se contentera d'afficher le numéro de la tâche.)