

## HMIN105M TD-TP

### Processus multitâches ou *multi-threads* et synchronisation

Durée estimée pour réaliser ce travail : 7,5h

## 1 Introduction

Ce TD-TP étudie le fonctionnement des processus multi-tâches et la synchronisation entre activités parallèles d'un même processus. On retrouve souvent ce fonctionnement dans la littérature sous le vocabulaire *multi-threads*. Les exercices suivants ont pour objectif d'apprendre à utiliser les outils vus en cours pour résoudre différents problèmes.

### 1.1 La base : Parallélisme de tâches

Téléchargez les fichiers sources fournis sur Moodle.

1. Compléter le squelette pour que le programme principal lance plusieurs tâches s'exécutant en parallèle et pour montrer (voir à l'écran) que ces tâches se déroulent bien en parallèle.
2. Que se passe-t-il si la tâche principale se termine sans attendre la fin des autres tâches ?
3. Que se passe-t-il si une des tâches fait un appel à `exit()` ?
4. Proposer un schéma algorithmique ainsi que son implémentation permettant de constater que lorsqu'un processus crée une donnée en mémoire puis génère plusieurs tâches, toutes les tâches ont accès à cette même donnée. Pour rappel, ceci permettra de constater un fonctionnement opposé à celui des processus enfants d'un même parent.

## 2 Exercice : produit scalaire multi-threads

On souhaite réaliser le produit scalaire (somme de produits) de deux vecteurs en parallèle (à l'aide de plusieurs threads). On suppose que si les vecteurs sont de dimension  $N$ , il y aura  $N$  threads dédiés aux multiplications.

Dans la solution demandée :

- l'initialisation des deux vecteurs est à faire par le thread principal ;
- les données nécessaires au calcul des threads sont à passer en paramètre ;
- une fois les threads de multiplication terminés, le thread principal fera la somme des résultats ;
- le résultat final est à afficher par le thread principal.

1. Proposer et implémenter le schéma algorithmique correspondant à cette spécification
2. On suppose maintenant que l'addition n'est plus effectuée par le thread principal mais par les threads : chaque thread de multiplication ajoute son résultat à une somme globale (variable qui sera donc partagée). Le thread principal affichera le résultat final.

## 3 Rendez-Vous

Dans un problème classique de rendez-vous entre tâches, chaque tâche lancée effectue un premier travail puis attend que toutes les autres aient aussi terminé leur premier travail (point de rendez vous) avant de poursuivre leur exécution pour effectuer un second travail.

1. Quel paradigme de synchronisation utiliserez vous pour mettre en place ce rendez-vous : exclusion mutuelle ou attente d'un événement ? Les critères de choix sont la correction et l'efficacité.
2. Proposer une solution pour ce problème et pour  $n$  tâches sous forme de schéma algorithmique puis implémenter votre solution.

## 4 Traitement synchronisé

On envisage un traitement parallèle d'une image par plusieurs activités d'un même processus, chacune ayant un rôle déterminé. Plus précisément, il s'agit d'effectuer une suite ordonnée de traitements d'image, chaque traitement est implémenté par une activité et chaque activité peut travailler sur un sous ensemble de points (pixels) de l'image, appelée *zone*, en parallèle avec un autre traitement.

En supposant que l'image est une suite de *zones* ordonnées, les traitements doivent se faire :

- avec garantie d'exclusivité : une activité ne doit pas accéder à une zone en cours de traitement par une autre activité,
- en respectant un ordre bien déterminé entre les activités : sur toute zone, l'activité  $T_1$  doit passer en premier, puis  $T_2$  etc.

On impose aussi le fait que chaque activité traite les zones dans leur ordre successif (1, 2, 3, ...)

Pour commencer, on se limite à deux activités.

1. Proposer une structure de l'application et une solution algorithmique permettant un fonctionnement correct et efficace pour deux activités. Implémenter et tester votre solution.

On passe maintenant à trois activités (et plus) et on considère la solution suivante :

À chaque activité  $T_i$  est associée une donnée **commune**  $d_i$ , telle que  $d_i$  contient le numéro de zone en cours de traitement par  $T_i$ .  $T_{i+1}$  peut savoir, en consultant  $d_i$  si elle peut traiter la zone à laquelle elle veut passer.

**Exemple** : si l'activité  $T_2$  est en train de traiter la zone  $Z_5$ ,  $d_2$  contient la valeur 5, positionnée par  $T_2$  avant de commencer le traitement de  $Z_5$ .  $T_3$  ne pourra travailler sur  $Z_5$  que lorsque  $d_2$  sera positionnée à  $> 5$ .

Conséquence : il y a un « espace commun » à toutes les activités (tableau commun  $d[n]$ ), contenant autant d'éléments que d'activités traitant l'image. Chaque activité peut consulter ces données mais ne peut modifier que la donnée correspondant à son rang (chaque  $T_k$  peut modifier seulement  $d_k$ ).

2. On utilise un seul verrou protégeant l'accès au tableau. Écrire le schéma algorithmique d'une activité. Montrer que cette solution fonctionne correctement (répond à la contrainte d'exclusivité et d'ordre). Cette solution est-elle efficace ?
3. Si vous pensez que la solution précédente n'est pas efficace, proposez une meilleure solution.
4. Implémenter progressivement vos solutions, en simulant un temps de travail aléatoire pour chaque activité travaillant sur l'image.