

TP n° 8: Banqueroute

L'objet de ce TP est de continuer à gérer les problèmes de compétitions dans les programmes concurrents.

Exercice 1.

Les Sims

Dans cet exercice, on va utiliser les fils pour simuler la gestion d'un compte commun à un couple. Deux threads correspondront aux rentrées d'argent par leurs salaires respectifs, effectuées de manière plus ou moins régulière¹. Un thread représentera une horloge atomique, mettant à jour un compteur représentant la date dans la simulation (à intervalle régulier, cette fois ci, d'une journée²). Deux threads représenteront enfin les deux membres du couple, utilisant deux cartes bancaires (une par personne), reliées au même compte commun.

Le but est d'afficher à l'écran les différentes transactions telles qu'elles apparaissent et ce qu'il reste sur le compte.

La sortie aura le format suivant :

« *Date* - *Utilisateur* : Opération de *Valeur*€. Il reste *Valeur*€. »

Si une opération ne peut pas être faite (par exemple s'il ne reste plus d'argent sur le compte), afficher

« *Semaine* 23 - *Anne* : Retrait de 25 € n'a pas pu être effectué. Il reste 12€. ».

Le montant des opérations (et le solde sur le compte) doivent utiliser des double.

1. Coder la fonction correspondant au corps du fils « horloge atomique ». A-t-on besoin de primitives de synchronisation liées à la gestion du temps ?
2. Coder la fonction correspondant au corps des fils « salaires », qui n'effectue que des dépôts, et y intégrer un peu d'aléa. Attention, chaque salaire doit être cependant versé une unique fois par intervalle d'un mois, et les salaires des deux membres du couple ne sont ni égaux³, ni versés au même moment.
3. Coder la fonction correspondant au corps des fils « membre du couple ». Chaque membre du couple effectue des retraits et dépôts de montants aléatoires, à intervalles aléatoires.
4. Lancer la simulation et vérifier que tout fonctionne bien. Vérifier que la loi de Benford⁴ est bien vérifiée.

Exercice 2.

Parallélisme automatisé

Cet exercice est un exercice d'introduction à *OpenMP*. *OpenMP* permet de simplifier la gestion du parallélisme en mémoire partagée. L'inclure à utiliser est `omp.h`, et le drapeau de compilation de *gcc* à utiliser est `-fopenmp`. L'un des intérêt d'*OpenMP* est d'utiliser des `pragma` (des directives préprocesseur). Ces directives sont de la forme :

```
#pragma omp parallel num_threads(4)
printf("Toto\n");
```

Dans ce code, 4 *threads* vont exécuter le code dans la région parallèle, ici afficher "Toto". L'intérêt des `pragmas` étant que, si on compile sans mettre `-fopenmp`, cela compile quand même sans utiliser *OpenMP*!

Par exemple, dans code suivant on souhaite afficher un tableau de 1000 entiers. Pour cela on peut utiliser 4 *threads*, chaque *threads* va afficher 250 valeurs du tableau.

```
int n;
#pragma omp parallel for num_threads(4)
for (n=0; n<1000; ++n)
    printf("%d\n", array[n]);
```

1. Utilisez le bout de code précédent pour afficher le tableau {0, 1, 2, ..., 999}. Que remarquez-vous?

1. Quoi, c'est pas le dernier mercredi du mois? <https://www.cf-credits.com/calendrier-salaire-fonctionnaire>

2. On va dire que l'électron est un peu fatigué, vivement les vacances.

3. L'un des membres étant auparavant normalien élève, il a déjà passé un échelon.

4. https://fr.wikipedia.org/wiki/Loi_de_Benford

2. Reprenez votre code pour le produit matriciel et, à la place des `pthread`s, utilisez *OpenMP*.
3. Essayez les différentes options d'*OpenMP* (pour faire varier par exemple l'ordonnancement, le nombre de threads, etc.), quel impact cela a-t-il sur les performances ?
4. Que pouvez-vous dire des performances d'*OpenMP* par rapport à votre programme multi-fils.