Programmation système avancée
TP n° 9 : Threads

Exercice 1 :

Le but de cet exercice est d'écrire un programme qui attend pendant au plus cinq secondes que l'utilisateur appuie sur la touche entrée, et cela sans utiliser de fonctions avec des *timeouts* explicites (par exemple `select`) et sans utiliser de signaux.
Écrivez un programme qui crée une variable booléenne globale g, un *mutex* μ qui protège g, et une variable de condition κ. Il crée ensuite deux *threads* t_1 et t_2. Le *thread* t_1 attend indéfiniment que l'utilisateur appuise sur entrée, et, si c'est le cas, affecte vrai à g puis signale κ. Le *thread* t_2 attend pendant cinq secondes (`sleep`) puis signale κ. Quant au programme principal, il attend sur κ puis affiche un message qui dépend de la valeur de g.

Exercice 2 :

Le but de cet exercice est de calculer π en évaluant le rapport entre la surface d'un carré de côté c et celle d'un quart de disque de rayon c : le rapport entre les deux est $\frac{\pi}{4}$.

1. Implémentez une fonction `interieur(int c)` qui compte le nombre de points à l'intérieur du quart de disque de centre $(0, 0)$ et de rayon c, c'est-à-dire le nombre de points (x, y), pour tous entiers $0 \leq x \leq c$ et $0 \leq y \leq c$ qui vérifient

$$x^2 + y^2 \leq c^2$$

   Écrivez un main qui prend `c` en ligne de commande et affiche `4.0*interieur(c)/(c*c)`. Regardez comment ce rapport évolue quand c devient grand (il doit tendre vers π), et notez les performances. (N'oubliez pas de compiler avec `-O2`.)

2. Accélèrer le calcul en utilisant des *pthreads*. Chaque thread devra prendre en paramètre, outre le côté c, deux entiers i_1 et i_2, et compter les points (x, y) qui vérifient $x^2 + y^2 \leq c^2$, pour tous $i_1 \leq x \leq i_2$ et $0 \leq y \leq c$. Vous utiliserez une variable globale pour compter les points trouvés, dûment protégée par un *mutex*.

   Le main prendra en ligne de commande le nombre t de threads, en plus de c, et il attendra la terminaison de tous les *threads* (`pthread_join`). Notez les performances.

3. Le programme précédent est très inefficace, car il faut prendre un *mutex* à chaque point trouvé. Modifiez-le pour que chaque *thread* compte les points dans une variable locale, puis ajoute sa valeur à la variable globale à la fin. Notez les performances.

4. Que se passe-t-il si vous modifiez le programme de la question 2 pour qu'il utilise une variable atomique ? (La variable atomique est une donnée globale partagée, ce qui cause du *cache line bouncing*.)