Programmation Système
TP n° 8 : Mutexes et variables de condition
12-14 mars 2019

Exercice 1 : Barrière de synchronisation

Le but de cet exercice est d'implémenter une *barrière de synchronisation* pour un nombre N de processus. Les processus atteignant cette barrière de synchronisation doivent attendre que le dernier processus atteigne la barrière pour continuer leurs exécutions respectives. Concrètement, il faut implémenter un type `barriere_t` et une fonction

```
void barriere(barriere_t*)
```

qui sera appellée exactement une fois dans chaque processus, et qui vérifiera la propriété suivante :

- si un processus n'est pas le dernier à atteindre la barrière, alors il doit attendre jusqu'à ce que le dernier processus atteigne la barrière pour pouvoir continuer ; i.e. la fonction `barriere` doit bloquer le processus appellant jusqu'à ce que le dernier processus appelle `barriere` ;
- lorsque le dernier processus appelle `barriere`, tous les processus sont débloqués simultanément.

Afin d'implémenter `barriere_t` et la fonction `barriere`, vous aurez besoin d'un mutex, d'une variable stockant le nombre de processus ayant atteint la barriere et d'une variable permettant au dernier processus de signaler que les $N-1$ précédents peuvent continuer.
La structure de type `barriere_t` devra bien sûr être stockée en mémoire partagée.
Veillez à initialiser le mutex à l'aidede `pthread_mutexattr_init`, `pthread_mutexattr_init` et enfin `pthread_mutex_init`.
Vous testerez votre implémentation en deux étapes :

- un processus père initialisera la structure `barriere_t` et son contenu puis via `fork` créera N processus
- chacun des processus attendra un temps aléatoire de quelques secondes avant de faire un appel à `barriere`

Via la commande `top`, vous surveillerez la consommation CPU des processus en attente. Que constatez-vous ?

Exercice 2 : Barrière de synchronisation avec variables de condition

Modifiez votre solution de l'exercice précédent en utilisant des variables de condition (via `pthread_cond_wait`) plutôt qu'une attente active.
Via la commande `top`, vous surveillerez la consommation CPU des processus en attente. Comparez avec la solution de l'exercice précédent.