TP 4 (à rendre) – Synchronisation

Modalités de remise

Ce TP peut-être réalisé en binôme.

Vous déposerez votre TP sous forme d'une archive (au format « *login*.tar.gz » où *login* est votre nom de login sur Turing) sur Moodle, dans l'espace de devoirs prévu à cet effet. Vous n'oublierez pas de mettre vos noms et prénoms en évidence dans vos fichiers.

Date limite : mercredi 13 novembre à 12 heures (heure limite du devoir Moodle).

Exercice 1

On cherche dans cet exercice à implémenter le mécanisme de producteur-consommateur en utilisant les threads POSIX, les sémaphores et tout autre mécanisme de synchronisation (barrière, mutex, etc.) que vous jugerez utile.

Le programme qu'on vous demande de rédiger doit prendre en argument le nombre de threads producteurs et le nombre de threads consommateurs, et se terminer suite à la pression par l'utilisateur de la touche CTRL-C (signal `SIGINT`) en affichant 4 valeurs, quel que soit le nombre de threads : le nombre total de valeurs produites et leur somme, ainsi que le nombre total de valeurs consommées et leur somme.

Chaque producteur produit des valeurs aléatoires (entiers $\in [0, 256[$) et les place dans un tampon partagé que vous passerez en paramètre au thread. Les producteurs consomment ces données à partir du tampon que vous passerez également en paramètre. Lorsque l'utilisateur appuie sur CTRL-C, les producteurs doivent signaler le fait aux consommateurs (par exemple en produisant la valeur -1, en faisant attention à ce qu'aucun autre producteur ne produise postérieurement une donnée valide), puis se terminer en renvoyant (valeur de retour du thread) le nombre de valeurs produites et leur somme. Chaque consommateur doit, lorsqu'il détecte la terminaison, se terminer en renvoyant le nombre de valeurs consommées et leur somme.

Exercice 2

On souhaite maintenant remplacer l'implémentation des sémaphores par votre propre implémentation, c'est-à-dire qu'on souhaite définir 4 fonctions similaires aux fonctions standard POSIX :

```
int monsem_init (monsem_t *s, int val) ;
int monsem_wait (monsem_t *s) ;
int monsem_post (monsem_t *s) ;
int monsem_getvalue (monsem_t *s, int *v) ;
```

En vous aidant des mutex et des conditions, rédigez ces quatre fonctions et adaptez votre programme de l'exercice précédent pour les utiliser.

Note : vous n'oublierez pas de prendre en compte le fait qu'un appel unique à `pthread_cond_signal` peut réveiller plusieurs threads attendant la condition (et pas seulement un seul).