POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Dpartement de Génie Informatique et Génie Logiciel
INF2610 - Noyau d'un système d'exploitation
TP 3 : Synchronisation des threads
Hiver 2021

Objectifs

Ce TP a pour but de vous familiariser avec les threads et les mécanismes de synchronisation de l'API *POSIX*. Il s'agit de compléter le programme *GuerreDesChiffres.c*, pour créer et synchroniser des threads selon le modéle de synchronisation producteurs et consommateurs. Dans ce modèle, il y a un ensemble de threads producteurs et un ensemble de threads consommateurs qui communiquent via un tampon de taille limitée. Les nombres de threads producteurs et de threads consommateurs ainsi que la taille du tampon sont des paramètres du programme *GuerreDesChiffres.c*.

Programme GuerreDesChiffres.c à compléter :

```
// ...
// fonction exécutée par les producteurs
void* producteur( void* pid) {
  //...
  while(1) {  // ...
              /* générer aléatoirement un chiffre non nul  \\
             à déposer dans le tampon.Vous pouvez
             utiliser : srand(time(NULL));  x=(rand() % 9) + 1;
            pour générer aléatoirement un chiffre dans x*/
        // ...
       }
  // ...
}
// fonction exécutée par les consommateurs
void* consommateur(void *cid) {
    // ...
    while(1) {  // ...
        // retirer un chiffre du tampon.
        // ...
    }
   // ...
 }
```

```
// fonction main
int main(int argc, char* argv[]) {
   /* Les paramètres du programme sont, dans l'ordre :
      le nombre de producteurs, le nombre de consommateurs
      et la taille du tampon.*/
  // ....
}
```

Comportement attendu du programme

Les threads producteurs et consommateurs sont créés dans la fonction main du programme (thread principal).

Chaque thread producteur consiste en une boucle sans fin. À chaque itération, il génère aléatoirement un chiffre non nul qu'il dépose dans le tampon. Les threads producteurs mémorisent dans une variable globale la somme de tous les chiffres qu'ils ont générés.

Chaque thread consommateur est composé d'une boucle sans fin. À chaque itération, il récupère du tampon un chiffre. La somme des chiffres récupérés par tous les threads consommateurs est sauvegardée dans une autre variable globale.

Après la création de tous les threads, le thread principal du programme arme une alarme de 1 seconde (en utilisant l'appel système *alarm*) puis se met en attente de la fin de tous les threads producteurs. Le programme doit donc faire le nécessaire pour capter le signal $SIGALRM$. Le traitement du signal consiste à mettre à $true$ une variable globale $flag\_de\_fin$ initialisée à $false$.

À la fin de tous les threads producteurs, le thread principal dépose dans le tampon autant de chiffres 0 qu'il y a de threads consommateurs puis se met en attente de tous les threads consommateurs. Enfin, il affiche les sommes mémorisées des chiffres produits et consommés avant de se terminer. Ces deux sommes devraient être égales.

En plus du traitement décrit ci-dessus, chaque thread producteur teste, à la fin de chaque itération, la valeur de la variable $flag\_de\_fin$. Il met fin à son traitement, si cette valeur est $true$. Chaque thread consommateur

met fin à son traitement lorsqu'il récupèrera le chiffre 0 du tampon. Il doit cependant compléter l'itération en cours avant de se terminer.

Travail à faire et remise

Il vous est demandé de compléter le programme $GuerreDesChiffres.c$ afin qu'il réalise le traitement attendu décrit ci-dessus. Faites attention aux accès concurrents et aux interblocages.

Utilisez les sémaphores $POSIX$ ($sem\_wait, sem\_post, sem\_init$, etc.), pour éviter les conditions de concurrence et synchroniser adéquatement tous les threads $POSIX$ créés. Indiquez, sous forme de commentaires dans le code, le rôle de chaque sémaphore.

Compilez votre programme en utilisant la ligne de commande :

$$gcc - pthread\ GuerreDesChiffres.c\ -o\ GuerreDesChiffres$$

Testez votre programme pour différentes valeurs de paramètres. Par exemple, pour le tester pour le cas de 3 producteurs, 2 consommateurs et un tampon de taille 5, tapez la ligne de commande suivante :

$$./GuerreDesChiffres\ \ 3\ \ 2\ \ 5$$

En cas de doute sur le bon fonctionnement de votre programme, n'hésitez pas à tester les valeurs de retours de vos appels système et variables. L'appel système $sem\_getvalue$ permet de recupérer la valeur courante d'un sémaphore.

Pour la remise, déposez le fichier contenant votre programme sur le site moodle du cours.

Barème

Description	Points
Création des threads et tampon	4
Synchronisation adéquate des threads	6
Signal, terminaison et attente des threads	5
Résultat correct	3
Clareté du code et commentaires	2