INF2610

TP #5 : Synchronisation des threads

Polytechnique Montréal

Automne 2023
Date de remise: Voir le site Moodle du cours
Pondération: 5%

Objectifs

Ce TP a pour but de vous familiariser avec les threads et les mécanismes de synchronisation de lAPI *POSIX*. Il sagit de compléter le programme `GuerreDesChiffres.c` suivant, 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`.

Comportement attendu du programme

Les threads producteurs et consommateurs sont créés dans la fonction main du programme (thread principal). Ces threads ont chacun un seul paramètre qui correspond à son numéro: 0 est le numéro du premier producteur/consommateur créé, 1 est le numéro du second producteur/consommateur créé, etc.

Chaque thread producteur consiste en une boucle sans fin. À chaque itération, il génère aléatoirement un chiffre non nul quil dépose dans le tampon. Il calcule également le nombre de chiffres non nuls générés. Ce nombre est communiqué au thread principal via `pthread_exit` et `pthread_join`. Les threads producteurs mémorisent aussi dans une variable globale la somme de tous les chiffres quils ont générés.

De façon similaire, chaque thread consommateur est composé dune boucle sans fin. À chaque itération, il récupère du tampon un chiffre. Il calcule également le nombre de chiffres non nuls récupérés. Ce nombre est communiqué au thread principal via `pthread_exit` et `pthread_join`. La somme des chiffres récupérés par tous les threads consommateurs est mémorisée 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 lappel 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 qui est initialisée à `false`.

À la fin de tous les threads producteurs, le thread principal dépose dans le tampon autant de chiffres 0 quil y a de threads consommateurs puis se met en attente de tous les threads consommateurs. Il affiche ensuite les sommes mémorisées des chiffres produits et consommés. Ces deux sommes devraient être égales. Il affiche également les nombres de chiffres produits par les producteurs et consommés par les consommateurs avant de se terminer. Ces deux nombres devraient aussi être égaux.

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 lorsquil récupèrera le chiffre 0 du tampon. Il doit cependant compléter litération en cours avant de se terminer.

Listing 1: 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));
       (à appeler une seule fois) et 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.*/
  // ....
}
```

Travail à faire et remise

Il vous est demandé de compléter le programme `GuerreDesChiffres.c` afin quil 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. Attention à lutilisation abusive de sémaphores.

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, nhésitez pas à tester les valeurs de retours de vos appels système et variables.

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

Barème

Description	Points
Gestion des paramètres, création du tampon et des threads	4
Synchronisation adéquate des threads	5
Signal, attente, communication et terminaison des threads	5
Bon fonctionnement du code	4
Clarté du code et commentaires	2