Université de Reims Champagne-Ardenne
Département de Mathématiques, Mécanique et Informatique

Pierre Delisle

Travaux pratiques 2

Mutex et variables de condition

Exercice 1 – Compteur, affichage et remise à 0

1. Écrivez un programme qui utilise 3 threads (4 avec le main) :
    - Un thread qui incrémente un compteur global partagé, puis dort 2 secondes ;
    - Un thread qui affiche le compteur quand c'est un multiple de 2, puis dort 1 seconde ;
    - Un thread qui remet le compteur à 0 quand il atteint 9.

Le thread principal (le main) doit créer l'ensemble des threads. Les 3 threads ainsi créés devront exécuter leur routine dans une boucle infinie et afficher un message à chaque itération de la boucle de sorte à indiquer qui ils sont : « je suis le thread incrémenteur », « je suis le thread afficheur », …

2. Modifiez votre programme de sorte à ce qu'il y ait 4 threads afficheurs et 4 threads de remise à 0, donc 9 threads (10 avec le main). Chacun de ces threads doit afficher un identifiant entre 1 et le nombre de threads de son type (afficheur 1, afficheur 2, etc.). Lorsqu'une des 2 opérations (affichage ou remise à 0) doit être réalisée, un seul thread devra la réaliser parmi ceux disponibles. Autrement dit, lorsqu'il faudra afficher le compteur, il ne devra être affiché qu'une seule fois par un des threads afficheurs.
3. Vous avez probablement appelé la fonction **pthread_cond_signal** pour réveiller les threads dans vos 2 programmes précédents (vous auriez du !). Remplacez ces appels par **pthread_cond_broadcast**. Vérifiez que votre programme fonctionne toujours correctement et si ce n'est pas le cas, corrigez-le en conséquence.

Exercice 2 – Le service des étrangers de la sous-préfecture de Reims

À partir des informations données plus bas, écrivez le programme complet « **sousPref.c** ».

La sous-préfecture de Reims fait appel à vous pour simuler le fonctionnement de son service aux étrangers : une grande file d'attente pleine d'étrangers (en train d'être absents à leur travail et à leurs cours) et seulement quelques fonctionnaires pour traiter leurs dossiers ! Pour ce faire, vous devez lui proposer un programme multithread écrit en langage C et utilisant la bibliothèque Pthreads. Ce programme doit être constitué du thread principal (main), d'un thread permettant de faire entrer des étrangers dans la salle d'attente et de plusieurs threads simulant chacun des guichets des fonctionnaires pouvant recevoir les étrangers.

Les constantes

Les seules constantes à définir sont le nombre de guichets utilisés dans le programme et la limite de capacité de la salle d'attente. Le code peut donc contenir les définitions de constantes « #define NB_GUICHETS 4 » (nous sommes optimistes sur le nombre de fonctionnaires disponibles en même temps) et « #define CAPACITE 10 » .

Les structures de données

Afin de gérer l'arrivée des étrangers, nous avons besoin d'une structure représentant une salle d'attente. Cette structure doit contenir, entre autres, le nombre d'étrangers présents à un moment précis dans la salle. Au début du programme, la salle d'attente est vide, donc le nombre d'étrangers est initialisé à 0. Ce nombre est ensuite augmenté (de 1 ou plusieurs à la fois) lors de l'arrivée d'étrangers et diminué lors du traitement du dossier d'un étranger par un fonctionnaire. Vous n'avez pas à gérer l'identification personnelle de chaque étranger : supposez que le premier étranger arrivé est toujours le premier traité. Aussi, comme toute sous-préfecture qui se respecte, lorsque la salle d'attente dépasse sa limite de capacité, on verrouille les portes de la sous-préfecture. Vous devez déclarer et initialiser la variable représentant la salle d'attente en global (à l'extérieur de toute fonction).

Les threads

Vous devez définir un thread permettant de simuler l'arrivée des étrangers dans la salle d'attente. Pour ce faire, ce thread doit permettre d'entrer au clavier un certain nombre d'étrangers d'un coup et augmenter le nombre de personnes dans la salle en conséquence. Le thread doit permettre d'entrer de nouvelles arrivées durant l'exécution du programme et ce, tant que la capacité de la salle le permet. On tolérera un dépassement de la capacité lorsqu'une entrée individuelle fait dépasser la capacité de la salle : si la capacité est de 10, qu'il y a actuellement 8 personnes dans la salle et que 5 personnes arrivent en même temps, on permettra l'entrée des 5 personnes et la présence de 13 personnes dans la salle. Les portes seront alors verrouillées jusqu'à ce que le nombre d'étrangers redevienne inférieur à la capacité de la salle.

Vous devez aussi créer autant de threads qu'il y a de guichets, chacun possédant un identificateur entre 0 et NB_GUICHETS-1. Ces threads sont en attente d'étrangers pendant l'exécution du programme. Lorsque des étrangers arrivent dans la salle d'attente, ils peuvent être traités par un des guichets disponibles, un guichet ne pouvant traiter qu'un seul étranger à la fois. Lorsqu'un fonctionnaire appelle un étranger pour traiter son dossier, le système affiche le message suivant :

```
Le prochain étranger est appelé au guichet 2, il reste 8 personnes dans la salle d'attente.
```

Finalement, le thread principal (main) doit créer les différents threads en les initialisant correctement.

Lorsque vous aurez terminé l'exercice, reprenez-le en utilisant un nombre de guichets initialisé de façon dynamique, par exemple au clavier ou en paramètre du programme.

Exercice 3 – Calcul de la somme de *n* valeurs

Écrivez un programme multi-threadé permettant de calculer la somme des valeurs d'un tableau d'entiers positifs initialisés aléatoirement. Vous pouvez définir le nombre de valeurs (la taille du tableau) et le nombre de threads comme constantes dans le programme. Pour simplifier, supposez aussi que le nombre de valeurs est toujours un multiple du nombre de threads.

Directives :

- Le tableau de valeurs et la somme doivent être des variables déclarées de façon globale ;
- Chaque thread doit afficher la somme des valeurs qu'il a traitées, ainsi que son indice (entre 0 et nbThreads-1) ;
- Le thread principal (main) doit afficher à l'écran la somme globale.