Utilisation des sémaphores

Exercice 1 : Un rendez-vous

Énoncé

Cet exercice est tiré de *A little book of semaphores*, par A. B. Downey. Soit les séquences de code suivantes :

```
void threadA ()
{
     instructionA1 ;
     instructionA2 ;
}
```

```
void threadB ()
{
     instructionB1 ;
     instructionB2 ;
}
```

Nous voulons garantir que la ligne instructionA1 soit exécutée avant la ligne instructionB2, et que la ligne instructionB1 soit exécutée avant la ligne instructionA2.

Cela porte le nom de *rendez-vous* car les deux threads se rencontrent en un point d'exécution donné.

Pour résoudre ce problème, il suffit de créer deux sémaphores, initialisés tous les deux à zéro, et que nous appellerons aRencontre et bRencontre.

Voici une proposition de solution invalide :

```
void threadA ()
{
     instructionA1;
     down(&bRencontre);
     up(&aRencontre);
     instructionA2;
}
```

```
void threadB ()
{
     instructionB1;
     down(&aRencontre);
     up(&bRencontre);
     instructionB2;
}
```

Expliquez ce qui ne va pas dans cette proposition et proposez une solution qui fonctionne.

Exercice 2 : Laverie automatique

Énoncé

Une laverie entre dans l'ère de l'informatique : le client saisit le nombre de machines à laver dont il a besoin sur un des ordinateurs à sa disposition. Ils sont reliés à un ordinateur central qui assigne automatiquement les machines à laver disponibles. Le client dépose son linge à laver dans les machines indiquées. Lorsqu'une machine à laver finit son cycle, elle informe l'ordinateur central qu'elle est de nouveau disponible.

L'ordinateur central tient à jour un tableau de booléens disponible[NMACHINES] qui indique si la machine à laver correspondante est disponible (NMACHINES est une constante : le nombre de machines à laver dans la laverie). Voici le code pour allouer et relâcher une machine ; le tableau disponible[] est initialisé à VRAI et nlibre est un sémaphore initialisé à NMACHINES.

```
#define VRAI 1
#define FAUX 0
int alloue() /* Retourne l'indice d'une machine disponible. */
{
  sem_wait(&nlibre); /* Attend qu'une machine soit disponible */
  for (int i=0; i < NMACHINES; i++) /* On parcourt les machines */
    if (disponible[i] == VRAI)      /* La première machine disponible */
    {
      disponible[i] = FAUX;         /* On réserve la machine */
      return i;              /* Retourne le numéro de la machine allouée */
    }
}
void relache(int machine) /* Libère une machine */
{
  disponible[machine] = VRAI;
  sem_post(&nlibre);
}
```

Est-ce que ce code fonctionne correctement ? En particulier, vous détaillerez ce qui peut se passer lorsque deux clients demandent en même temps une machine à laver et vous montrerez si un problème peut se produire.

Soit vous justifierez le fait qu'il n'y a pas de problème, soit, dans l'hypothèse contraire, vous modifierez le code en conséquence, et vous donnerez les arguments nécessaires pour convaincre de la justesse de votre proposition.

Cette solution, bien qu'elle fonctionne, n'est pas la plus efficace : pendant qu'un processus exécute la boucle for, aucun autre processus ne peut y accéder. Elle est cependant suffisante pour notre problème.

Exercice 3 : Réservation de trains et sémaphores

Énoncé

La SNCF a reçu un grand nombre de réclamations pour un problème lié aux réservations et qui se produisait par grande affluence. Chaque train a un nombre limité de places. Supposons qu'un voyageur décide de changer de train. S'il commence par annuler la réservation pour le premier train et que le second soit plein, il risque de ne plus retrouver de place dans le train qu'il vient d'annuler. Cela peut arriver si un autre voyageur réserve pour le premier train dans l'intervalle. La SNCF souhaite donc mettre en place un système de permutation de réservation, permettant à un voyageur de changer de billet sans perdre son trajet initial.

On propose l'implémentation suivante :

```
#define FAUX 0
TrainT train[N];
sem_t SEMtrain[N];

void TrainEchange (VoyageurT voyageur, int v1, int v2) {
     sem_wait(SEMtrain[v1]);
     annule (train[v1], voyageur);
     if (estPlein(train[v2]) == FAUX) {
          sem_wait(SEMtrain[v2]);
          reserve (train[v2], voyageur);
          sem_post(SEMtrain[v2]);
     }
     sem_post(SEMtrain[v1]);
}
```

Les sémaphores SEMtrain[] sont binaires et initialisés à 1.

Vérifiez si l'implémentation est correcte. Si elle l'est, vous expliquerez en détail pourquoi, en montrant de quelle façon est géré le cas où deux voyageurs (ou plus) veulent accéder en même temps au système. Si elle ne l'est pas, listez et expliquez l'ensemble des problèmes, et proposez une solution opérationnelle.

Exercice 4 : Le coiffeur endormi

Énoncé

Ce problème, posé par Dijkstra, a pour cadre un salon de coiffure, où `nbChaises` sont présentes, ainsi qu'un fauteuil et un coiffeur. En l'absence de clients, le coiffeur s'endort sur son fauteuil. Lorsqu'un client arrive, il réveille le coiffeur, et celui-ci le coiffe. Les clients suivants, s'ils le peuvent, s'assoient sur les chaises, en attendant leur tour ; s'il n'y a plus de chaise disponible, ils quittent le salon.

Le problème consiste à un écrire un programme qui pilote le thread coiffeur et les threads clients en évitant les interblocages.

En termes informatiques, les clients invoquent une fonction `obtenirCoupe()` et le coiffeur une fonction `coiffer()`. Lorsque le coiffeur invoque `coiffer()`, il doit y avoir « exactement » en même temps un unique thread qui invoque `obtenirCoupe()`.