

# Programmation concurrente

## TP3 : Gestion d'un parc de vélos

### Objectif

Le but de ce travail est d'implémenter un problème de programmation concurrente qui utilise des sémaphores et des mutex.

### Description

Une ville comprend un ensemble de sites permettant à ses citoyens d'aller de site en site à vélo. Chaque site comporte un nombre fixe de bornes et chaque borne peut contenir un seul vélo. Les habitants prennent un vélo d'un site, se rendent à un autre site puis arriment le vélo à une borne du site pour un prochain usage. Chaque site a  $N$  bornes pour y placer des vélos.

De manière succincte, un habitant a le comportement suivant :

#### ***Boucle $M$ fois***

1. Attendre qu'un vélo du site  $i$  devienne disponible et le prendre.
2. Se rendre dans un nouveau site  $j \neq i$ ,  $j$  étant choisi aléatoirement. Cela prend un peu de temps, simulé par un délai  $D$ .
3. Attendre qu'une borne du site  $j$  devienne libre et rendre son vélo.
4. Faire une activité près du site  $j$ , cela dure un délai  $D$ .
5.  $i$  est remplacé par  $j$

#### ***Fin de la boucle***

$D$  doit être choisi aléatoirement entre 1000 et 1999  $\mu s$ .

Il est possible que plusieurs habitants entrent en conflit, par exemple quand ils attendent qu'une borne se libère ou qu'un vélo deviennent disponible. Dans ce cas, ceux qui ne peuvent pas prendre ou rendre leur vélo doivent attendre. Dès qu'un habitant est débloqué (par la libération ou le remplissage de l'emplacement d'une borne), il part, **sans tenir compte de l'ordre d'arrivée !**

Cette ville a aussi une équipe d'employés qui répartissent au mieux les vélos parmi les sites afin de les ramener aux destinations les moins populaires et aussi de laisser des bornes vacantes (éviter la surcharge d'un site). Cette équipe part d'un dépôt, puis circule de site en site de manière continue à l'aide d'une camionnette pouvant transporter les vélos.

Cette équipe suit l'algorithme suivant tant que tous les habitants n'ont pas effectué tous leurs trajets:

### **Boucle sur chaque site**

1. Pour  $i = 1$  à  $S$  faire
  - a. Si le site contient plus que  $N-2$  vélos, et que la capacité de la camionnette est suffisante, elle tente d'emporter 1 ou 2 vélos de façon qu'il ne reste  $N-2$  vélos dans le site.
  - b. Si le site contient moins de 2 vélos, et que la camionnette contient encore des vélos, elle tente de réapprovisionner le site pour que ce dernier en contienne 2.
2. Passer au dépôt et s'assurer rééquilibrer le contenu de la camionnette à 2 vélos : c'est-à-dire vider dans le dépôt l'excès de vélos, ou ajouter des vélos provenant du dépôt, ou ne rien ajouter, si le dépôt n'a plus de vélo.
3. Faire une pause (délai  $D_c$ )

### **Fin de la boucle**

### **Précisions :**

- Le dépôt n'a pas de limite supérieure de stockage.
- $D_c$  doit être choisi aléatoirement entre 100 et 199  $\mu s$ .
- La camionnette peut transporter **4 vélos** au maximum.
- Au départ :
  - o les habitants sont positionnés sur l'un des sites au hasard
  - o le dépôt est vide : il ne contient **aucun vélo**.
  - o la camionnette contient 2 vélos.
  - o chaque site contient  $N-2$  vélos

## **Contraintes de développement**

Comme le but de l'exercice est aussi d'exercer les principes de concurrence et de programmer « proprement », voici les points à respecter dans le cadre de ce développement :

- Le programme se termine lorsque tous les habitants ont effectué  $M$  trajets. A ce moment, affichez la somme des vélos du parc complet, c'est-à-dire ceux qui sont aux bornes, ceux de la camionnette et du dépôt. La somme doit toujours correspondre à  $S(N - 2) + 2$ .
- Le nombre de sites  $S > 2$ , le nombre d'habitants  $H > 2$ , le nombre de trajet par habitant  $M > 0$  et le nombre de bornes par site  $N > 4$  sont constants durant l'exécution du programme, mais doivent pouvoir être renseignés au lancement du programme (en ligne de commande). Un test sera effectué avec les valeurs suivantes :
  - o  $S=10, H=100, M=10, N=10$
- Chaque habitant est modélisé par un thread
- L'équipe de maintenance est modélisée par un thread
- **Toutes les attentes doivent être passives**
- **Le programme doit permettre la concurrence, donc si vous utilisez des sections critiques, elles doivent être les plus courtes possibles**

- Affichez les informations utiles lorsqu'un habitant arrive à une borne (s'il prend ou rend un vélo, s'il est bloqué, le nombre de vélos sur la borne, le nombre d'habitants bloqué, etc). Un exemple d'affichage désiré est donné en annexe.
- Ecrivez les commentaires en anglais
- Vous pouvez utiliser des variables globales, mais préférez les variables locales lorsque cela est possible
- Lorsque votre programme fonctionne, ôtez les attentes D et Dc, par exemple en désactivant la fonction d'attente en définissant une macro du type `#define uspsleep(x)` **et relancez plusieurs fois votre programme** en vérifiant:
  - qu'il n'y a pas de blocage
  - que la somme des vélos de tout le parc (dépôt compris) à la fin du programme est correcte
  - que chaque site contient  $0 \leq x \leq N$  vélos

## Rendu du travail

- **Ecrivez un mini-rapport** qui contient :
  - le nom du TP
  - vos noms, prénoms et numéro de groupe
  - le statut du projet rendu (fonctionnel ou non, bugs résiduels, etc.). Ce statut doit être exhaustif concernant le comportement du programme.
  - une explication brève et claire de votre algorithme de gestion des personnes, indiquant comment elles prennent et déposent leurs vélos, comment et sur quel principe elles attendent. Indiquez aussi de quelle façon vous gérez les aspects de concurrence dans ce problème (variables, affichage).
  - le mini-rapport doit rester court comme son nom l'indique. Ne répétez pas les éléments de cet énoncé, en revanche vous pouvez y faire référence si nécessaire. Vous pouvez ajouter des précisions qui faciliteraient la compréhension de votre code et/ou de votre algorithme si nécessaire.
- **Créez une seule archive** contenant tous les fichiers du projet ainsi que le mini-rapport et nommez-la **G<numéro\_du\_groupe>\_bykes.zip**. Ce fichier doit être rendu sur Cyberlearn.

## Annexe : exemple d’affichage à l’exécution

...

(GET) person 93 starts from terminal 9 (0 bykes, 3 persons waiting)  
(GET) person 89 starts from terminal 9 (0 bykes, 4 persons waiting)  
(GET) person 2 starts from terminal 0 (10 bykes, 3 persons waiting)  
(PUT) person 80 arrives at terminal 9 (0 bykes, 3 persons waiting)  
person 80 leaves terminal 9 by foot  
person 2 leaves terminal 0 by byke  
person 49 leaves terminal 0 by foot  
truck removes 2 bykes at terminal 0 (bykes: 10, waiting people: 1, left: in truck: 4)\*  
(PUT) person 50 arrives at terminal 0 (10 bykes, 2 persons waiting)\*  
person 50 STUCK at terminal 0: rack full\*  
(PUT) person 52 arrives at terminal 7 (5 bykes, 0 persons waiting)  
person 50 leaves terminal 0 by foot\*  
(GET) person 38 starts from terminal 9 (0 bykes, 4 persons waiting)  
person 52 leaves terminal 7 by foot  
person 38 leaves terminal 9 by byke  
(PUT) person 28 arrives at terminal 0 (10 bykes, 3 persons waiting)  
person 28 STUCK at terminal 0: rack full  
person 28 leaves terminal 0 by foot

...

person 81 leaves terminal 7 by foot  
person 81 stops  
TRUCK new loop, person's threads finished: 100  
\*\*\*\*\* CYCLING TERMINATED \*\*\*\*\*  
Terminal 0 contains 8 bykes)  
Terminal 1 contains 6 bykes)  
Terminal 2 contains 5 bykes)  
Terminal 3 contains 8 bykes)  
Terminal 4 contains 8 bykes)  
Terminal 5 contains 8 bykes)  
Terminal 6 contains 8 bykes)  
Terminal 7 contains 10 bykes)  
Terminal 8 contains 3 bykes)  
Terminal 9 contains 2 bykes)  
Total number of bykes in town: 66, in depot: 14, in truck: 2, total: 82

\* explication: le texte indique que le camion retire 2 vélos du terminal 0, mais en fait il retire « 2 personnes en attente, laissant 10 vélos dans le terminal. Cependant, les gens ne sont pas encore partis de ce terminal. Lorsque la personne 50 arrive, le terminal est encore plein et elle s’annonce comme une personne supplémentaire en attente. En réalité, comme 2 personnes « en attente » ont pu être libérées grâce au passage du camion, la personne 50 peut repartir du terminal 0.