

Laboratoire de Programmation Concurrente

semestre automne 2025

Gestion de ressources par moniteurs

Temps à disposition: 4 périodes (soit 2 semaines)

1 Objectif pédagogique

Réaliser la modélisation d'un problème concurrent par moniteur de Mesa.

2 Énoncé du problème

Une ville comprend un ensemble de sites permettant à ses citoyens d'aller de site en site en vélo. Chaque site comporte un nombre fixe de bornes et chaque borne ne peut attacher qu'un seul vélo à la fois. 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.

De manière succincte, un habitant a le comportement suivant:

Boucle infinie

1. Attendre qu'un vélo du site i devienne disponible et le prendre.
2. Aller au site $j \neq i$.
3. Attendre qu'une borne du site j devienne libre et libérer son vélo.
4. Faire une activité à pied pour se rendre à un autre site k .
5. $i \leftarrow k$

Fin de la boucle

Le système est relativement sophistiqué, et propose des vélos de différentes catégories: VTT, route et gravel. Chaque habitant a sa préférence et n'utilisera que des vélos lui correspondant. Cette préférence est définie dans le constructeur de `Person`.

Quand plusieurs habitants entrent en conflit pour une attente quelconque, par exemple quand ils attendent qu'une borne se libère ou qu'un vélo deviennent disponible, on supposera que ces habitants se servent selon leur ordre d'arrivée. Initialement, tous les habitants sont sur un site. Les sites de déplacement sont ensuite choisis de manière aléatoire.

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). Ils essaient évidemment d'équilibrer les types de vélos. Par exemple si aucun VTT n'est disponible ils en rajouteront. Cette équipe circule de site en site de manière continue à l'aide d'une camionnette pouvant transporter les vélos.

3 Cahier des charges

Réalisez le programme énoncé avec les hypothèses et les contraintes ci-dessous :

- Le nombre de sites, $S \geq 2$, et le nombre d'habitants sont constants durant l'exécution du programme. Ces nombres sont définis dans le fichier `config.h`.

- Chaque habitant est modélisé par un thread exécutant la méthode `Person::run()`.
- Le temps que les habitants prennent pour se déplacer entre deux sites, que ce soit à vélo ou à pied, sera modélisé par des temps aléatoires. Des fonctions `walk/bikeTravelTime()` sont là pour ça.
- Le nombre de bornes par site, $B \geq 4$, est aussi défini dans le fichier de configuration, et demeure invariant durant toute la durée du programme.
- La camionnette de maintenance peut contenir au plus 4 vélos (`VAN_CAPACITY`).
- Le nombre de vélos, V , doit aussi être défini dans le programme principal et doit respecter $V \geq S(B - 2) + 3$. Initialement, chaque site contiendra $B - 2$ vélos, et le solde sera déposé dans le dépôt d'où part et revient l'équipe de maintenance. Ceci est vérifié au début du `main`.
- L'équipe de maintenance est modélisée par un thread exécutant la méthode `Van::run()`.
- En notant par D le nombre de vélos au niveau du dépôt et par V_i le nombre de vélos au site i , l'équipe de maintenance aura le comportement suivant:

Boucle infinie

1. Mettre $a = \min(2, D)$ vélos dans la camionnette.
 2. Pour $i = 1$ à S faire
 - 2a. Si $V_i > B - 2$ alors
Prendre $c = \min(V_i - (B - 2), 4 - a)$ vélos du site i et les mettre dans la camionnette.
 $a \leftarrow a + c$.
 - 2b. Si $V_i < B - 2$ alors
Calculer $c = \min((B - 2) - V_i, a)$ (nombre de vélos à déposer au site i).
Initialiser $c_{\text{déposés}} \leftarrow 0$.
Pour chaque type de vélo t
 Si le site i ne contient aucun vélo de type t
 et la camionnette contient au moins un vélo de type t alors
 Déplacer un vélo de type t de la camionnette vers le site i .
 $a \leftarrow a - 1, \quad V_i \leftarrow V_i + 1, \quad c_{\text{déposés}} \leftarrow c_{\text{déposés}} + 1$.
 Si $c_{\text{déposés}} = c$ alors sortir de la boucle sur les types.Tant que $c_{\text{déposés}} < c$ et $a > 0$ faire
 Déplacer un vélo quelconque de la camionnette vers le site i .
 $a \leftarrow a - 1, \quad V_i \leftarrow V_i + 1, \quad c_{\text{déposés}} \leftarrow c_{\text{déposés}} + 1$.
 3. Vider la camionnette au dépôt : $D \leftarrow D + a, \quad a \leftarrow 0$.
 4. Faire une pause.
- Fin de la boucle

Le temps requis pour se déplacer entre les sites et aussi entre le dépôt et les sites sera un temps aléatoire, mais la durée de la pause de l'équipe sera constante. Afin de simplifier le problème, on supposera que le chargement des vélos dans la camionnette et leur déchargement se fait de manière instantanée ($= 0$).

- Le nombre de vélos peut augmenter ou diminuer à tout moment dans le dépôt. Pour cela, deux boutons sont présents dans la GUI pour en ajouter ou en retirer (1 à la fois). Vous pouvez changer le stock max si nécessaire dans le `main` ligne 60.
- La fin de la simulation doit être mise en place de votre côté. Une fonction du `main` sera appelée lorsque l'utilisateur appuie sur le bouton Stop simulation. Vous devez terminer proprement tous les threads et éviter qu'ils restent bloqués dans votre système de synchronisation. **La simulation n'a pas besoin de pouvoir être redémarrée.** Donc gérez correctement l'arrêt du Van

et des Person. Évidemment, si les personnes où le van sont dans un trajet il est pas demandé d'annuler le trajet.

- Le bouton Close simulation sert à fermer proprement l'application, mais il n'est cliquable que si la simulation a été arrêtée au préalable.
- Vos threads ne devront comporter aucune attente active, et toutes les synchronisations devront être faites par moniteurs de Mesa.
- Un squelette de programme vous est fourni. Vous y trouverez des exemples sur l'usage de l'interface graphique. N'oubliez jamais à la fin d'une méthode de mettre à jour les stocks des Bikestations avec `binkingInterface->setBikes(..)`.
- La synchronisation sera réalisée via la classe `BikeStation`, dont un squelette de code vous est fourni.
- L'utilité de chaque fonction ainsi que ce qui est attendu d'elle sont décrits dans les en-têtes correspondants.

4 Travail à rendre

- Un rapport décrivant le système implémenté ainsi que les tests effectués pour valider votre code.
- Inspirez-vous du barème de correction pour savoir là où il faut mettre votre effort.
- Vous devez travailler en équipe de deux personnes.
- Rendez votre travail avec le script de rendu présent dans le répertoire svp.
- N'oubliez pas de noter vos noms dans tous les fichiers édités et dans votre rapport.

5 Barème de correction

Conception, conformité au cahier des charges, structure et simplicité	20%
Exécution et fonctionnement(Votre code fait ce qui est attendu et résiste aux tests)	45%
Rapport (Choix, description des problèmes de concurrence, tests effectués..)	35%