HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Prof. Yann Thoma

Jonas Chapuis

Laboratoire de Programmation Concurrente

semestre printemps 2017 - 2018

Gestion de ressources

Temps à disposition : 8 périodes (travail débutant en semaine 5)

1 Objectifs pédagogiques

Réaliser un programme concurrent contenant des situations de compétition et de gestion de ressources grâce à des sémaphores.

2 Cahier des charges

Sur l'une des quatre maquettes Märklin du laboratoire, implémentez un programme en C++ qui réalise la gestion et le contrôle de deux locomotives. Chaque locomotive part depuis un point prédéterminé et réalise un parcours cyclique pour revenir à son point de départ. Après deux (2) tours, la locomotive inverse son sens de marche et effectue à nouveau le même parcours, mais en sens inverse. Ces parcours se réalisent indéfiniment. Le seul moyen d'arrêter les locomotives est d'actionner un *arrêt d'urgence*, et aussitôt actionné, les 2 locomotives devront immédiatement s'arrêter, grâce à l'appel de la procédure `mettre_maquette_hors_service`.

Une des locomotives circule selon deux tracés circulaires prédéterminés. Le premier tracé de cette locomotive est son *circuit principal*, alors que le second tracé sert de *voie de déviation*. Le tracé principal de cette locomotive et le tracé de la seconde locomotive devront comporter un tronçon commun. La voie de déviation sert à éviter l'attente de ce tronçon et elle lui sera parallèle. Ainsi, lorsque la locomotive demande le tronçon commun et que celui-ci lui est refusé, la locomotive emprunte *momentanément* sa voie de déviation. Cette locomotive ne devra donc jamais s'arrêter aussitôt qu'elle quitte son point de départ. Si cette même condition se présente pour la seconde locomotive, celle-ci n'a pas d'autre alternative que de s'arrêter jusqu'à la libération du tronçon.

En temps que chef de quai vous devez avoir la possibilité de donner une priorité absolue à une locomotive sur l'autre. Ceci se fera via l'envoi d'une commande grâce au champ texte offert dans l'interface graphique. Les commandes devront être les suivantes :

- Donner la priorité à la locomotive exploitant la voie d'évitement. L'autre locomotive ne doit plus du tout pouvoir accéder au tronçon commun.
- Donner la priorité à la locomotive n'exploitant pas la voie d'évitement. L'autre locomotive doit donc toujours utiliser la voie d'évitement.
- Supprimer le concept de priorité.

Les commandes doivent pouvoir être lancées en tout temps, et évidemment si une locomotive est sur le tronçon commun il faut qu'elle puisse en sortir sans qu'un accident ne puisse survenir. La méthode `getCommand()` vous permet de récupérer une commande entrée par l'utilisateur, mais attention, cette méthode est bloquante.

3 Remarques

— Le code de l'interface avec les maquettes, un exemple, ainsi que la documentation de cette interface sont disponibles sur cyberlearn.

— Attention, il est interdit d'utiliser la fonction `tryAcquire()` des sémaphores, de même que la fonction `available()`.

4 Travail à rendre

— Les modalités du rendu se trouvent dans les consignes qui vous ont été distribuées.

— Comme pour les laboratoires précédents, la description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans votre fichier README.md. Les fichiers sources doivent évidemment être correctement documentés et commentés. Aucun rapport n'est demandé.

— Une démonstration de votre implémentation pourra être demandée sur l'une des maquettes.

— Inspirez-vous du barème de correction pour savoir là où il faut mettre votre effort.

— Vous pouvez travailler en équipe de deux personnes au plus.

5 Barème de correction

Conception, conformité au cahier des charges et simplicité	35%
Exécution et fonctionnement	20%
Codage	15%
Documentation	20%
Commentaires au niveau du code	10%