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

Prof. Yann Thoma
Gregory Medwed
Fiorenzo Gamba

Laboratoire de Programmation Concurrente
semestre printemps 2020

Gestion de ressources

Temps à disposition : 8 périodes

1 Objectifs

— Réalisez un programme concurrent où il y a des situations de compétition et de gestion de ressources, et ce à l'aide de sémaphores.

2 Enoncé

Sur l'une des deux maquettes Märklin au laboratoire, implémentez un programme en C++ avec utilisation de sémaphores qui réalise la gestion et le contrôle de deux locomotives. Cette année nous resterons toutefois en simulation, sans passage sur maquette réelle.

Les locomotives suivent des tracés circulaires. Vous êtes libres de choisir le tracé des locomotives, toutefois il y a une contrainte à respecter : il doit y avoir au moins un tronçon commun aux deux parcours (une section partagée). Deux programmes vous sont demandés, ils seront testés les deux, mais seul le dernier sera corrigé.

2.1 Programme 1

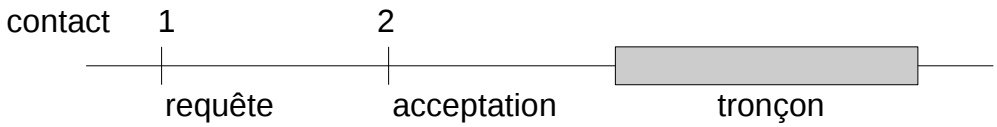
Les locomotives partent d'un point particulier de leur tracé. Chaque fois que les locomotives réalisent 2 tours complets, elles inversent leur direction et repartent pour 2 tours supplémentaires dans le sens contraire, et ceci de manière infinie. Les locomotives formulent une requête pour obtenir un tronçon partagé et, si celui-ci n'est pas disponible, la locomotive s'arrête avant le tronçon demandé. Si au contraire le tronçon demandé est immédiatement disponible, les locomotives ne devront pas s'arrêter (pas même une micro-seconde). Les locomotives ont la même priorité d'être choisies lorsqu'il faut les départager.

Votre programme devra avoir un arrêt d'urgence actionné par l'interface graphique. Cette fonctionnalité permettra d'arrêter toutes les locomotives immédiatement.

2.2 Programme 2

Modifiez le programme 1 pour y introduire un traitement prioritaire. Les deux locomotives ont une priorité distincte qui régit l'ordre de passage sur le tronçon commun. Cette priorité doit être définie dans votre code, et demeure constante. La demande du tronçon commun se réalise en 2 étapes délimitées par des contacts qui précèdent le tronçon :

contact 1 : formulation de la requête avec la priorité de la locomotive en paramètre ;
contact 2 : arrêt de la locomotive ou passage de celle-ci en cas d'acceptation.



Au moment du passage sur le contact *d'acceptation*, si une locomotive de priorité supérieure a demandé le tronçon, alors la locomotive courante doit s'arrêter. Elle doit évidemment aussi le faire si le tronçon est déjà occupé.

3 Remarques

Le code qui vous est fourni est décomposé de manière à avoir une classe qui fourni la synchronisation. Celle-ci devra dériver de l'interface suivante :

```
class SharedSectionInterface
{
public:

    enum class Priority {
        LowPriority,
        HighPriority
    };

    virtual void request(Locomotive& loco, Priority priority) = 0;
    virtual void getAccess(Locomotive& loco, Priority priority) = 0;
    virtual void leave(Locomotive& loco) = 0;
};
```

Il est impératif de respecter cette interface, car les tests du code l'utiliseront tel quel. Pour le programme 1, la fonction `request()` n'est pas nécessaire, et ne sera donc pas utilisée, les locomotives n'utilisant que `getAccess()` et `leave()`.

- Le code du simulateur de maquettes ainsi que la documentation du simulateur et des maquettes est fourni.
- Le projet à ouvrir dans QtCreator est `QtrainSimStudent.pro`. Il s'agit d'un projet décomposé en 2 sous-projets : prog1 et prog2. Vous devez développer le programme 1 dans prog1 et le programme 2 dans prog2. Le script de génération de l'archive récupérera les deux codes.
- Lors de votre première compilation il faut aller dans le dosser de build et lancer la commande `make install` avant de pouvoir lancer l'application. Ceci peut aussi être fait depuis QtCreator (cf. documentation de QTrainSim).
- Afin de faire rentrer ou sortir les locomotives du tronçon commun (section partagée) il vous faudra commander les aiguillages de manière appropriée.
- Ne pas modifier les fichiers `locomotive.h/cpp`, `launchable.h` et `sharedsectioninterface.h`. Vous pouvez créer de nouveaux fichiers ou classes pour gérer vos parcours au besoin.
- L'arrêt d'urgence est une nécessité pour tout système critique où un arrêt peut être réalisé. Dans le présent laboratoire, une manière simple et triviale de réaliser cet arrêt consiste à faire un appel à la procédure `mettre_maquette_hors_service`. Ceci équivaut à couper toute l'alimentation du réseau ferroviaire. Dans le contexte de ce laboratoire, il est demandé d'arrêter les deux locomotives sans utiliser cette fonction.
- Vous devrez nous rendre les deux programmes complets, selon les consignes données en annexe. Seul le code du programme 2 sera relu, mais les deux programmes seront évalués en exécution.
- La description de l'implémentation, ses différentes étapes, la manière dont vous avez vérifié son fonctionnement et toute autre information pertinente doivent figurer dans un petit rapport rendu avec le code. Celui-ci se nommera `rapport.pdf` et se toruvera au même niveau que le script `pco_rendu.sh`, ce dernier servant à générer l'archive à rendre sur Cyberlearn.
- Inspirez-vous du barème de correction pour savoir là où il faut mettre votre effort.

— Vous pouvez travailler en équipes de deux personnes.

4 Barème de correction

Conception et conformité à l'énoncé	25%
Exécution et fonctionnement (10% pour programme 1 et 15% pour programme 2)	25%
Codage	15%
Documentation et en-têtes des fonctions	25%
Commentaires au niveau du code	10%