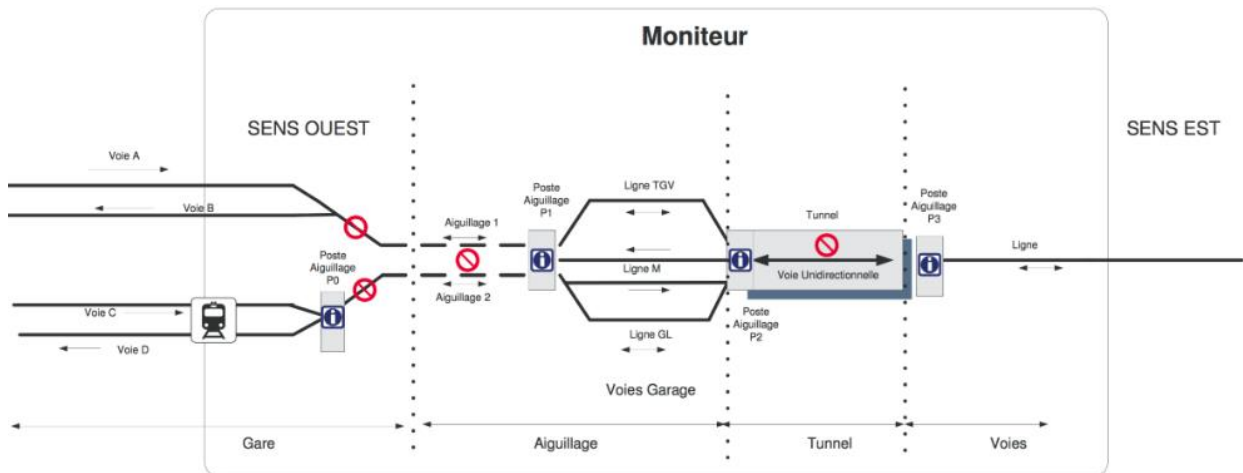


LO41 - Trains



Introduction

Dans le cadre de l'UV LO41, nous avons été amenés à travailler sur un projet. Ce projet a pour but de consolider nos connaissances en les appliquant à un cas d'étude concret. Il consiste en la gestion de trains arrivant et partant d'une plateforme de routage telle que ci dessous afin que chaque train puisse aller vers sa destination sans entrer en collision avec un autre.



Pour simuler cette situation, nous devons utiliser des moyens tels que les threads, les sémaphores, les moniteurs, la mémoire partagée, l'interblocage ou encore les signaux.

L'implémentation

Pour traiter ce sujet nous avons choisi d'utiliser des threads, des moniteurs, des signaux ainsi que des sémaphores.

Dans un premier temps, nous avons choisi de mettre la planification des trains dans un fichier (trouvable dans le dossier "tests"), afin que tous les trains prévus pour la journée soient réunis et modifiables facilement. Nous pouvons donc ajouter des scénarios à notre convenance (pour ajouter un nouveau scénario, veuillez vous référer au fichier README).

Ces informations sont ensuite récupérées par notre programme grâce à un lecteur de fichier qui va parcourir chaque ligne du fichier source afin de créer un tableau avec toutes les informations nécessaires (le nombre de colonnes sera le nombre de trains, et pour chaque train on aura le numéro du train, son point de départ, son point d'arrivée et son type).

Ce tableau nous permet par conséquent de savoir combien de trains arrivent par la voie EST, la voie A et la voie C, nous avons donc trois boucles créant les threads associés aux différents points de départ (chaque thread correspondant à un train). Ces trains reçoivent en argument à leur création un pointeur sur la colonne du tableau correspondant à leurs données personnelles.

La dernière étape de cette initialisation consiste à la vérification des trains de chaque file grâce à un affichage ainsi qu'au réveil du Superviseur qui va commencer à effectuer son travail. Ce dernier a en charge le bon déroulement de la journée en faisant passer chaque train de façon à ce qu'aucun train n'entre en collision avec un autre.

Les trains sont donc représentés par des threads, ils suivent chacun la voie qui leur est attribuée par le Superviseur. Au départ ils sont donc dans la file d'attente de leur voie et se mettent en attente à l'aide d'un moniteur. Une fois réveillés par le Superviseur, ils se retirent de leur file d'attente, et partent pour leur destination jusqu'au prochain point d'arrêt ou ils vont réveiller le Superviseur. Le Superviseur va ensuite décider quel train va devoir passer, il va se rendormir et se fera à nouveau réveiller pour choisir le prochain train et ainsi de suite jusqu'à ce que tous les trains aient atteint leur destination.

Le superviseur va donc décider de l'ordre de passage des trains. Pour ce faire il dispose d'informations sur les voies, notamment les trains qui s'y trouvent, ainsi que d'un algorithme pour décider quel train doit passer le premier pour que tous puissent atteindre leur destination. Il est représenté par un thread qui tourne en boucle jusqu'à ce qu'il n'y ait plus de trains dans les files d'attentes. A chaque itération, il attend de se faire réveiller par un train, ensuite il vérifie les files d'attente, choisit et fait passer le train le plus prioritaire (en le réveillant) et enfin se rendort.

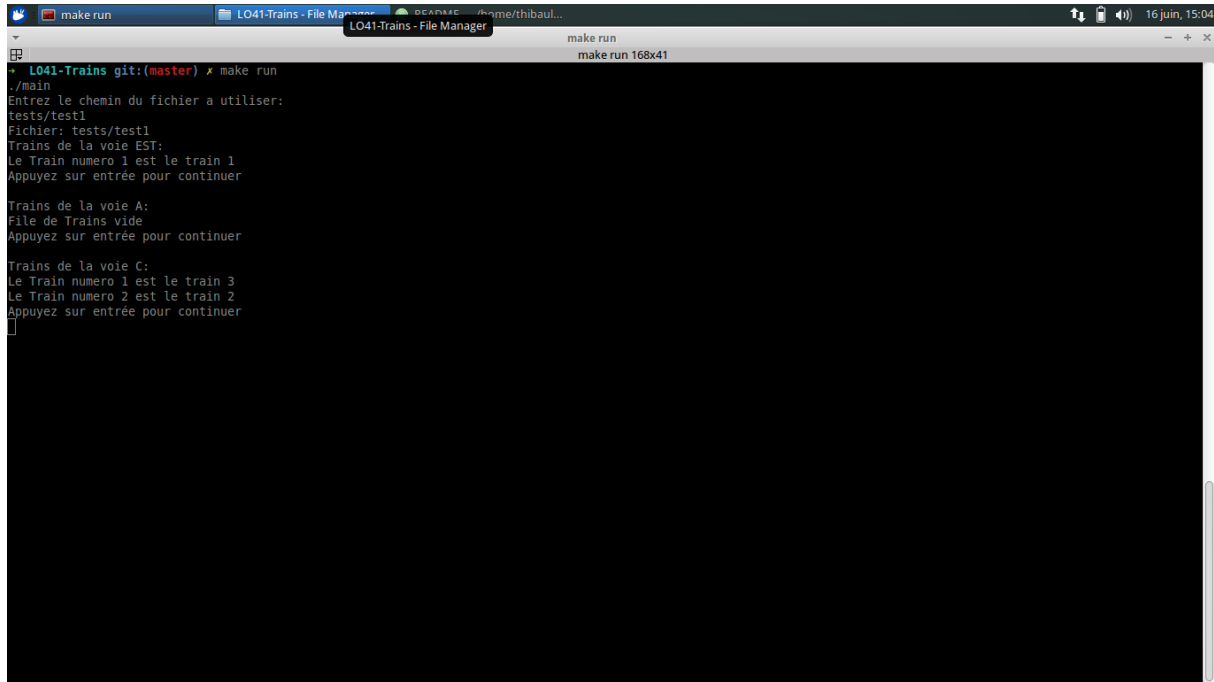
Ces files d'attentes sont représentées par des listes chaînées. A chaque fois qu'un train arrive, il va se rajouter en fin de liste, et à chaque fois qu'un train quitte la section, il sort par la tête de la file (ce qui nous assure un traitement des trains par FIFO).

Une fois tous les trains passés, les tables IPC sont nettoyées, le programme principal attends que tous les threads soient terminés (avec la fonction `pthread_join`) et se termine à son tour.

Vous pouvez lancer le programme se lance à l'aide du script "run.sh" (qui utilise un Makefile).

Résultats

Voici les captures d'écran d'un exemple d'utilisation du programme. En premier lieu nous avons l'initialisation avec la vérification des différents trains sur les différentes lignes, et ensuite le fonctionnement pas à pas de l'application.

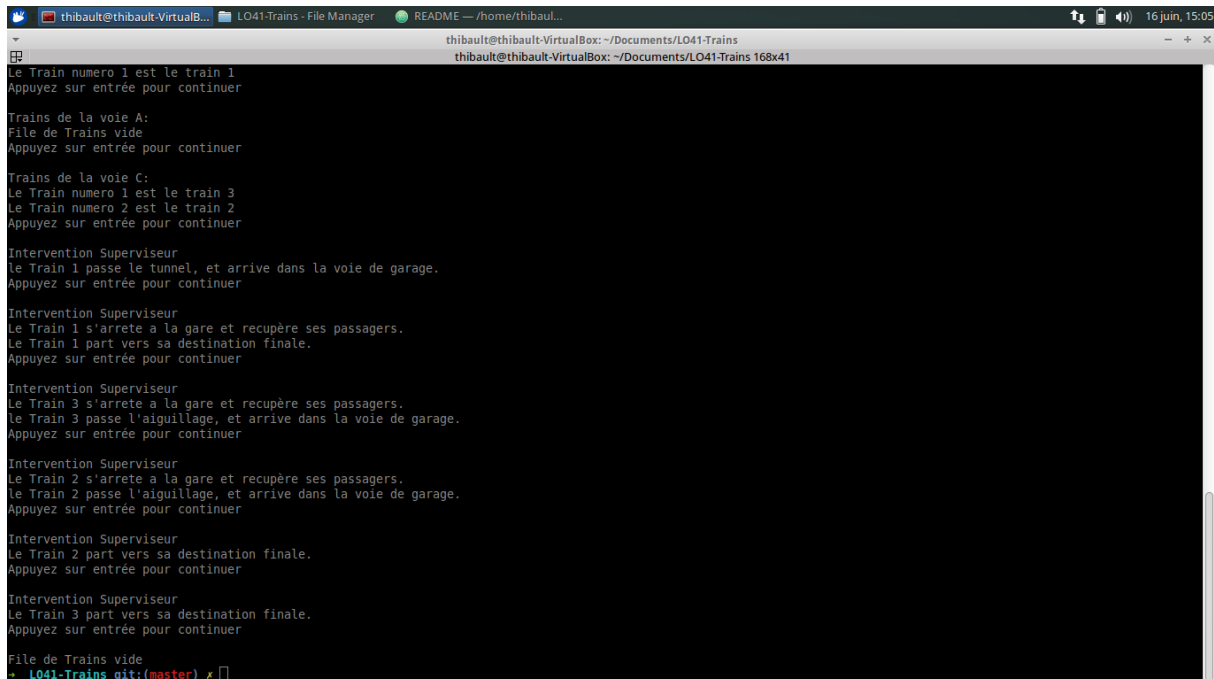


```
make run
make run 168x41

+ LO41-Trains git:(master) x make run
./main
Entrez le chemin du fichier à utiliser:
tests/test1
Fichier: tests/test1
Trains de la voie EST:
Le Train numero 1 est le train 1
Appuyez sur entrée pour continuer

Trains de la voie A:
File de Trains vide
Appuyez sur entrée pour continuer

Trains de la voie C:
Le Train numero 1 est le train 3
Le Train numero 2 est le train 2
Appuyez sur entrée pour continuer
```



```
thibault@thibault-VirtualBox: ~/Documents/LO41-Trains
thibault@thibault-VirtualBox: ~/Documents/LO41-Trains 168x41

Le Train numero 1 est le train 1
Appuyez sur entrée pour continuer

Trains de la voie A:
File de Trains vide
Appuyez sur entrée pour continuer

Trains de la voie C:
Le Train numero 1 est le train 3
Le Train numero 2 est le train 2
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 1 passe le tunnel, et arrive dans la voie de garage.
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 1 s'arrete a la gare et recupere ses passagers.
Le Train 1 part vers sa destination finale.
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 3 s'arrete a la gare et recupere ses passagers.
Le Train 3 passe l'aiguillage, et arrive dans la voie de garage.
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 2 s'arrete a la gare et recupere ses passagers.
Le Train 2 passe l'aiguillage, et arrive dans la voie de garage.
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 2 part vers sa destination finale.
Appuyez sur entrée pour continuer

Intervention Superviseur
Le Train 3 part vers sa destination finale.
Appuyez sur entrée pour continuer

File de Trains vide
+ LO41-Trains git:(master) x
```

Problèmes rencontrés

Pendant le développement de notre programme nous avons été confrontés à plusieurs problèmes:

La contrainte de temps en premier lieu, nous avons donc dû nous focaliser sur un programme fonctionnel et non un programme optimisé, néanmoins, le programme répond aux attentes du sujet.

Ensuite, le choix d'avoir un fichier séparé représentant le planning des trains pour la journée, un lecteur de fichier étant compliqué à réaliser en C, nous avons fait au mieux pour avoir une récupération de données fonctionnelle prenant le moins de temps possible (d'où le fait d'avoir une information par ligne).

Puis, nous avons dû rechercher dans d'anciennes connaissances afin de mettre en place des listes chaînées, travailler avec des pointeurs et des structures, ce qui peut être confus dans certains cas. Malgré tout nous avons réussi à présenter des fonctions lisibles et compréhensibles surtout concernant la gestion des files d'attente des différentes voies.

Pour finir, la partie test, débogage et finalisation est celle qui nous a pris le plus de temps, afin de bien gérer les différents comportements du programme, nous avons dû procéder à une batterie de tests avant qu'il soit vraiment fiable. Nous avons notamment rencontrés des erreurs dans les files d'attente avec les pointeurs et dans le passage du tableau en argument des threads, mais ces problèmes mineurs ont été résolus assez facilement.

Pistes d'amélioration

Concernant les pistes d'amélioration, on peut noter les suivantes:

Une amélioration du système de récupération de données à partir du fichier, mettre toutes les informations concernant un même train sur une seule ligne afin de faciliter la création ainsi que la lecture de celui-ci. Voir même un système interne au programme pour ajouter des trains à la volée (en cas de reroutage d'un train à la dernière minute) ou d'entrée par l'utilisateur au début du programme.

L'ajout d'un menu pour rendre le programme plus agréable et plus intuitif, nous permettant de choisir peut être le mode de fonctionnement (génération de trains aléatoires ou planification par fichier comme actuellement). On pourrait également ajouter des éléments graphiques pour la représentation des déplacements. Au lieu de simples lignes décrivant l'action du train, on pourrait avoir un affichage en temps réel des déplacements.

L'algorithme du Superviseur peut lui aussi être amélioré, en effet, on pourrait utiliser un algorithme d'optimisation de trajets (algorithmes utilisant des méta-heuristiques) afin d'avoir le meilleur trajet en terme de temps et d'attente (ici on implémenterait un problème d'optimisation multicritères).

Nous pourrions améliorer le code en termes de lisibilité en séparant le code en plusieurs parties et donc en plusieurs fichiers, ceci permettrait une lecture du code plus agréable ainsi que la possibilité de réutiliser certaines parties du code pour des applications ultérieures.

Ensuite, nous pourrions fournir un mode de test de performances. On pourrait générer un grand nombre aléatoire de trains avec des lieux de départ et d'arrivée aléatoires afin de voir la réaction du programme dans des situations extrêmes, ce qui aurait un grand intérêt dans le cas de l'implémentation d'un algorithme d'optimisation où on pourrait voir les résultats en termes de temps et d'efficacité.

Pour finir, nous pourrions ajouter un paramètre supplémentaire: le temps. En effet, en ajoutant un temps d'arrivée dans la section et un temps limite de sortie, on aurait un challenge en plus qui serait de faire transiter les trains de manière à respecter cette contrainte.

Conclusion

Malgré les difficultés rencontrées, le programme est fonctionnel et répond aux exigences du sujet. En effet, les trains ne se retrouvent jamais en situation de collision ou de telle sorte à bloquer toute l'infrastructure, ils partent tous de leur point de départ et arrivent à leur point d'arrivée sans incidents, en affichant les différentes étapes de leur trajet.

Ce projet nous a permis d'appliquer nos connaissances dans un cadre concret, en effet, nous avons pu revoir une bonne partie du programme de LO41 incluant les moniteurs, les sémaphores, les files d'attente et les signaux entre autres.

Néanmoins, ce projet peut encore être amélioré comme nous avons pu le voir précédemment, ajouter une interface plus engageante vis à vis de l'utilisateur ou encore optimiser le code en lui-même ainsi que l'algorithme de sélection du Superviseur.