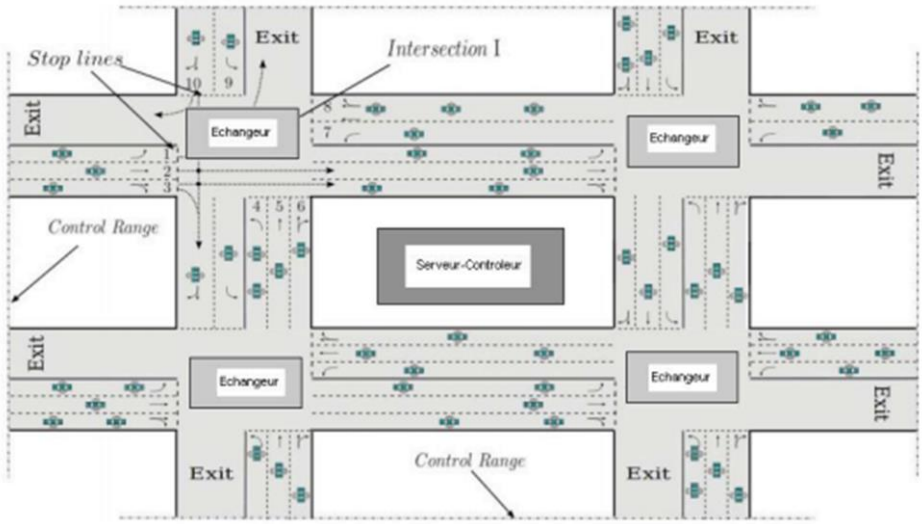
UNIVERSITE DE TECHNOLOGIE DE BELFORT-MONTBELIARD

PROJET DE LO41

Simulation d'un ensemble de 4 carrefours



12/06/2015

ETUDIANTS

Nicolas BOUFFARD (Etudiant en Info 02)

Arnel ZEUFACK (Etudiant en Info 02)

Table des matières

I. Introduction

Le projet de LO41 est destiné à faire appréhender par la pratique à l'étudiant divers concepts de la programmation système, tels que la gestion de la mémoire, la programmation concurrentielle, la synchronisation, ainsi que divers mécanismes de fonctionnement des systèmes d'exploitation. C'est dans cette optique qu'il nous a été proposé ce semestre un projet intitulé «Gestion d'un ensemble de 4 carrefours».

Le projet consiste à simuler le déplacement des véhicules sur un ensemble de 4 carrefours (échangeurs), dépourvus de feux de signalisation, tout en évitant que les véhicules n'entrent en collision. Le véhicule devra traverser une voie selon un système d'autorisations reçues par son échangeur. Ce dernier transmettant les requêtes à un serveur contrôleur qui régule la circulation en fonction de l'état du trafic et de la position des véhicules sur l'ensemble des 4 carrefours.

Afin de respecter ce cahier des charges, nous essayerons dans un premier temps de nous faire une conception générale du projet, pour ensuite faire le point sur la démarche à adopter et les différents concepts de la programmation système à employer.

II. Organisation de l'application

1. Problématique

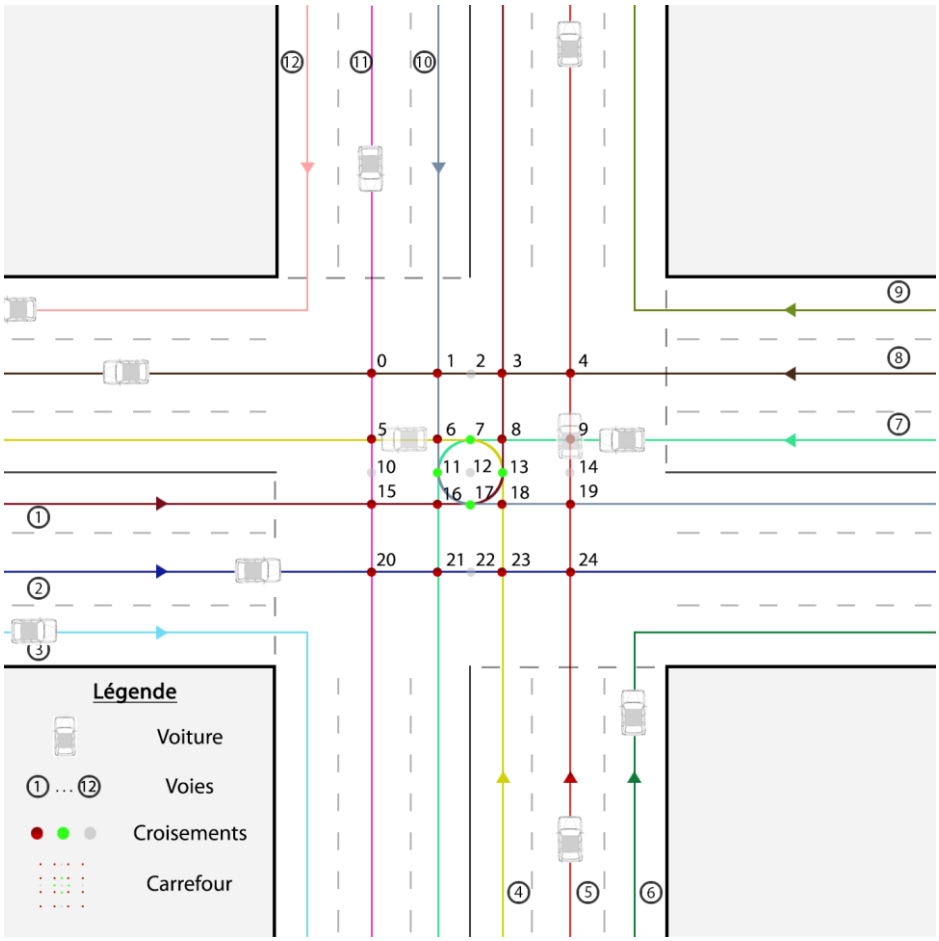
La partie la plus importante du projet a été d'analyser le problème posé, afin de pouvoir proposer une solution adaptée, optimisée, et répondant aux différentes restrictions:

- l'absence de feux de signalisation
- les carrefours reçoivent les requêtes des voitures et les transmettent au serveur-contrôleur, puis renvoient les réponses de ce dernier aux voitures
- le serveur-contrôleur traite les requêtes et renvoie les réponses aux carrefours

2. Idée adoptée

a. Représentation d'un carrefour

Après analyse des différentes situations possibles, nous sommes arrivés à une généralisation du problème, que nous illustrons par le schéma ci-dessous pour un seul carrefour :



On distingue divers types d'informations :

- 1 carrefour
- 12 voies
- 25 croisements
- des voitures

Le carrefour est composé des 25 croisements, qui représentent des intersections entre les 12 voies. On s'aperçoit que les croisements sont eux-mêmes séparés en 3 catégories :

- les points rouges
- les points verts
- les points transparents

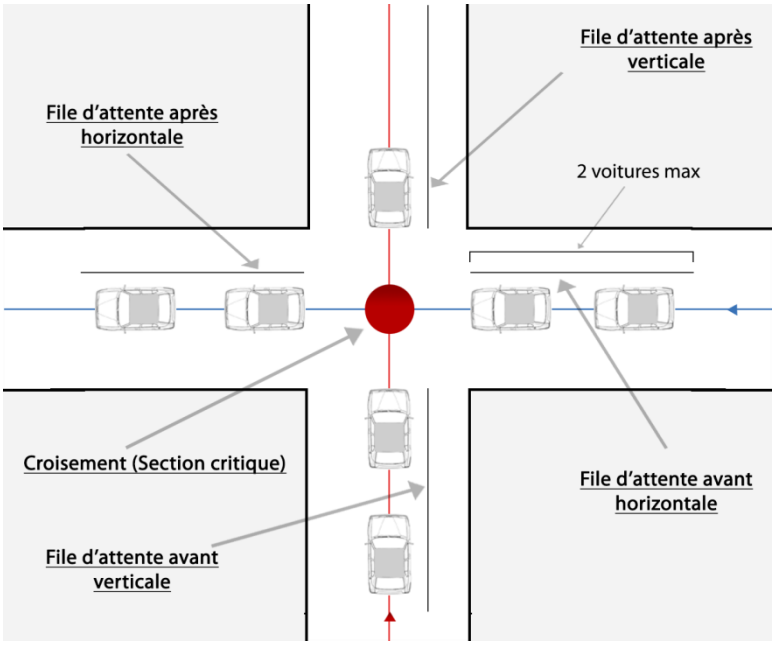
Si l'on analyse le tracé des voies, il vient que l'on peut former une matrice 4x4 dont chaque élément est une intersection entre 2 voies : les points rouges.

Il reste cependant 4 points d'intersection qui ne figurent pas dans la matrice : les points verts.

Afin de simplifier la représentation au sein du programme, nous avons décidé d'ajouter 4 points supplémentaires, pour se retrouver à nouveau avec une matrice carrée, cette fois ci de taille 5x5 : les points transparents. Ces points n'interviennent pas dans le système de requête - réponse, puisqu'il n'y a jamais plus d'une voie qui les croise. Ils favorisent cependant la compréhension et simplifient dans certaines fonctions de test.

b. Représentation d'un croisement

Chaque croisement peut être vu de la façon suivante :



On distingue 4 files d'attente :

- une avant le croisement, sur la voie verticale
- une après le croisement, sur la voie verticale
- une avant le croisement, sur la voie horizontale
- une après le croisement, sur la voie horizontale

Un certain nombre de voitures peuvent se trouver dans les files d'attentes à un instant t. C'est cette limite qui dimensionne la taille du carrefour.

Le croisement en lui-même (le point entre les files d'attente) représente la zone critique : il ne peut y avoir qu'une seule voiture qui traverse un croisement à un instant t, le contraire reviendrait à une collision.

c. Solution de gestion des priorités

Afin d'assurer qu'aucune collision n'occure, les voitures appliquent l'algorithme suivant :

Pour chaque croisement de la voie où se trouve la voiture
    Faire
        Demander au serveur la permission d'arriver dans la file d'attente avant le croisement
    Tant que la permission est refusée

    Entrer dans la fille d'attente avant le croisement

    Faire
        Demander au serveur la permission de traverser le croisement
    Tant que la permission est refusée

    Traverser le croisement

    Faire
        Demander au serveur la permission d'arriver dans la file d'attente après le croisement
    Tant que la permission est refusée

    Entrer dans la fille d'attente après le croisement
Fin de pour chaque croisement

Les voitures sont donc stoppées à chaque instant dans l'attente d'une autorisation du serveur. Elles avancent ainsi par étape, uniquement si la situation le permet (place disponible dans les files d'attente / croisement libre).

d. Fonctionnalités utilisées

Le programme utilise différentes fonctionnalités vues en cours pour remplir ses objectifs :

- des files de messages pour communiquer entre les voitures, les carrefours, et le serveur-contrôleur.
- des segments de mémoire partagée pour que le processus serveur puisse accéder aux informations d'état des 4 carrefours qui auront été modifiées par les processus carrefours.
- un sémaphore pour synchroniser certaines opérations d'affichage et l'accès aux variables globales et segments de mémoire partagée.

III. Outils utilisés

Afin de mener le développement de notre projet dans les conditions les plus favorables nous avons utilisé divers outils logiciels entièrement gratuits et open source.

1. Gestionnaire de Version

- **Git**

Crée par Linus Torvalds, il permet de gérer l'évolution du contenu d'une arborescence et facilite le travail de groupe en accélérant le processus de développement déroulement du projet.

2. Editeurs de textes

Nous avons choisi de nous abstenir des éditeurs classiques orienté console tel que vi, et avons privilégié des logiciels plus aisés et possédant plus de fonctionnalités tels que SciTE et Gedit.

- **SciTE**

Crée par Neil Hodgson et placé sous une licence libre peu connue (Historical Permission Notice and Disclaimer), SciTE est un éditeur de texte graphique, gratuit et open source fonctionnant sous les environnements Linux et Windows. Il convient bien aux langages tels que C, C#, Perl, Java, Ruby, et même HTML.

- **Gedit**

Gedit est un éditeur de texte libre (sous licence GPL) Il est fourni par défaut avec l'environnement graphique GNOME. Il a été conçu à l'aide de la bibliothèque GTK de façon à avoir une interface graphique simple et propre, inspiré des idéaux du projet GNOME.

3. Générateur de documentation

- **Doxygen**

Nous avons jugé utile de générer une documentation pour le code source du projet. Pour cela nous nous sommes tournés vers Doxygen qui est un outil de génération de documentation entièrement gratuit et open source, capable de produire une documentation logicielle à partir du code source d'un programme. Pour cela, il tient compte de la grammaire du langage dans lequel est écrit le code source, ainsi que des commentaires s'ils sont écrits dans un format particulier. Il est capable d'analyser les fichiers sources écrits dans divers langages : C, Objective C, C#, PHP, C++, Java, Python, IDL, Fortran, VHDL, Tcl.

4. Outil d'édition d'image

- **Gimp :**

Est un logiciel d'édition et de retouche d'images diffusé sous la licence GPLv3 également gratuit et libre. Nous l'avons utilisé dans ce projet pour la réalisation de schémas illustrant notre conception du carrefour et des collisions.

IV. Difficultés rencontrées

1. Mise en place des idées

La plus grande difficulté aura été d'arriver à une représentation valide du carrefour et un système de gestion des priorités fonctionnel, garantissant un risque zéro concernant les collisions. Une fois ce système défini, un certain nombre de problèmes sous-jacents purement liés à la programmation sont survenus. Des tests et investigations ont été menés pour déceler et corriger les sources d'erreurs.

2. Extension à 4 carrefours

La seconde difficulté aura été d'étendre le problème à 4 carrefours, car nous avons dans un premier temps développé cette dernière avec un seul carrefour, qui plus est sans l'aspect serveur-contrôleur (les carrefours faisaient office de serveurs). Tout la mise à niveau a été effectuée en une étape, impliquant de nombreux problèmes et difficultés dont les causes n'étaient pas toujours évidentes.

V. Améliorations possibles

1. Files d'attentes imprécises

En l'état actuel, si l'aspect des files d'attente est présent autour des croisements, rien ne garantit leur continuité : les voitures arrivent dans ces files, mais la première voiture à y être entrée ne sera pas forcément la première à en sortir. Cela est dû au fait que les files sont vues comme un simple compteur, comptabilisant le nombre de voitures aux points en question, et non comme de vraies listes.
Une amélioration possible corrigerait donc cela et assurerait que les voitures soient bien en file et ne puissent pas se dépasser.

2. Absence de files d'attentes aux entrées / sorties des voies

Si des files d'attente ont été implémentées au niveau des croisements, rien n'a été fait aux alentour. Ainsi, une infinité de voitures peuvent théoriquement s'amasser aux entrées et sorties des voies, ce qui n'est pas représentatif de la réalité.
Une amélioration possible implémenterait un système de files pour les voies également.

3. Fluidité irréelle

La solution adoptée implique que les voitures s'arrêtent constamment pour demander l'autorisation d'avancer au serveur-contrôleur. Cette solution fonctionne mais n'est pas optimale, puisqu'en réalité les voitures ne s'arrêtent pas (ni ne redémarrent) instantanément.
Une solution serait d'anticiper les demandes au serveur avant l'arrivée aux sections critiques, pour s'arrêter si aucune réponse n'a été reçue, ou une réponse négative. Cette solution peut être couplée avec l'implémentation de vitesses, d'accélérations, et de décélérations pour les voitures.

4. Véhicules prioritaires

En raison des délais et d'autres impératifs, certains points du sujet n'ont pas pu être traités, comme l'aspect prioritaire de certains véhicules (ambulances, etc…).
Une amélioration mettrait en place un système de priorités avancées selon les types de véhicules.

5. Interface

L'affichage actuel est textuel et peu clair, surtout lorsque beaucoup de véhicules circulent en même temps sur les carrefours.
Une amélioration verrait l'implémentation d'une interface visuelle, sous la forme d'une animation par exemple, qui montrerait le déplacement des voitures sur l'ensemble des carrefours en temps réel.

6. Multi-sémaphore

Un seul sémaphore est utilisé pour l'ensemble du programme. Or l'affichage est indépendant des accès aux segments de mémoire partagée. De plus, les 4 segments pour les carrefours sont indépendants les uns des autres. Le fait qu'un seul sémaphore soit utilisé bloque certains processus inutilement dans bon nombre de situations.
Une amélioration ajouterait des sémaphores afin de ne bloquer des processus qu'en cas de réelle nécessité.

7. Protection des variables

La manière dont a été codé le programme ne garantit pas la sécurité des variables à l'heure actuelle. En effet, certaines variables sont accessibles à des processus qui n'en ont pas l'utilité et pourraient les modifier sans autorisation.
Une amélioration isolerait ces variables de manière efficace.

VI. Conclusion

Arrivés au terme de notre projet, nous avons pu mettre en pratique plusieurs notions importantes de la programmation système tels que les processus, les objets IPC (files de message, mémoire partagée, synchronisation, sémaphores,…). Ce projet nous aura également appris que la programmation de bas niveau inclut un grand nombre d'aspects assez complexes qui ne sont pas toujours évidents à contrôler.
Malgré quelques difficultés rencontrées dans de ce projet, notamment lors de l'élaboration de la méthode à adopter et de la conception de la solution en elle-même, nous avons néanmoins su exploiter les connaissances acquises tout au long de l'UV et avons pu mettre en œuvre une solution fonctionnelle.
En dépit d'un affichage qui ne permet pas de vérification efficace lorsqu'un trop grand nombre de véhicules sont en concurrence, la solution proposée est satisfaisante et rempli les contraintes imposées. Cela semble valable dans la limite d'une centaine de véhicules sur l'ensemble des 4 carrefours, ce qui reste plausible dans la réalité.
Ce projet fût pour nous l'occasion idéale de nous confronter à de réelles problématiques de la programmation concurrentielle et nous a permis d'acquérir de réelles compétences dans le domaine de la programmation système.