Projet de LO41
Gestion des déchets

Florian CAULET
Johann NOVAK

18 juin 2014

Chapitre 1

Description du problème et attentes personnelles

1.1 Description du projet

La problématique que nous avons dû traiter est celle de la gestion de déchets. Il s'agit de réaliser, en langage C, une **simulation** de vie urbaine où des utilisateurs consomment et jettent des déchets dans les poubelles qu'ils louent à la mairie. Ils peuvent soit avoir un **bac** individuel où eux seuls peuvent jeter des déchets, soit une **clé** permettant d'ouvrir un conteneur public, soit les **deux** en même temps. Des conteneurs dédiés au déchets recyclables sont quant à eux disponibles pour tout le monde.
Dans cette ville, les individus sont facturés en fonction du **volume** de déchets jetés. Ainsi, plus ils jettent, plus ils paient Cependant, les déchets jetés dans les conteneurs recyclables ne sont pas comptabilisés dans la facture. Dans ce projet, les poubelles peuvent **communiquer** directement avec le service de ramassage ainsi qu'avec la mairie via des puces électroniques, notamment lorsqu'elles sont pleines, ce qui permet de transmettre à la mairie le volume de déchets jetés à **chaque utilisation**. Le poids transmis vient alors s'ajouter au poids déjà jeté par ce même utilisateur. Au bout du mois, une **facture** est envoyée à chaque client.
En ce qui concerne le ramassage des poubelles, comme nous l'avons dit, la poubelle est capable de prévenir le service de ramassage lorsqu'elle est pleine et demande la **venue immédiate** d'un camion poubelle. Ainsi, lorsque ce signal est reçu, le premier camion poubelle **libre** ira vider cette dernière avant de retourner au service de ramassage.
Le but de ce projet est de civiliser et d'organiser toute cette population pour que tout puisse se faire dans le **bon ordre** sans violence, triche, ni blocage. Le dépôt sauvage d'ordure est également interdit.

1.2 Analyse du projet

Nous avons réfléchis sur les différentes **règles** que devra respecter notre projet, et avons défini les règles suivantes :

- → Un utilisateur sera représenté par une **famille**. Elle peut comporter de 1 à 13 individus.
- → Les utilisateurs sont différenciés via un système d'IDs.
- → D'après la moyenne nationale de consommation, un individu jette **1L** de déchets par jours. Ainsi, une famille jettera n déchets par jour, avec n = nombre de personnes de la famille.
- → Les utilisateurs qui ont une **clé** possèdent des sacs poubelle d'un volume de **30L**. Lorsque l'utilisateur en arrive à remplir son sac, alors il ira le jeter dans le **conteneur public** qui lui a été assigné.
- → Pour les utilisateurs qui possèdent un bac individuel, la taille de ce dernier est **variable** et dépend du nombre d'individus dans la famille. S'il n'y a qu'une personne, il sera de $80L$. S'il y a 2 personnes, il sera de $120L$, S'il y a 3 ou 4 personnes, il sera de $180L$. Sinon pour 5 personnes ou plus, il sera de $240L$.
- → Un utilisateur ne peut **pas jeter au-delà** de la capacité de stockage de son bac individuel. Si ce dernier est plein, alors il se retrouve dans l'impossibilité de jeter ses déchets et devra

attendre l'arrivée d'un camion poubelle pour pouvoir jeter et donc pouvoir à nouveau consommer.

→ Un utilisateur possédant à la fois une **clé et un bac** individuel préférera jeter dans son bac jusqu'à ce qu'il soit plein. Lorsque ce cas survient, en attendant qu'il soit vidé, il utilisera les sacs de $30L$ qu'il pourra jeter dans un conteneur public. Une fois son bac vidé, l'utilisateur **transfère** le contenu du dernier sac non-vide dans son bac avant de recommencer son cycle de consommation.

→ Un utilisateur qui utilise un conteneur public occupe ce dernier et ne peut donc être dérangé par aucun autre utilisateurs. Les utilisateurs doivent donc **attendre** leur tour les uns après les autres.

→ Une poubelle ou un conteneur public est **inaccessible** lorsqu'un utilisateur est en train de l'utiliser ou si un camion poubelle est en train de le/la vider.

→ Un utilisateur qui arrive à un conteneur public et qui observe qu'il est plein s'en ira de lui-même.

→ Toutes les poubelles (hormis les sacs et les conteneurs recyclables) transmettent à la mairie le **litrage** exact de déchets qui vient d'être jeté pour permettre de créer la facture mensuelle de l'utilisateur.

→ Un camion poubelle ne peut aller chercher **qu'une seule poubelle** ou conteneur à la fois, en un seul trajet. Ainsi, lorsqu'il est en déplacement, le service de ramassage ne peut pas faire appel à lui.

→ Lorsque qu'un bac ou un conteneur public est **plein**, il envoie un signal au service de ramassage avec son adresse afin d'être vidé. Une fois que le service de ramassage a trouvé un camion disponible, il l'envoie à l'adresse de la poubelle. Ce dernier, une fois arrivé vide la poubelle avant de rentrer au service de ramassage.

→ Des conteneurs de types spéciaux sont attribués pour le recyclage, permettant respectivement d'accueillir soit du **papier/carton**, du **plastique** ou du **verre**. Des camions spéciaux seront également à disposition de chacun de ces types de conteneur.

→ L'unique rôle de la mairie est donc de recevoir des messages de la part des conteneurs, signalant l'ID de l'utilisateur et le litrage jeté afin de construire les factures mensuelles. Cependant, la mairie s'occupe de la mise service du service de ramassage et de tous les abonnements, elle pourrait donc si elle le souhaitait accéder à n'importe quelle information du système.

1.3 Attentes et objectifs

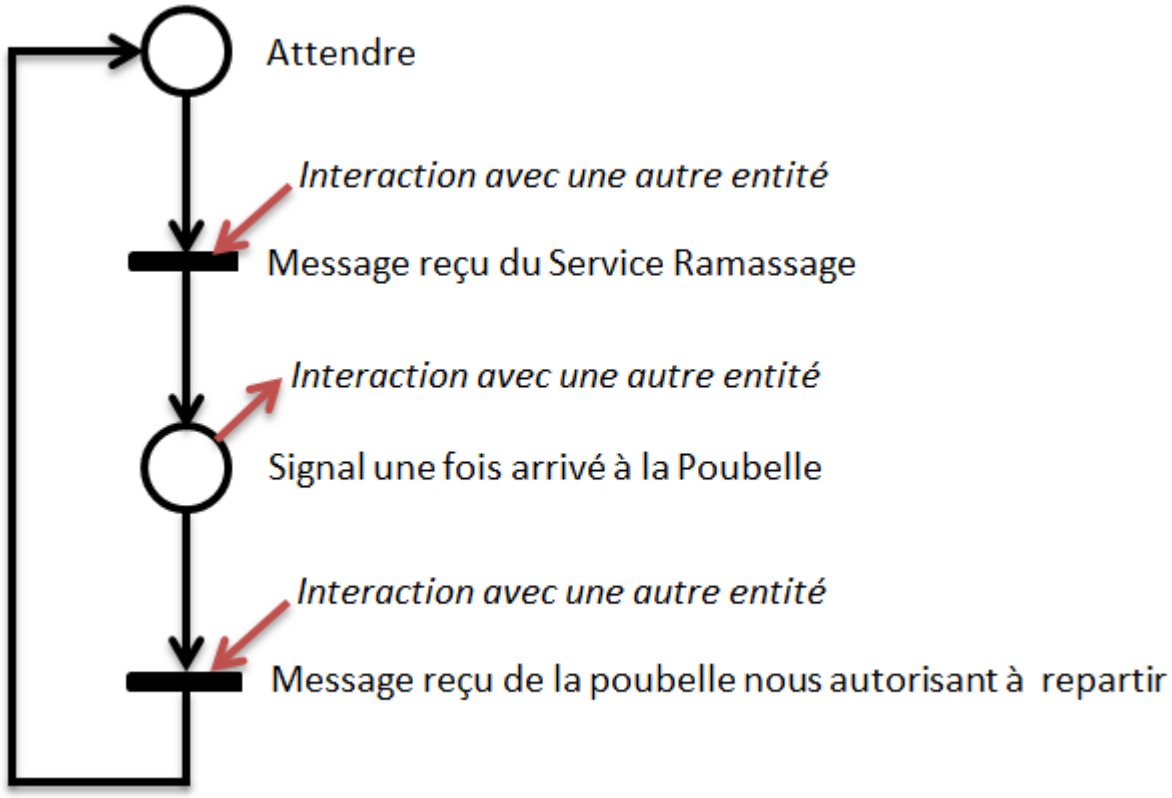
L'objectif que nous nous sommes fixé au début de ce projet était de bien **maîtriser** tout ce qui s'apparente à de la synchronisation. Pour nous deux c'était une **première** expérience de gestion de mémoires critiques (partagées) et des cas d'inter-blocages, et ainsi une application concrète de toutes les notions associées vues en cours. Avec beaucoup de nouveaux concepts à assimiler, nous nous attendions à ce que le projet de LO41 soit d'une difficulté relevée et avons essayé de nous appliquer en conséquence.

Chapitre 2

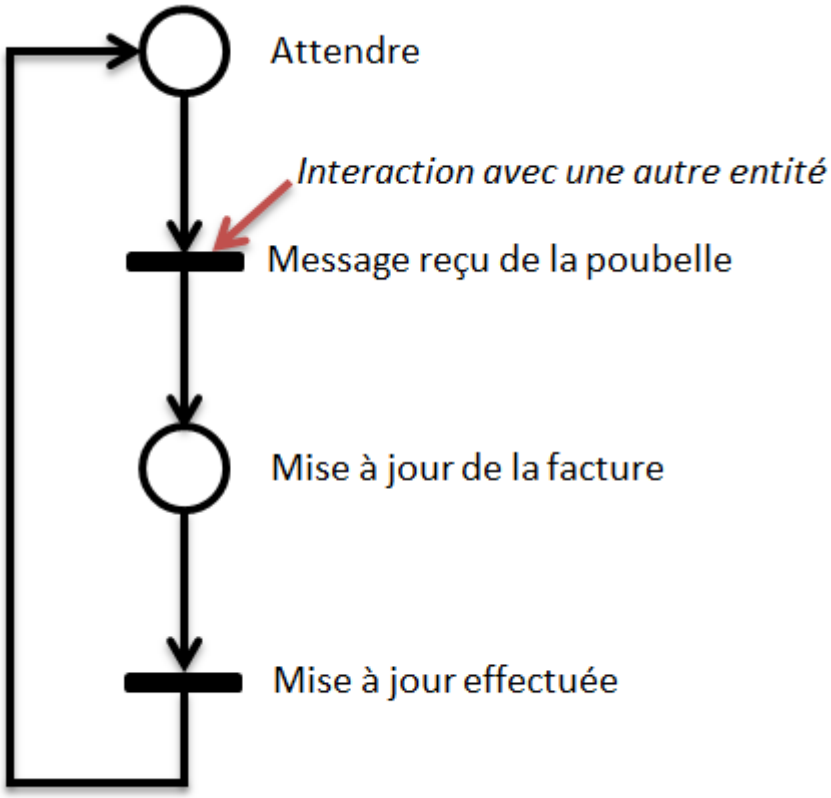
Modélisation du problème

Après avoir relu plusieurs fois l'analyse du projet décrite plus haut, nous avons déduit un certain nombre d'acteurs, dont les tâches sont représentées par les réseaux de Pétri suivants.

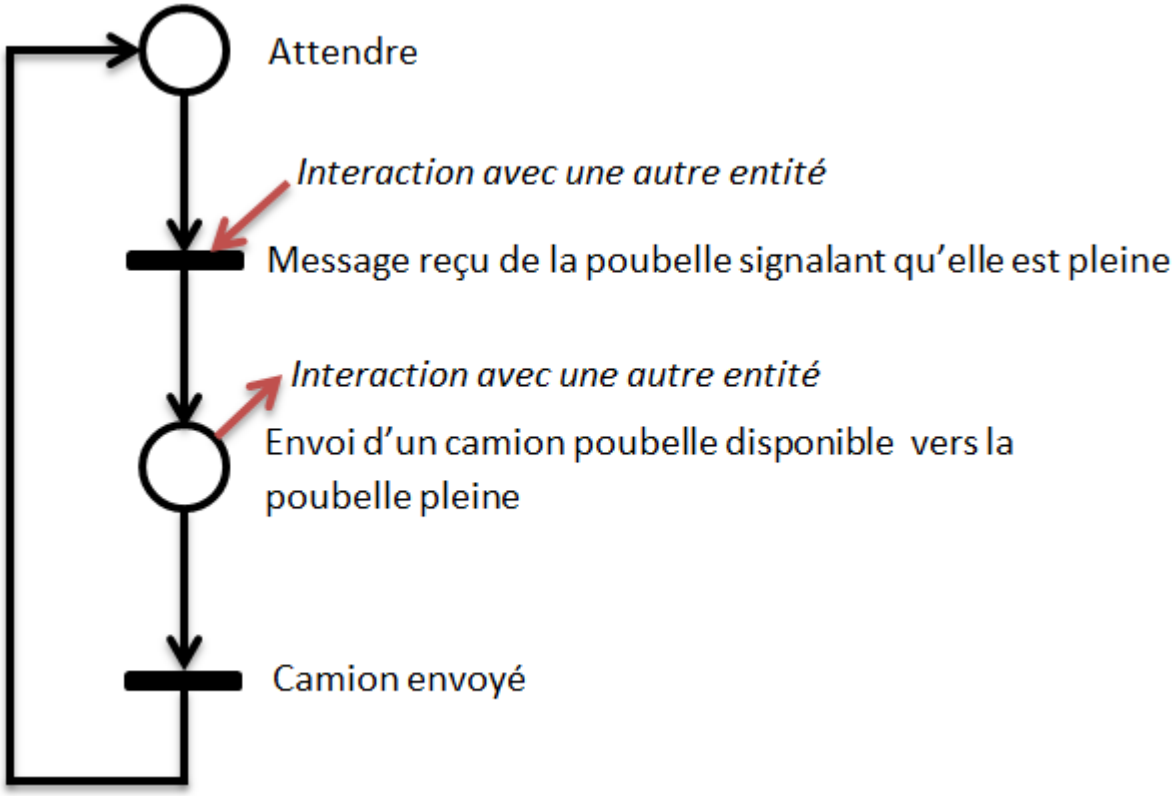
2.1 Réseaux de Pétri : Camion



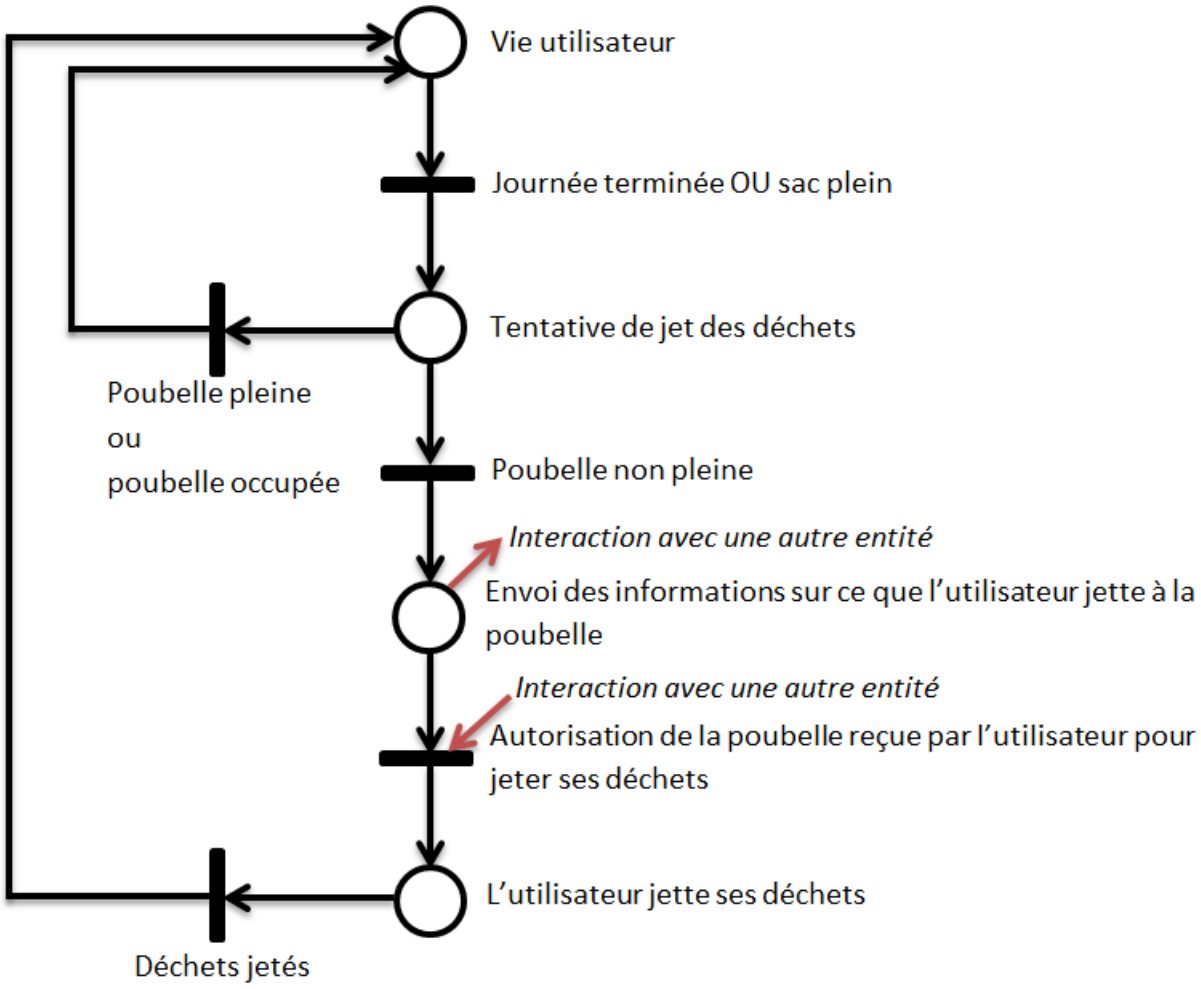
2.2 Réseaux de Pétri : Mairie



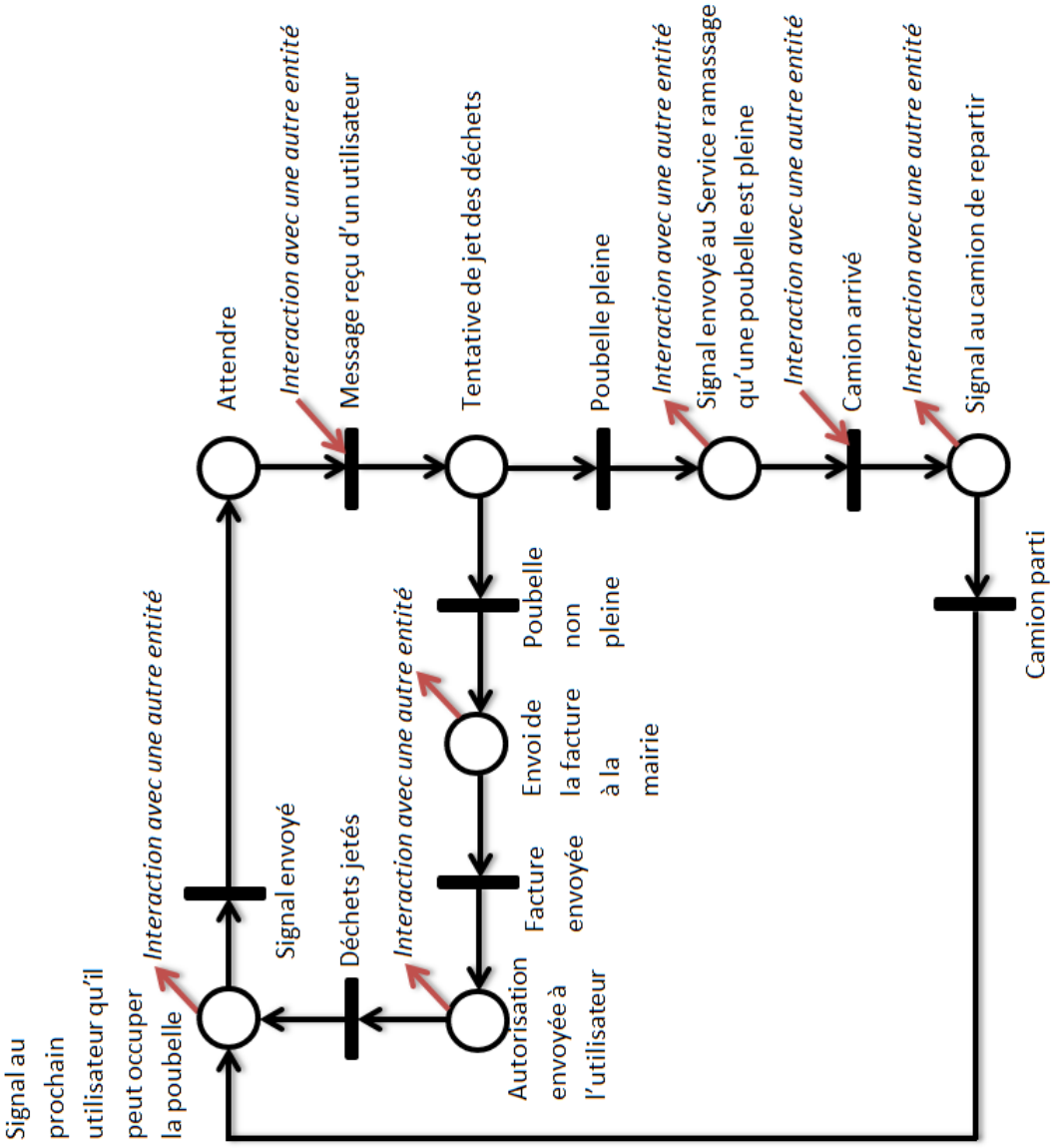
2.3 Réseaux de Pétri : Service de ramassage



2.4 Réseaux de Pétri : Utilisateur



2.5 Réseaux de Pétri : Conteneur/Bac



Chapitre 3

Implementation en C

3.1 Les procédés de synchronisations utilisés

Dans ce projet, nous avons utilisé 4 éléments importants de **synchronisation**.

→ **Les threads.**
Nous en avons eu besoin pour permettre à chacune des entités de vivre leur vie. Ainsi nous pouvons compter :

- 1 thread par utilisateur;
- 1 thread par bac individuel;
- 1 thread par conteneurs;
- 1 thread par camions poubelles;
- 1 thread pour la mairie;
- 1 thread pour le service de ramassage.

Chacun de ces threads accèdent à des ressources qui sont partagées entre eux, c'est pour cela que nous avons eu besoin d'un **mutex** pour les exclusions mutuelles et un bon partage des données.

→ **Les mutex.**
Dans notre projet, nous avons plusieurs **variables partagées** entre les différents threads. Entre autre :

- une variable globale indiquant s'il faut **continuer** le programme ou pas,;
- un booléen permettant de vérifier si une poubelle est **occupée** ou non;
- un booléen permettant de vérifier si une poubelle est **pleine** ou non;
- la **capacité** courante d'une poubelle, **ID** et autres informations potentielles;
- toutes les utilisations des **moniteurs** avec les primitives pthread_cond_signal et pthread_cond_wait...

→ **Les moniteurs.**
Comme nous fonctionnons avec des **threads**, nous avons utilisé des moniteurs pour faire de l'attente passive. Ces moniteurs gèrent tout ce qui est attente sans passage de données, comme par exemple :

- lorsqu'un utilisateur souhaite **accéder** à une poubelle alors qu'elle est occupée;
- pour qu'une poubelle **autorise** l'utilisateur qui l'occupe à jeter ses déchets;
- pour qu'une poubelle puisse signaler au **prochain utilisateur** de la file d'attente de venir jeter ses déchets;
- lorsqu'une poubelle attend qu'un **camion poubelle** vienne la vider...

→ **les files de messages.**

Cet outil est très utile lorsqu'il s'agit d'envoyer des messages d'un thread à un autre grâce aux primitives **msgsnd** et **msgrcv**. Dans notre projet, nous avons créés 3 IPC, l'une pour les messages qui vont **d'entités à entités** (une entité correspond soit à une poubelle, soit à un camion) via leur ID; et l'autre pour les messages que **plusieurs** peuvent envoyer à un **seul et même receveur**.

Ainsi, nous les utilisons de la façon suivante :

- quand un utilisateur se sert d'une poubelle, il envoie son **ID** et le **volume** de déchets à cette dernière pour aider au processus de facturation;
- quand la poubelle est pleine et qu'elle **demande** un camion poubelle au service de ramassage, avec comme message son **adresse**...
- quand une poubelle est pleine et qu'elle communique avec le camion poubelle.
- quand un utilisateur jette des déchets dans une poubelle, cette dernière envoie le litres de déchets à la mairie.

3.2 Structures et fonctions importantes

3.2.1 Utilisateur

Structure Utilisateur

Cette structure correspond à une famille d'individus. Elle est définie par plusieurs attributs :

- **int id**
  ID de l'utilisateur qui permet de l'identifier parmi les autres utilisateurs. La mairie utilisera cet ID pour la facturation;
- **int nbFamille**
  Nombre de personnes composant dans la famille;
- **bool clé**
  Booléen vrai si et seulement si l'utilisateur possède une clé pour ouvrir un conteneur public;
- **bool bac**
  Booléen vrai si et seulement si l'utilisateur possède un bac individuel;
- **Poubelle* pbac**
  Pointeur sur Poubelle. Correspond au bac individuel de l'utilisateur s'il en possède un.
- **Poubelle* psac**
  Pointeur sur Poubelle. Correspond au sac poubelle dont l'utilisateur se sert lorsqu'il possède une clé.
- **Poubelle* conteneur_normal**
  Pointeur sur Poubelle. Correspond au conteneur public de déchets normaux où l'utilisateur jettera son sac poubelle lorsqu'il sera plein. Existe uniquement s'il possède une clé.
- **Poubelle* conteneur_carton**
  Pointeur sur Poubelle. Correspond au conteneur recyclable de déchets papier/cartion fréquenté par l'utilisateur.
- **Poubelle* conteneur_plastique**
  Pointeur sur Poubelle. Correspond au conteneur recyclable de déchets plastique fréquenté par l'utilisateur.
- **Poubelle* conteneur_verre**
  Pointeur sur Poubelle. Correspond au conteneur recyclable de déchets en verre fréquenté par l'utilisateur.
- **MsgBuffer* msgPoubelle**
  Structure de message utilisée par les ipcs. Ici, le message contiendra l'id de l'utilisateur et le volume qu'il souhaite jeter, et sera envoyer à la poubelle en question.

3.2.2 Poubelle

Énumération TypePoubelle

Cette énumération nous permet de différencier les différents types de conteneur présent dans notre ville. On retrouvera donc les types :

- **CARTON**
  Correspondant au conteneurs recyclables accueillant du papier/carton uniquement;
- **PLASTIQUE**
  Correspondant au conteneurs recyclables accueillant du plastique uniquement;
- **VERRE**
  Correspondant au conteneurs recyclables accueillant du verre uniquement;
- **NORMAL**
  Correspondant à tous les autres conteneurs, les sacs poubelles et les bacs non recyclable.

Structure Poubelle

Cette structure représente les différents composants d'un conteneur. Elle est définie par plusieurs attributs :

- **TypePoubelle type**
  Représente le type de poubelle, à l'aide de l'énumération définie précédemment;
- **int id**
  ID unique de la poubelle qui permet de l'identifier parmi les autres entités. Cet ID est également utile pour envoyer des messages à d'autres entités;
- **int capaMax**
  Capacité de stockage maximale de la poubelle;
- **bool capaCourante**
  Capacité courante de stockage de la poubelle;
- **bool occupee**
  Booléen vrai si et seulement le poubelle est actuellement occupée par un utilisateur;
- **bool pleine**
  Booléen vrai si et seulement si la poubelle est pleine;
- **pthread_cond_t viderPoubelle**
  Moniteur utilisé lorsque la poubelle est pleine et attend qu'un camion poubelle arrive;
- **pthread_cond_t prochainUtilisateur**
  Moniteur utilisé pour signaler au prochain utilisateur qu'il peut accéder à la poubelle;
- **pthread_cond_t remplirPoubelle**
  Moniteur utilisé pour autoriser un utilisateur à jeter ses déchets;
- **pthread_mutex_t mutexPoubelle**
  Mutex pour protéger les ressources critiques de cette poubelle uniquement.

3.2.3 Mairie

Structure Mairie

Cette structure représente notre mairie, elle possède donc toutes les informations concernant la ville, bien que son rôle dans notre projet reste la facturation des utilisateurs. Elle est définie par les attributs :

- **ServiceRamassage* serviceRamassage**
  Pointeur sur le service ramassage;
- **Utilisateur utilisateurs**
  Pointeur sur les différents utilisateurs;

– **int* compteurDechets**
Tableau d'entiers de la taille du nombre d'utilisateurs. C'est ici que la mairie stocke en mémoire la consommation de chaque utilisateurs au cours du mois;

– **int* indexPremierUtilisateur**
Cet entier correstion à l'ID à partir duquel les utilisateurs on été créés, chaque entité du projet ayant un ID unique, nous permettant ainsi de parcourir correctement le tableau de facturation défini précédemment.

3.2.4 Service ramassage

Structure ServiceRamassage

Cette structure correspond à l'entité service ramassage. Elle contient donc les attributs :

– **Poubelle* conteneursNormaux**
Pointeur sur les conteneurs normaux de la ville;

– **Poubelle* conteneursCarton**
Pointeur sur les conteneurs recyclables de type papier/carton de la ville;

– **Poubelle* conteneursPlastique**
Pointeur sur les conteneurs recyclables de type plastique de la ville;

– **Poubelle* conteneursVerre**
Pointeur sur les conteneurs recyclables de type verre de la ville;

– **Poubelle* bacs**
Pointeur sur les bacs individuels des différents utilisateurs;

– **Camion* camionsNormaux**
Pointeur sur les camions destinés aux conteneurs non recyclables de la ville;

– **Camion* camionsCarton**
Pointeur sur les camions destinés aux conteneurs recyclables de type papier/carton de la ville;

– **Camion* camionsPlastique**
Pointeur sur les camions destinés aux conteneurs recyclables de type plastique de la ville;

– **Camion* camionsVerre**
Pointeur sur les camions destinés aux conteneurs recyclables de type verre de la ville;

Fonction : Camion chercher_camion_poubelle(ServiceRamassage* sr, TypePoubelle type)

Cette fonction prend en paramètre le service de ramassage et un type pour déterminer dans quelle liste de camions chercher. Elle retourne le premier camion **disponible**, c'est-à-dire le premier camion qui n'est pas déjà en tournée.

Fonction : bool envoyer_camion_poubelle(Camion c, MsgBuffer* m)

Cette fonction est utilisée après avoir trouvé un camion à l'aide de la fonction précédente, et envoie au camion passé en paramètre le message passé en second paramètre. Le camion libre **reçoit ainsi l'adresse** d'une poubelle pleine et part la vider.

3.2.5 Camion

Structure Camion

Cette structure représente un camion poubelle. Ce dernier est composé des différents attributs :

– **int id**
ID unique du camion;

- **Poubelle* poubelle**
  Pointeur sur la poubelle qu'il va vider. NULL si le camion est en attente;
- **bool enDeplacement**
  Booléen vrai si et seulement si le camion est actuellement en tournée. Ainsi, le service ramassage ne pourra plus faire appel à lui tant qu'il ne sera pas de retour au service;
- **pthread_mutex_t mutex_camion**
  Mutex permettant la protection des re ssources critiques liées au camion;

3.2.6 IPC

Structure Message

Comme nous l'avons déjà abordé, différents messages sont envoyés entre les différentes entités lors de la vie du programme. Nous avons décidé de définir la structure de message de la façon suivante :

- **long type**
  Imposé par les ipcs, cet attribut nous permet également un certain filtrage des messages;
- **void** arg**
  Nous avons décidé d'utiliser un attribut original : un void**. En effet, ce dernier nous laisse une liberté presque totale puisque c'est à nous de définir le type de données à traiter à chaque utilisation d'un message. Ainsi, nous avons une liberté à la fois sur le type de données, mais également sur sa taille ou sur le nombre de paramètres si nous décidons de passer sous forme de tableau.

Conclusion

Nous pouvons donc conclure que les problèmes de synchronisation sont extrêmement **complexes**. En effet, dès nos premières réflexions, et tout au long de l'élaboration de notre projet, nous avons pu observer qu'il fallait correctement réfléchir aux différents processus de **synchronisation** pour ne pas se retrouver dans des situations d'**interblocage** ou accéder simultanément à des **données partagées** entre plusieurs threads.
De plus, l'élaboration des différents **réseaux de Pétri** nous a aidé à mieux anticiper les différents scénarios qui pourraient se produire et donc réfléchir à des **solutions** lorsque nous faisions face à des problèmes, avant même de commencer notre code.
Les **structures** de données ont elles aussi jouées un rôle clé puisqu'elles nous ont permises de mettre en place de façon ciblée nos moyens synchronisations, tout en nous laissant suffisamment de liberté quand aux informations partagées.

Les optimisations que nous pouvons apporter à notre projet pourraient être par exemple de permettre aux camions poubelles de récupérer **plusieurs** conteneurs ou bacs individuels en un même déplacement, tant que sa capacité de stockage le permet. Une file de message entre le service de ramassage et le camion poubelle permettrait d'appliquer ce concept. Ensuite, ce à quoi nous avions pensé était le fait de rendre les utilisateurs plus **intelligents**. Par exemple, il pourra sortir de chez lui pour voir si son conteneur assigné est bientôt plein ou non. S'il l'est il va soit commencer à réduire sa consommation, soit aller jeter les poubelles tant que c'est possible. Ceci lui éviterait des temps d'attentes inutiles.

Enfin, en plus de consolider les connaissances obtenues lors du suivi de l'UV, ce projet aura été une expérience supplémentaire de travail en binôme et de gestion des projet.

Sources

→ Compilateur : gcc
http://gcc.gnu.org/
→ Éditeur de texte : vim et gedit
http://www.vim.org/
http://doc.ubuntu-fr.org/gedit
→ Logiciels utilisés pour coder en LaTeX : Texmaker et ShareLatex
http://www.xm1math.net/texmaker/index_fr.html
https://www.sharelatex.com/
→ Logiciel de versioning offline : Meld
http://meldmerge.org/
→ Apprentissage du LateX :
http://fr.openclassrooms.com/informatique/cours/redigez-des-documents-de-qualite-avec-latex