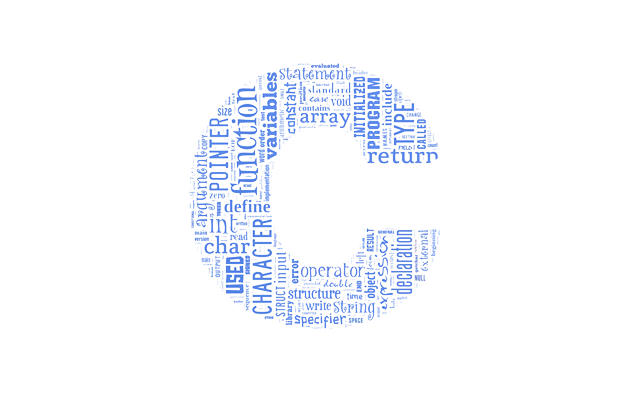
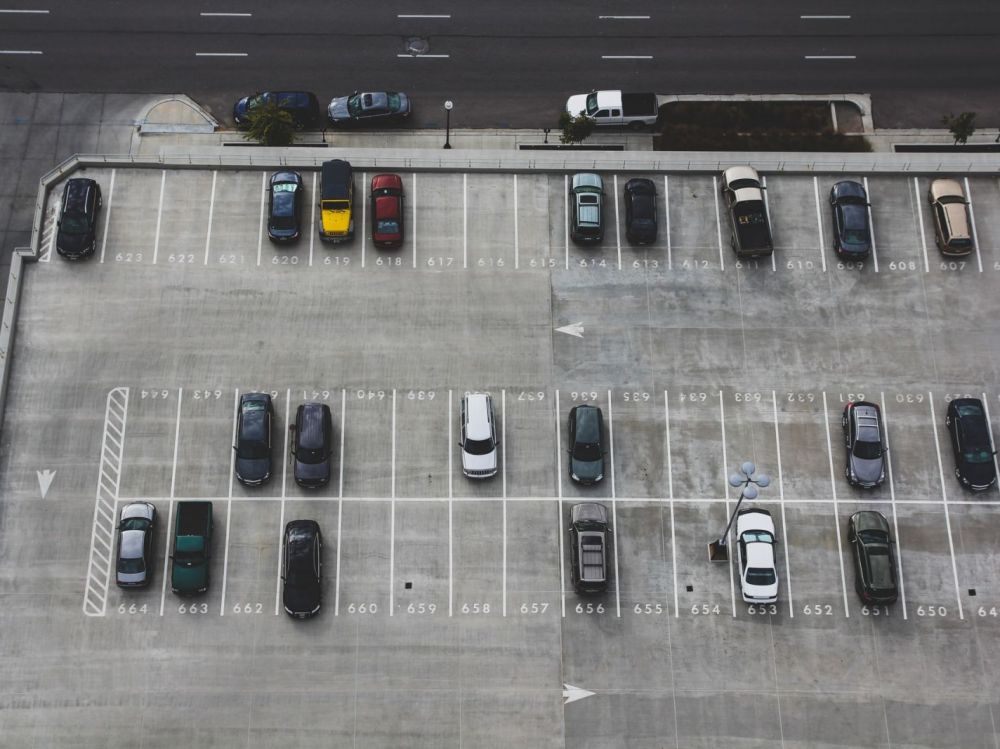
**Semestre printemps 2022**

Rapport de projet - SY40



PROJET LANGAGE C

SIMULATION D’UN PARKING

Projet réalisé par

**Jérémy KIEFFER**

**Titouan BURGY**

Projet encadré par

**Philippe DESCAMPS**

Sommaire

[I. Présentation du projet 3](#_Toc106179879)

[1. Description du projet 3](#_Toc106179880)

[2. Analyse des demandes 3](#_Toc106179881)

[II. Les méthodes utilisées 4](#_Toc106179882)

[1. Choix de conception – Les Threads 4](#_Toc106179883)

[2. Mise en place des structures 5](#_Toc106179884)

[a. Abonné et non-abonné 5](#_Toc106179885)

[b. Parking 6](#_Toc106179886)

[c. Chronomètre 6](#_Toc106179887)

[III. Implémentation des méthodes 7](#_Toc106179888)

[1. Génération des threads 7](#_Toc106179889)

[IV. Affichage sur le terminal 8](#_Toc106179890)

[1. Partie gauche : Heure + places de parking 8](#_Toc106179891)

[2. Partie droite : Actions des usagers 8](#_Toc106179892)

[V. Conclusion 8](#_Toc106179893)

[1. Partie gauche : Heure + places de parking 8](#_Toc106179894)

# ****Présentation du projet****

## Description du projet

Dans le cadre de l’UV SY40 (Architecture des systèmes d’exploitation) en première année d’Informatique à l’UTBM, il nous a été demandé de réaliser un projet pour mettre en pratique les différentes connaissances acquis durant le semestre avec comme finalité la mise en place d’une simulation d’un parking public à l’aide du langage de programmation C. Les différentes consignes pour ce projet peuvent être téléchargé ici :



## Analyse des demandes

Après une première analyse des différentes demandes du projet, nous avons pu mettre en place une liste de problèmes à résoudre pour mener à bien ce projet :

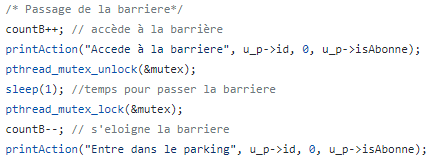
* Quelle(s) méthode(s) utiliser pour la simulation des usagers, du parking etc … ?
* Quelles sont les différentes structures à mettre en place ?
* Comment gérer la création de processus non-stop ? Comment gérer le temps (primordial pour la mise en place de la zone de débordement) ?
* Quel affichage ? Intérieur du parking ou activités à la barrière ?

# Les méthodes utilisées

## Choix de conception – Les Threads

Notre choix pour la simulation des processus s’est assez vite tourné vers les threads. En effet, nous avons privilégié cette méthode pour plusieurs raisons :

* Les différentes fonctions vues en cours nous paraissaient adaptée aux différents problèmes que présente ce projet. Le thread étant un processus léger, la communication entre threads est beaucoup moins lourde et coûteuse que la communication entre processus système (dit lourds, comme l’utilisation de fork). De plus, étant donné que les threads se lancent en même temps que le processus main, la multiplication des threads et la synchronisation des différentes tâches pour la simulation du parking est largement simplifiée.
* D’un point de vue personnel, nous avons tous les deux bien compris et assimilé la notion de thread. Lors des différents travaux pratiques de l’UV, nous étions à l’aise avec l’utilisation des threads là où certains aspects d’autres méthodes nous paraissaient beaucoup plus compliqués pour effectuer la même action.
* La protection des variables ou des parties de code dont la synchronisation est très importante (comme le nombre d’usager dans le parking ou le nombre d’usager qui attende à la barrière) pouvait très simplement être résolue à l’aide des mutex :



Dans l’exemple ci-dessus, on remarque en effet que le principe de « passer la barrière » est très important. Dans la consigne il est en effet précisé qu’un seul utilisateur peut passer de l’autre côté et le mutex nous permis de respecter ce principe-là en rajoutant simplement deux fonctions (implémenter avec la librairie des threads).

## Mise en place des structures

Pour le bon fonctionnement du projet, nous avons mis en place trois structures particulières dont nous allons vous expliquer ci-dessous les détails.

### Abonné et non-abonné

Structure d'un usager

Description générée automatiquement

La première structure mise en place est celle qui permet de représenter les utilisateurs de notre parking. Comme vous pouvez le voir, nous l’avons nommé « Usager » et cette dernière possède trois champs différents. Le premier (id) est simplement l’identifiant de l’usager. Dans notre cas, ce dernier est en faite l’ordre de création des threads ([expliqué un peu plus bas](#_Génération_des_threads_1)), c’est-à-dire que le premier usager créé aura l’id 1, le deuxième aura l’id 2 et ainsi de suite … Le deuxième champ est un booléen (isAbonne) qui, comme cela est noté en commentaire du code, est à vrai si l’usager est un abonné et à faux s’il ne l’est pas. Le dernier champ est sans doute le plus important : c’est un entier (stationnement) qui contient le numéro de la place à laquelle est garé l’usager (s’il n’est pas garé, cet entier est mis à -1). Grâce à cet entier, nous pouvons mettre à jour l’affichage sur le terminal ([voir Affichage sur le terminal](#_Affichage_sur_le)) et gérer finalement les différentes actions que peuvent réalisées l’usager (thread).

### Parking

Une image contenant texte

Description générée automatiquement

La deuxième structure que nous avons créée est celle permettant de représenter le parking en lui-même. Sur l’image ci-dessus, vous pouvez voir que nous avons appelé la structure « PlaceParking » qui représente comme son nom l’indique une place de parking. Elle possède trois champs : l’entier id (numéro de la place), le booléen isAbonne (même chose que pour la structure « Usager », est à vrai si la place est réservée à un abonné, faux si elle ne l’est pas) et l’entier idUsager qui va contenir l’id de l’usager qui est actuellement garé sur cette place). Enfaite, le parking va être lui représenté par un tableau de « PlaceParking » :



NUM\_P étant une constante définie dans le fichier « CodingFunctions.h » comme NUM\_P\_ABONNE (nombres places pour abonné) + NUM\_P\_NABONNE (nombres places pour non-abonné) dont les valeurs peuvent être modifiées pour tester le programme avec différentes tailles de parking et même différentes proportions de place réservée aux abonnés.

### Chronomètre

Une image contenant texte

Description générée automatiquement

Pour pouvoir mettre en place le système de débordement à partir de 18h, il nous fallait une structure pour gérer le parking en fonction de l’heure. Nous avons donc ajouté au programme la structure « Heure » qui possède deux champs : un entier (h) pour gérer donc l’heure fictive et un deuxième entier (min) qui représente les minutes. Ce deuxième entier a été implémenter pour plus de compréhension sur la notion du temps dans notre simulation.

# Implémentation des méthodes

## Génération des threads

## 

# Affichage sur le terminal

## Partie gauche : Heure + places de parking

## Partie droite : Actions des usagers

# Conclusion

## Nos impressions

## Les points d’améliorations