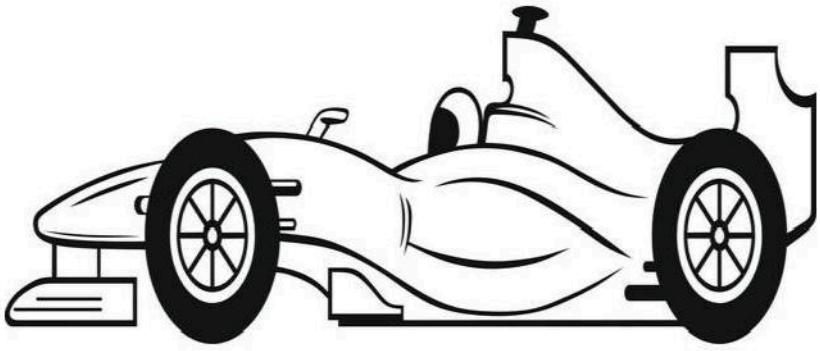
EPHEC

# Rapport de projet “Formule 1”
# Groupe 4

OS travaux pratiques

Daniel O., Martin M., Morgan V., Martin P.

07 dec 2019



# Table des matières

# Liste des tableaux



# Table des figures



# Listings

# 1 Rapport du projet : F1-of-Linux

**Groupe 4**

Notre groupe est constitué de 4 personnes :

- Daniel Olivier
- Martin Michotte
- Morgan Valentin
- Martin Perdaens

## 1.1 Introduction et présentation du projet

Ce projet consiste à présenter un week-end complet d’un grand prix de Formule 1, depuis les séances d’essais du vendredi jusqu’à la course du dimanche, en passant par les essais du samedi et la séance de qualifications.

Notre but consiste à reproduire cela dans un langage de programmation performant à l’exécution des méthodes implémentées, le langage C. Nous devons générer un affichage qui gérera les séances d’essais libres, les qualifications ainsi que la course. De plus, certaines informations doivent être disponible : temps au tour, temps secteur, disqualification, arrêt aux stands, temps depuis le début de la course.

De plus, nous devons appliquer des concepts vus en cours en première année ainsi qu’en deuxième : processus père-fils (dont `fork` est la création d’un nouveau processus utilisateur), sémaphores (pour gérer la synchronisation des processus) et la mémoire partagée (allocation et utilisation par appel des mémoires partagées via leurs identificateurs).

## 1.2 Cahier des charges du projet

### 1.2.1 Projet OS Octobre 2019

Le but du projet est de gérer un week-end complet d’un grand prix de Formule 1, depuis les séances d’essais du vendredi jusqu’à la course du dimanche, en passant par les essais du samedi et la séance de qualifications.

Il y a 20 voitures engagées dans un grand prix. Leurs numéros sont : 44, 77, 5, 7, 3, 33, 11, 31, 18, 35, 27, 55, 10, 28, 8, 20, 2, 14, 9, 16.

Un circuit de F1 est divisé en 3 secteurs (S1, S2, S3).

Le calendrier d’un week-end de F1 est établi comme suit :

- Vendredi matin, une séance d’essais libres d’1h30 (P1)
- Vendredi après-midi, une séance d’essais libres d’1h30 (P2)
- Samedi matin, une séance d’essais libres d’1h (P3)
- Samedi après-midi, la séance de qualifications, divisée en 3 parties :
  - Q1, durée 18 minutes, qui élimine les 5 dernières voitures (qui occuperont les places 16 à 20 sur la grille de départ de la course)
  - Q2, durée 15 minutes, qui élimine les 6 voitures suivantes (qui occuperont les places 11 à 16 sur la grille de départ de la course)
  - Q3, durée 12 minutes, qui permet de classer les 10 voitures restantes pour établir les 10 premières places sur la grille de départ de la course
- Dimanche après-midi, la course en elle-même.

Votre projet devra prendre en charge les choses suivantes.

### 1.2.2 Première partie : gestion des séances d’essai, des qualifications et de la course

**Lors des séances d’essais (P1, P2, P3) :**

- ☒ Relever les temps dans les 3 secteurs à chaque passage pour chaque voiture
- ☒ Toujours savoir qui a le meilleur temps dans chacun des secteurs
- ☒ Classer les voitures en fonction de leur tour complet le plus rapide
- ☒ Savoir si une voiture est aux stands (P)
- ☒ Savoir si une voiture est out (abandon de la séance)
- ☒ Dans ces 2 derniers cas, on conserve toujours le meilleur temps de la voiture et celle-ci reste dans le classement
- ☒ Conserver le classement final à la fin de la séance

**Lors des qualifications (Q1, Q2, Q3) :**

- ☒ Relever les temps dans les 3 secteurs à chaque passage pour chaque voiture
- ☒ Toujours savoir qui a le meilleur temps dans chacun des secteurs
- ☒ Classer les voitures en fonction de leur tour complet le plus rapide
- ☒ Savoir si une voiture est aux stands (P)
- ☒ Savoir si une voitures est out (abandon de la séance)
- ☐ Dans ces 2 derniers cas, on conserve toujours le meilleur temps de la voiture et celle-ci reste dans le classement
- ☒ A la fin de Q1, il reste 15 voitures qualifiées pour Q2 et les 5 dernières sont placées à la fin de la grille de départ (places 16 à 20)
- ☒ A la fin de Q2, il reste 10 voitures qualifiées pour Q3 et les 5 dernières sont placées dans les places 11 à 15 de la grille de départ
- ☒ Le classement de Q3 attribue les places 1 à 10 de la grille de départ
- ☒ Conserver le classement final à la fin des 3 séances (ce sera l’ordre de départ pour la course).

**Lors de la course :**

- ☒ Le premier classement est l’ordre sur la grille de départ
- ☒ Le classement doit toujours être maintenu tout au long de la course (gérer les dépassements)
- ☒ Relever les temps dans les 3 secteurs à chaque passage pour chaque voiture
- ☒ Toujours savoir qui a le meilleur temps dans chacun des secteurs
- ☒ Toujours savoir qui a le tour le plus rapide
- ☐ Savoir si la voiture est out (abandon) ; dans ce cas, elle sera classée en fin de classement
- ☒ Savoir si la voiture est aux stands (PIT), gérer le temps aux stands et faire ressortir la voiture à sa place dans la course (généralement 2 ou 3 PIT par voitures)
- ☒ Conserver le classement final et le tour le plus rapide

***Remarque :* les stands se trouvent toujours dans le secteur 3.**

De plus, il vous est demandé de paramétrer votre programme.

En effet, les circuits peuvent être de longueur très variable et, dès lors le nombre de tours pour la course varie également (on essaie que le nombre total de kilomètres soit toujours plus ou moins le même pour chacune des courses du calendrier).

**On vous demande de :**

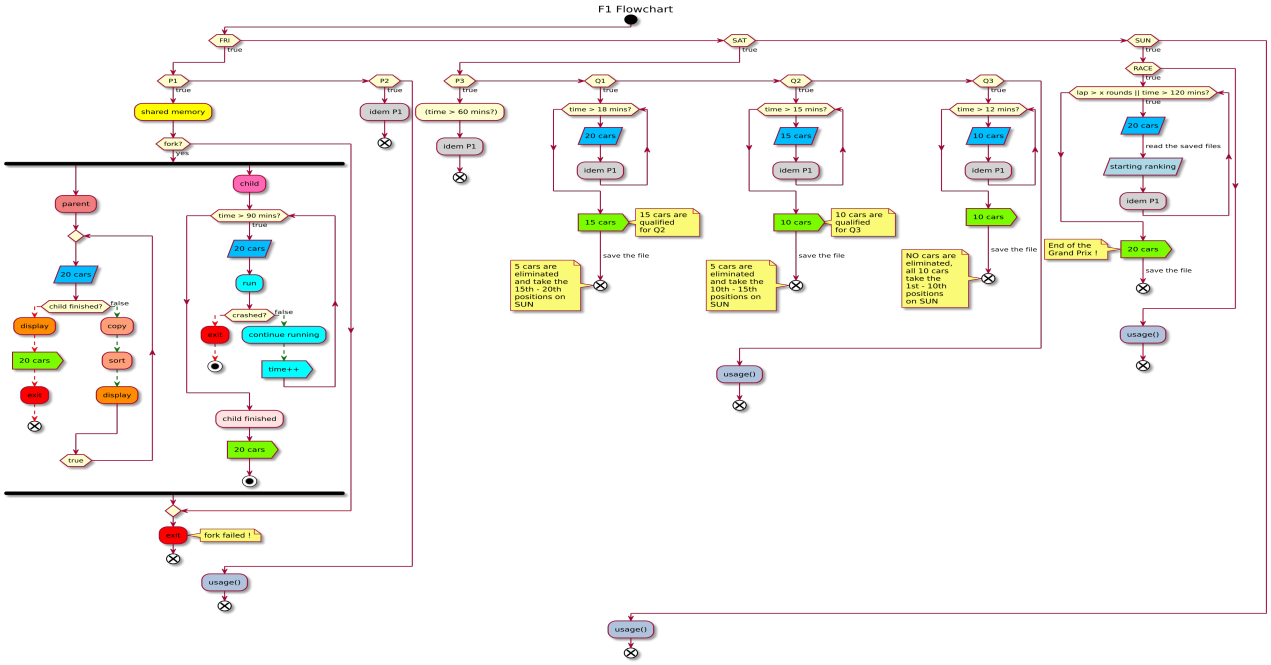
- ☒ Réaliser le programme en C sous Linux
- ☒ Utiliser la mémoire partagée comme moyen de communication inter-processus
- ☒ Utiliser les sémaphores pour synchroniser l’accès à la mémoire partagée

## 1.3 Analyse du travail

### 1.3.1 Plan du programme

Afin d’améliorer notre rapport, nous avons retenu les remarques faites par la professeur lors d’une séance de TP. Suite à ce dernier, nous avons décidé de commencer par décortiquer les demandes et en faire un flowchart afin de mieux visualiser le projet :

**Fig. 1 :** Flowchart

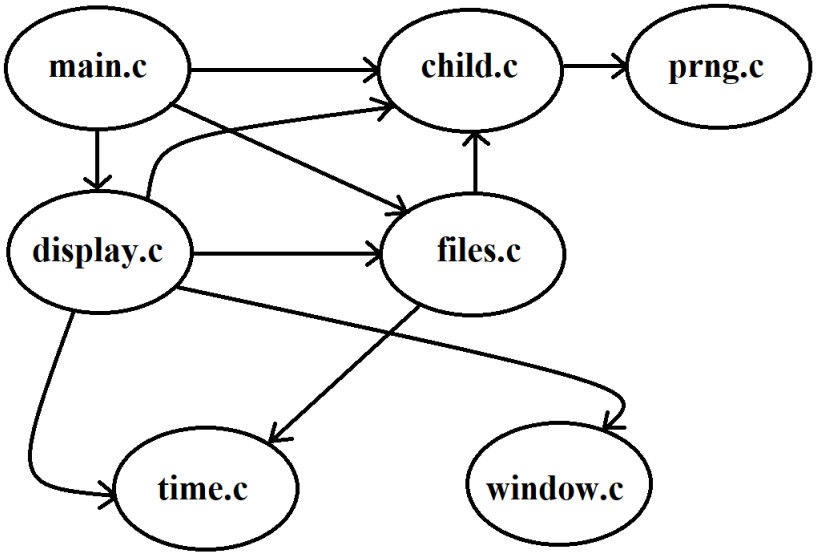



Malheureusement vu la taille du l’image, on était obligé de l’imprimer à part sur une feuille A3 qui se trouve en annexe du rapport. Comme vous pouvez le voir sur l’image le projet est divisé en 3 parties. Vendredi, samedi et dimanche. P1 et P2 sont exécutés le vendredi, samedi on a les étapes suivantes : P3, Q1, Q2, Q3 et afin dimanche on a la course finale.

Pour une question de sécurité si le nom jour passé comme argument du programme est ni vendredi ni samedi ni dimanche, le programme affiche un manuel en console et s’arrêter.

### 1.3.2 Découpage en plusieurs fichiers

**Fig. 2 :** Structure des fichiers




Pour simplifier le projet, nous avons décidé de découper ce dernier en plusieurs fichiers au lieu d’avoir tout le code dans un seul fichier. Nous avons **7 fichiers** C qui communiquent entre eux pour produire un exécutable. Le fait d’avoir coupé le code en plusieurs fichiers nous a beaucoup aidé lors du débogage des problèmes rencontrés au fur à mesure qu’on avançait.

### 1.3.3 Fichiers principaux

Sans surprise, le fichier le plus important c’est le fichier **main.c**. C’est dans ce dernier qu’on trouve la création de la mémoire partagée, des sémaphores, paramétrage du programme et également la création des fils/voitures qui vont participer au Grand Prix. Le fichier **display.c**. sert principalement à afficher les données triées en console, le fichier **child.c** comme son nom l’indique c’est dans ce fichier que les voitures créé par la fonction `fork` dans **main.c** vont écrire dans la mémoire partagée. et finalement parmis les fichiers principaux, on a le fichier **files.c** qui va se charger de tout ce qui a avoir avec un fichier. La création et la lecture des fichiers est géré par ce fichier **files.c**.

### 1.3.4 Description de la méthode de travail

Tout premièrement, nous avons décidé de travailler avec un logiciel de gestion de version notamment connu sous le nom de ***git***. Ce type de logiciel est devenu indispensable lorsqu’on travaille à plusieurs sur un même projet et donc sur le même code source. cela nous a permit deux choses :

- **suivre l’évolution du code source**, pour retenir les modifications effectuées sur chaque fichier et être ainsi capable de revenir en arrière en cas de problème.
- **travailler à plusieurs**, sans risquer de se marcher sur les pieds. Si deux personnes modifient un même fichier en même temps, leurs modifications doivent pouvoir être fusionnées sans perte d’information.

On a remarqué toute au début du projet qu’il était primordiale d’avoir une voiture qui tourne et affiche bien les données en console avant d’en avoir 20 qui tournent en même temps. Du coup on a commencé par créer le fichier **child.c** pour générer aléatoirement le temps secteurs, le temps passé au stand etc…

Une fois qu’on avait une voiture qui tournait correctement, on est passé à l’étape suivante l’affichage. On a décidé de représenter les données en console sous forme d’une table. Il existe plusieurs librairies qui permettent d’avoir une table en console mais la plupart supporte pas le rafraichissement. L’un de plus gros challenge qu’on a rencontré, c’était le rafraichissement des données dans une table en console. On a fini par utiliser une librairie disponible sur github sous le nom de **libfort** (lien) et quelques commandes bash qu’on a dû convertir en langage C pour obtenir le rafraichissement des données affichés en console.

Après avoir réussi à obtenir une voiture qui tournait correctement et un affichage qui nous convenait bien, on est passé à la création de la mémoire partagée, évidemment créer ce dernier sans avoir au moins 2 processus qui tournent n’a pas de de sens. On est resté bloquer sur la mémoire partagée pendant plusieurs semaines car on a connu plusieurs problèmes notamment les fils qui terminaient jamais, des processus zombie, les voitures qui tournaient plus alors qu’une tournait bien avant la mémoire partagée…Mais tous ces problèmes ont été résolu.

À ce stade, on avait un affichage correcte et la mémoire partagée qui fonctionnait correctement, on est passé l’étape suivant, le trie des données et surtout trouver un moyen d’avoir le bon classement pour la course de dimanche. Cette étape englobe plusieurs choses, pour pouvoir arriver à un bon classement des voitures pour la course de dimanche, il a fallu introduire le paramétrage du programme qu’on n’a pas encore traité càd passer les étapes P1, P2, Q1 jusqu’à RACE comme arguments de notre programme et la gestion des dépassements qui va engendrer la création des fichiers.

## 1.4 Explication des particularités du code

### 1.4.1 Fonctionnalités du code

Pour le paramétrage du programme, nous avons décider d'utiliser un parser la ligne de commande via la fonction `getopt_long(...)` disponible avec GNU C. Elle permet d'avoir des noms longs d'option, commençant par deux tirets. Pour l'implémentation voir le code en annexe dans le fichier **main.c**.

Notre programme prend en tant qu'arguments 4 options :

1. L'option ***–day*** qui prend comme paramètre le nom du jour
2. L'option ***–step*** qui prend comme paramètre le nom de l'étape
3. L'option ***–length*** qui prend comme paramètre la longueur du circuit en kilomètre
4. L'option ***–help*** qui prend aucun paramètre, sert juste à afficher le manuel du programme.

Si l'option ***–length*** n'est pas fourni comme argument du programme, une valeur par défaut est attribuée.

> Lors des séances d'essais (P1, P2, P3) :
>
> ```
> ./prog --day fri --step P2
> ```

> Lors des qualifications (Q1, Q2, Q3) :
>
> ```
> ./prog --day sat --step Q3
> ```

> Lors de la course (RACE) :
>
> ```
> ./prog --day sun --step RACE --length 10
> ./prog -d sun -s RACE -l 10
> ```

> Pour avoir le manuel (help) :
>
> `./prog --help` ou `./prog -h`

On lance la phase sélectionnée pour chacune des voitures participantes qui vont courir pendant un temps définit. Lors de la simulation, les voitures participantes vont générer des temps aléatoires à chaque secteur et ces données sont écrites dans la mémoire partagée.

Un tableau de valeurs reprenant des informations diverses est ensuite affiché afin de pouvoir suivre l'évolution de l'étape choisie. Les informations représentée dans ce dernier dépendent de l'étape

concernée. Ce tableau est également trié en fonction du meilleur temps de tour par pilote ou, dans le cadre de la course, trié en fonction de leur position.

Au départ de la course, chaque participant démarre dans l’ordre précédemment déterminé par les séances de qualifications et avec une pénalité relative à leur position de départ.

Lorsque la simulation d’une étape est terminée, les positions des pilotes est sauvegardée dans un fichier. Ce fichier sera chargé lors de l’étape suivante afin de déterminer les participants ainsi que leurs positions.

### 1.4.2 Mémoire partagée et communication entre processus

La mémoire partagée est un moyen efficace de transférer des données entre processus indépendants. On crée ce dernier via les appels systèmes `shmget(...)`, `shmat(...)` et `shmdt(...)`. L’appel système `shmget(...)` permet de créer un segment de mémoire partagée.

Le premier argument de `shmget(...)` est une clé qui identifie le segment de mémoire partagée, on fait ensuite appel à la fonction `shmat(...)` qui permet d’attacher un segment de mémoire partagée à un processus. Elle prend comme premier argument l’identifiant du segment de mémoire retourné par `shmget(...)`. Pour l’implémentation de ces appels systèmes voir le code en annexe dans le fichier **main.c**.

La mémoire partagée contient un `struct` comportant les informations de secteurs entre autres choses.

**Listing 1 :** shared struct

```
1  typedef struct F1_Car {
2      int id;
3      double lap_time;
4      double s1;
5      double s2;
6      double s3;
7      int best_s1;
8      int best_s2;
9      int best_s3;
10     int stand;
11     int out;
12     int lap;
13     int best_lap_time;
14     int done;
15 } F1_Car;
```

### 1.4.3 Sémaphores

Comme la mémoire partagée ne dispose d'aucun dispositif de synchronisation. Rien ne permet de veiller automatiquement à ce qu'un processus ne puisse commencer à lire la mémoire alors qu'un autre processus n'y a pas terminé son écriture ; c'est à nous de régler l'accès à cette ressource commune aux processus ayant accès à cette mémoire partagée.

Dans notre cas, la mémoire partagée n'est accédée ou modifiée qu'avec un seul « **écrivain** » et un seul « **lecteur** » à la fois ; il n'y aura jamais plus d'une écriture et lecture en même temps. Ici, chaque processus fils est un écrivain alors que le lecteur est le processus père.

Il y a plusieurs variété de sémaphores, les sémaphores du System V et également les plus récents les sémaphores POSIX. La librairie POSIX comprend deux types ; les **unnamed semaphores** & **named semaphores**. Dans notre projet on utilise les sémaphores POSIX de type **unnamed semaphores** qui sont disponibles dans la librairie standard C (GNU).

La glibc (GNU C Library) offre donc une implémentation des sémaphores. Ces derniers nous permettent de garantir l'accès exclusif à la mémoire partagée. Pour pouvoir utiliser un sémaphore, il faut d'abord l'initialiser. Cela se fait en utilisant la fonction `sem_init(...)` qui prend comme premier argument un pointeur vers le sémaphore à initialiser, deuxième argument pshared indique si ce sémaphore sera partagé entre les threads d'un processus ou entre processus.

Si pshared vaut 0, le sémaphore est partagé entre les threads d'un processus si non c'est entre les processus. Enfin, le troisième argument value spécifie la valeur initiale du sémaphore. Les opérations `sem_wait(sem_t *sem)` et `sem_post(sem_t *sem)` permettent respectivement de verrouiller et déverrouiller une sémaphore. Pour l'implémentation de ces fonctions voir le code en annexe dans le fichier **main.c**.

### 1.4.4 Libération des ressources de l'ordinateur

Afin de libérer les ressources de l'ordinateur, plusieurs étapes sont réalisées une fois que les processus enfants ont terminé leur fonction et que le programme est prêt à quitter.

Premièrement, il y a « détachement » de la mémoire partagée et ensuite ce dernier est supprimée.

L'appel système `shmdt(...)` permet de détacher un segment de mémoire qui avait été attaché en utilisant `shmat(...)`. L'argument passé à `shmdt(...)` est l'adresse d'un segment de mémoire attaché préalablement par `shmat(...)`. Lorsqu'un processus se termine, tous les segments auxquels il était attaché sont détachés lors de l'appel à `exit(...)`.

Détacher la mémoire partagée ne la supprime pas. Détacher la mémoire partagée permet juste de casser la correspondance entre les pages de l'espace virtuel dédiées au segment de mémoire et les pages frames de la mémoire physique dédiées au segment de mémoire partagée. Pour réellement supprimer la mémoire partagée on fait appel à la fonction `shmctl(...)`.

L’appel système `shmctl(...)` prend trois arguments. Le premier est un identifiant de segment de mémoire partagée retourné par `shmget(...)`. Le deuxième est une constante qui spécifie une commande. On utilise uniquement la commande *IPC_RMID* qui permet de retirer le segment de mémoire partagée dont l’identifiant est passé comme premier argument. Si il n’y a plus de processus attaché au segment de mémoire partagée, celui-ci est directement supprimé. Sinon, il est marqué de façon à ce que le noyau retire le segment dès que le dernier processus s’en détache. `shmctl(...)` retourne 0 en cas de succès et -1 en cas d’échec.

Deuxièmement, on fait presque la même chose avec les sémaphores mais avec des fonctions différentes. La fonction `sem_destroy(...)` permet de libérer un sémaphore qui a été initialisé avec `sem_init(...)`. Les sémaphores consomment des ressources qui peuvent être limitées dans certains environnements. Il est important de détruire proprement les sémaphores dès qu’ils ne sont plus nécessaires. L’implémentation de ces fonctions qui nous permettent de libérer des ressources se trouve en annexe dans le fichier **main.c**.

### 1.4.5 Création et gestion des processus

Chaque voiture correspond à un processus fils, tandis que le père s’occupe de la gestion des étapes passés comme arguments du programme et de l’affichage.

La création des processus se fait par la fonction `fork()`, faisant partie des appels système POSIX. Elle permet de donner naissance à un nouveau processus qui est sa copie.

La création des processus fils est présent dans le fichier de code source **main.c**.

#### Rôle du processus père

Dans notre cas, nous avons un processus père donnant naissance au nombre de processus fils nécessaire à l’étape choisie. Chaque processus fils représente une voiture.

Le processus père, quant à lui, va lire des informations provenant de la mémoire partagée. Il s’occupe également de l’affichage ainsi que du tri tout comme sauvegarde des informations sur fichier des étapes de qualifications et de la course de dimanche.

#### Rôle des processus fils

Dans le cadre de ce projet, les fils sont seulement chargés à courir. Càd exécuter les étapes à faire pour un week-end complet d’un grand prix de Formule 1. Pour y arriver on utilise une boucle `while(...)` qui a comme condition si *le temps de l’étape choisie n’a pas écoulé, alors les fils continuent à courir* en écrivant leur temps des secteurs dans le struct présent dans la mémoire partagée. Pour la course de dimanche les fils courent tant qu’ils n’ont pas fini les tours à faire.

Le nombre de tours à faire est déterminé par la longueur du circuit qui varie en fonction l’option ***–length*** passé comme argument du programme, si ce dernier n’est pas fourni une valeur par défaut est attribuée qui vaut 7km. Le code du fils est présent dans le fichier **child.c**.

### Affichage

Pour pouvoir les données dans table, on utilise une librairie public **libfort** disponible sur github : https://github.com/seleznevae/libfort. Voilà un exemple du résultat lors de l’étape Q2.

**TAB. 1 :** Table des résultats.

| Position | NAME | S1 | S2 | S3 | OUT | PIT | LAP | LAP TIME | BEST LAP TIME |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 7 | 32” :03 | 39” :03 | 44” :77 | 0 | 0 | 13 | 1’ :02” :19 | 1’ :19” :42 |
| 2 | 35 | 42” :53 | 41” :27 | 39” :23 | 0 | 1 | 9 | 1’ :11” :71 | 1’ :22” :31 |
| 3 | 40 | 36” :13 | 30” :03 | 44” :12 | 0 | 0 | 16 | 1’ :03” :36 | 1’ :44” :28 |
| 4 | 40 | 40” :04 | 43” :03 | 43” :03 | 0 | 0 | 10 | 1’ :40” :11 | 1’ :51” :47 |
| 5 | 77 | 33” :11 | 34” :43 | 42” :59 | 0 | 1 | 11 | 1’ :17” :23 | 2’ :12” :73 |

Pour les colonnes de secteurs, lap time et best lap time, les données qui sont initialement des entiers sont convertis en temps réel pour une question de lisibilité.

Il y a également une deuxième table pour savoir qui a le meilleur temps dans chacun des secteurs.

**TAB. 2 :** Table de meilleur temps dans chacun des secteurs.

| SECTOR | NAME | TIME |
|---|---|---|
| S1 | 3 | 31” :03 |
| S2 | 42 | 33” :27 |
| S3 | 36 | 38” :44 |

L’implémentation de ces tables voir le code en annexe dans le fichier **display.c**.

### Le trie

Avant de trier on fait une copie des données du struct partagée entre les processus via la fonction `memcpy(...)`. Cette fonction permet de copier un bloc de mémoire spécifié par le paramètre source, et dont la taille est spécifiée via le paramètre size, dans un nouvel emplacement désigné par le paramètre destination. Il est bien entendu qu’il est de notre responsabilité d’allouer suffisamment de mémoire pour le bloc de destination afin qu’il puisse contenir toutes les données.

Pour pouvoir classer les voitures en fonction de leur tour complet le plus rapide, ou pour pouvoir gérer les dépassements lors la course, on utilise la fonction de la librairie `qsort(...)`.

**Listing 2 :** man of qsort

```
1 void qsort(void *base, size_t nel, size_t width,
2             int (*compar)(const void *, const void *));
```

Son premier paramètre est un pointeur vers le début de la zone mémoire à trier, Le second est le nombre d’éléments à trier, le troisième contient la taille des éléments stockés dans le tableau et le quatrième argument est un pointeur vers la fonction qui permet de comparer deux éléments du tableau. Cette fonction retourne un entier négatif si son premier argument est inférieur au second et positif ou nul sinon.

Les deux paramètres de type `(const void *)` font appel à l’utilisation de pointeurs `(void *)` qui est nécessaire car la fonction doit être générique et pouvoir traiter n’importe quel type de pointeurs. `(void *)` est un pointeur vers une zone quelconque de mémoire qui peut être casté vers n’importe quel type de pointeur par la fonction de comparaison. Le qualificatif `const` indique que la fonction n’a pas le droit de modifier la donnée référencée par ce pointeur, même si elle reçoit un pointeur vers cette donnée. On retrouve régulièrement cette utilisation de `const` dans les signatures des fonctions de la librairie pour spécifier des contraintes sur les arguments passés à une fonction.

Les voitures sont triés sur leur tour complet le plus rapide et sur leur tour lors de la course de dimanche. Cela est géré par la fonction `compare(...)` qui est passé en paramètre à la fonction `qsort(...)`.

L’implémentation de ces fonctions voir le code en annexe dans le fichier **display.c**.

### 1.4.6 Création et gestion des fichiers

Pour avoir l’ordre sur la grille de départ lors de la course de dimanche, on passe par plusieurs étapes :

1. *Enregistrement des fichiers Q1, Q2, Q3*

   Avant la terminaison du programme, on sauvegarde ces 3 fichiers car ils seront chargés lors de l’étape suivante afin de déterminer les participants ainsi que leurs positions. La fonction `save_ranking(...)` permet de sauvegarder les positions des pilotes dans un fichier qui aura comme nom le nom de l’étape en cours d’exécution.

2. *Lecture des fichiers Q1, Q2, Q3*

   L’étape suivante est de lire ces fichiers pour pouvoir classer les qualifiés et non qualifiés. Les non qualifiés au Q1 et Q2 sont d’abord mis dans un array puis finalement dans un fichier car on exit le programme après chaque étape du week-end de Formule 1. La fonction `read_files(...)` permet de lire les 15 premiers lignes du fichier Q1 à l’étape Q2 pour pouvoir déterminer les participants ainsi que leurs positions. On fait la même chose à l’étape Q3 mais ici, c’est les 10 premiers lignes du fichier Q2 qui sont lus afin de déterminer ses participants. Et finalement le fichier Q3 est lu avant que la course de dimanche démarrer. Tout ça est géré par la fonction `read_files(...)`.

3. *Lecture des fichiers lastQ1, lastQ2, Q3*

   Le fichier lastQ1 contient les 5 derniers voitures du Q1, lastQ2 contient les 5 derniers voitures du Q2. Comme au Q3 aucune voiture est éliminé on n’a donc pas besoin de créer un troisième fichier pour les éliminés. On utilise le fichier Q3 généré à la fin de Q3 afin de déterminer et classer les 10 premiers positions sur la grille de départ. Les autres places restant sont remplis grâce aux fichiers lastQ1 et lastQ2. Pour l’implémentation de ces 3 étapes voir le code en annexe dans le fichier **files.c**.

### 1.4.7 Sécurité du programme

Pour éviter de rendre le code lourd on a décidé de rendre les séances d’essais libre. Càd on peut passer au P2 sans avoir exécuter P1. Par contre quand on passe au séances de qualifications surtout au Q2 et Q3, on a ajouté de la sécurité. Q2 sera jamais exécuter si Q1 n’a jamais été lancé car Q2 lit un fichier généré à la fin de Q1 afin de déterminer les participants ainsi que leurs positions. Si ce fichier n’est pas disponible le programme s’arrêter, également pour Q3 qui va jamais se lancer si le fichier Q2 est inexistant car Q3 dépend de Q2. Pour l’étape RACE, ce dernier dépend de plusieurs fichiers notamment Q3, les fichiers contenant les élimninés du Q1 et Q2.

Il existe également un manuel (***–help*** ) qui est affiché lorsque la commande passé comme arguments du programme est erroné afin d’éviter le crash du programme. Cela nous permet aussi de garantir que le programme se lancer uniquement si ce dernier a reçu correctement les arguments attendu. L’implémentation de tout cela voir le code en annexe dans le fichier **main.c**.
Difficultés rencontrées et solutions ——————————————

Concernant les difficultés rencontrées, la mémoire partagée et le classement de départ pour la course étaient les plus gros challenges du projet. Et comme on ne suit pas vraiment la formule 1, on a dû se renseigner un peu sur le sujet.

## 1.5 Évolutions futures

### 1.5.1 Intégration de codes couleurs dans l’affichage : DONE !

Il s’agit certes d’une implémentation de moindre importance, mais on pense que cela pourrait s’avérer pratique pour ressortir de manière plus rapide les informations les plus importantes. On a donc réalisé un code couleur pour :

- Les 3 premières places dans le classement,
- Le temps le plus rapide au tour,
- Les voitures au stand
- La voiture ayant le temps le plus rapide au tour depuis le début de la course,
- La ou les voiture(s) ayant abandonné la course (OUT).

### 1.5.2 Affichage cliquable : TODO !

Comme à la manière de `htop` dans Linux, la possibilité de cliquer sur un des en-têtes de colonne afin de trier automatiquement l’affichage en fonction de cette colonne pourrait s’avérer intéressante. En effet, si l’utilisateur souhaite prêter plus particulièrement son attention sur une catégorie d’information précise, cela pourrait lui être utile.

### 1.5.3 Options lié à la pression d’une touche de clavier : TODO !

Une autre idée d’implémentation est de proposer des options en fonction d’un bouton appuyé lorsque le programme est en cours de fonctionnement.

Imaginons par exemple les options suivantes :

- F1 : Help
- F2 : Mettre en pause / Reprendre
- F3 : Afficher / Retirer les codes couleurs
- F4 : Tri en fonction du meilleur temps au tour
- F5 : Tri en fonction du meilleur temps au tour total
- F6 : Tri en fonction du nom du pilote (id)
- F10 : Quitter

### 1.5.4 Phase d’essai entièrement libre : TODO !

Il serait possible, sans nécessairement y consacrer un temps considérable, de permettre aux différents pilotes de commencer et arrêter leurs séances d’essais libres lorsqu’ils le souhaitent voire même s’ils rouleront lors de la séance. La question concrète serait : *Est-ce que lors de la limite du temps imparti d’une séance d’essais libres, un pilote souhaite prendre le volant ou non et si oui, pour combien de tours ou combien de temps ?*

Cela correspondrait bien plus à une course de Formule 1 en condition réelle.

# 2 Conclusion

## 2.1 Daniel Olivier

L’avantage de ce projet est l’application de concepts multiples vue en cours théorique au courant du premier quadrimestre. Cela m’a permis de comprendre plus concrètement ce que ces concepts permettent de faire (allocation d’une zone mémoire, appel d’une zone mémoire, sémaphores, algorithmes, fork, etc…).

Ce projet m’a permis d’apprendre à programmer de façon plus assidue. Lors de l’écriture d’une nouvelle méthode, je testais systématiquement le projet et en cas de problème, je prenais le temps de relire le code (et si nécessaire, je testais différentes méthodes pour déboguer et avancer dans le projet). j’ai rencontré plusieurs difficultés de compréhension par rapport au cahier des charges ainsi que d’autres difficultés rencontrées. J’ai également découvert l’utilité de l’utilisation de quelques librairies, ainsi que d’une documentation disponible en ligne, me permettant de mieux comprendre certaines implémentations nécessaires.

## 2.2 Martin Michotte

Étant en année passerelle et donc n’ayant pas encore eu le cours d’OS de première année j’avais quelques craintes concernant la réalisation de ce projet, ou en tout cas de l’aide que je pouvais apporter au groupe.

Heureusement j’avais déjà programmé en C par le passé et il m’a donc fallu peu de temps pour réacquérir les bases. Cependant je n’avais aucune notion de programmation avec la gestion de multiples processus. J’ai dès lors du comprendre par moi-même le fonctionnement d’un `fork()` et de toutes ses contraintes ainsi que la gestion de la concurrence inter-processus.

Une fois à jour avec les autres membres du groupe j’ai pu participer activement à la réalisation du projet.

Concernant le projet de manière globale, bien qu’à première vue celui-ci paraissait complexe, une fois décortiqué en plusieurs petits morçeaux logiques, je ne le trouvais pas d’une grande difficulté. Bien évidemment nous avons rencontré quelques soucis mais aucun n’était insurmontable. L’utilisation de la mémoire partagée et des sémaphores était quelque chose d’un peu mystérieux au départ mais après avoir bien compris leur fonctionnement et après les avoir implémentés de manière adéquate, il s’est avéré que ceux-ci sont indispensables et assez simples d’utilisation.

Je peux donc conclure en disant que la difficulté de ce projet résidait pour moi dans le fait de devoir me mettre à niveau par rapport aux autres et dans la décomposition du projet en blocs logiques simples.

Ce projet m’a aussi permis d’apprécier un peu plus la programmation en C qui, auparavant, ne me plaisait qu’a moitié.

## 2.3 Morgan Valentin

Le projet est une bonne idée pour mettre en pratique ce qu’on voit en théorie, malgré qu’il m’aurais fallu un plus de cour théorique. Étant plutôt lent à programmer, j’aurais aimer avoir un quadrimestre dédié au langage C et au second quadrimestre avoir le projet afin de maximiser ma compréhension de la matière.

## 2.4 Martin Perdaens

En ayant pas encore eu le cours de OS de première année et ayant très peu programmer en C depuis que je suis dans l’informatique, j’avais des craintes sur la manière d’aborder le projet, surtout au niveau de mon aide au sein du groupe. Le plus dur était de comprendre comment le `fork()` et comment fonctionne plusieurs processus ensembles. Mais en sachant comment fonctionne la formule 1 cela m’a aidé pour la compréhension du projet, donc je pouvais plus me concentrer sur la compréhension du langage C.

Grâce aux autres membres j’ai pu mieux comprendre comment le langage C et apprendre
à mon rythme tout en essayant de comprendre les différentes caractéristique de celui-ci. Pour conclure le gros problème pour moi dans ce projet était de me mettre à niveau par rapport au autres membres du groupe, ce projet m’a permit d’apprendre un langage que je connaissais très peu et dans un future proche faire d’autres projets en C.

# 3 Références bibliographiques


- GUSTEDT Jens, **Modern C**. This is the 2nd edition (minor rev. 2) of this book, as of Oct. 10, 2019. The free version, sample code, links to Manning’s print edition and much more is available at (**Mordern C**)
- SELEZNEV Anton, librairie **libfort** disponible sur github : (**libfort**)
- LINUX man pages, disponible en ligne : https ://linux.die.net/man/
- Cours de Systèmes informatiques **SINF1252** donné aux étudiants en informatique à l’Université catholique de Louvain (UCL). Le cours est donné par Prof. Olivier Bonaventure et est disponible en ligne : (**SINF1252**)

# 4 Exemplaire du code

**Listing 3 :** child.c

```
#include "child.h"

int time_passed = 0;
int current_lap = 0;
F1_Car *vehicle;
Circuit circuit;

/*********************************************************
*                Gestion de crash d'une voiture          *
*********************************************************/

void car_crash() {
    if (car_crashed(10000000))
        vehicle->out = 1;
    else
        vehicle->out = 0;
}

/*********************************************************
*                Terminaison d'une étape                 *
*********************************************************/

int finished_running() {
    if (!strcmp(circuit.step_name, "RACE")) {
        return current_lap == circuit.number_of_laps;
    } else {
        return time_passed >= circuit.step_total_time;
    }
}

/********  fonction qui permet aux voitures de n'est pas courir à la même vitesse  *********/

int msleep(unsigned int tms) {
    return usleep(tms * 1000);
}

/** la fonction child fait tout ce qu'une voiture a à faire.
 *  càd tout ce qui est géré par l'enfant/voiture

*@param sem_t *sem c'est un sémaphore qui perment aux fils de n'est pas écrire en même temps
                    dans la mémoire partagée. Techniquement ils peuvent
                       mais on a choisi de procéder ainsi.
```

```
42 *@param F1_Car *car c'est la variable de type F1_Car qui pointe vers la
       mémoire partagée où les fils écrivent.
43 *@param int *car_names c'est la variable qui pointe vers le(s) tableau(
      x)qui contient les id.
44 */
45 void child(sem_t *sem, F1_Car *car, int *car_names) {
46 
47     random_seed(getpid());
48     vehicle = car;
49     vehicle->id = *car_names;
50 
51     while (!finished_running()) {
52 
53         //(!strcmp(circuit.step_name, "RACE")) ? sleep(10) : 0;
54 
55         sem_wait(sem);
56         vehicle->s1 = sector_range(30, 45, 10000000);
57         if (vehicle->best_s1 == 0 || vehicle->best_s1 > vehicle->s1) {
58             vehicle->best_s1 = vehicle->s1;
59         }
60         car_crash();
61         sem_post(sem);
62 
63         sem_wait(sem);
64         vehicle->s2 = sector_range(30, 45, 10000000);
65         if (vehicle->best_s2 == 0 || vehicle->best_s2 > vehicle->s2) {
66             vehicle->best_s2 = vehicle->s2;
67         }
68         car_crash();
69         sem_post(sem);
70 
71         sem_wait(sem);
72         vehicle->s3 = sector_range(30, 45, 10000000);
73 
74         int i = 1;
75         vehicle->stand = 0;
76         while (stand_probability(10)) {
77 
78             vehicle->s3 += stand_duration(1, 100);
79             i++;
80             vehicle->stand = 1;
81         }
82         if (vehicle->best_s3 == 0 || vehicle->best_s3 > vehicle->s3) {
83             vehicle->best_s3 = vehicle->s3;
84         }
85         car_crash();
86         msleep(80);
87 
88         vehicle->lap_time = vehicle->s1 + vehicle->s2 + vehicle->s3;
```

```
89          time_passed += vehicle->lap_time;
90
91          if (vehicle->best_lap_time == 0 ||
92              vehicle->best_lap_time > vehicle->lap_time)
93              vehicle->best_lap_time = vehicle->lap_time;
94          vehicle->lap++;
95          current_lap = vehicle->lap;
96          (time_passed >= circuit.step_total_time || current_lap ==
               circuit.number_of_laps) ? vehicle->done = 1 : 0;
97          sem_post(sem);
98          sleep(1);
99      }
100 }
```

**Listing 4 :** child.h

```
1  //
2  // Created by danny on 2/10/19.
3  //
4  #pragma once
5
6  #include "time.h"
7  #include "prng.h"
8  #include <semaphore.h>
9  #include <time.h>
10 #include <sys/shm.h>
11 #include <sys/sem.h>
12 #include <sys/ipc.h>
13 #include <sys/types.h>
14 #include <stdbool.h>
15 #include <sys/types.h>
16 #include <stdio.h>
17 #include <unistd.h>
18 #include <string.h>
19
20 #define NUMBER_OF_CARS 20
21
22 typedef struct Circuit {
23     char *step_name;
24     int step_total_time;
25     int number_of_laps;
26     int lap_km;
27     int number_of_cars;
28     int race_km;
29 } Circuit;
30
31
32 typedef struct F1_Car {
33     int id;
```

```
34      double lap_time;
35      double s1;
36      double s2;
37      double s3;
38      int best_s1;
39      int best_s2;
40      int best_s3;
41      int stand;
42      int out;
43      int lap;
44      int best_lap_time;
45      int done;
46  } F1_Car;
47
48  void child(sem_t *sem, F1_Car *car, int *car_names);
49
50  void car_crash();
51
52  int finished_running();
53
54  int msleep(unsigned int tms);
```

**Listing 5 :** display.c

```
1   //
2   // Created by danny on 5/10/19.
3   //
4
5   #include "display.h"
6
7
8   Circuit circuit;
9   F1_Car car_array[20];
10
11  /*************************************************************
12  *                        Gestion de trie                     *
13  *************************************************************/
14
15  int compare(const void *left, const void *right) {
16      const F1_Car *process_a = (F1_Car *) left;
17      const F1_Car *process_b = (F1_Car *) right;
18
19      if (strcmp(circuit.step_name, "RACE")) {
20          if (process_a->best_lap_time < process_b->best_lap_time)
21              return -1;
22          else if (process_a->best_lap_time > process_b->best_lap_time)
23              return 1;
24          else
25              return 0;
```

```
26      } else {
27          if (process_a->lap < process_b->lap)
28              return 1;
29          else if (process_a->lap > process_b->lap)
30              return -1;
31          else
32              return 0;
33      }
34  }
35
36  /*************************************************************
37  *                        Affichage                           *
38  *************************************************************/
39
40  void print_table() {
41
42      /********  Création de la table   *********/
43      ft_table_t *table = ft_create_table();
44
45      /********  Style des bordures  *********/
46      ft_set_border_style(table, FT_DOUBLE2_STYLE);
47
48      /********  Style des titres/headers de la table  *********/
49      ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_ROW_TYPE,
            FT_ROW_HEADER);
50      ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_CELL_TEXT_STYLE,
            FT_TSTYLE_BOLD);
51      ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
            FT_COLOR_CYAN);
52
53      ft_write_ln(table, "POSITION", "NAME", "S1", "S2", "SS3", "OUT", "
            PIT", "LAP", "LAP TIME", "BEST LAP TIME");
54
55      for (int i = 0; i < circuit.number_of_cars; i++) {
56          F1_Car current = car_array[i];
57
58          char sector1_time[10], sector2_time[10], sector3_time[10],
                lap_time[10], best_lap_time[10];
59
60          /********  Formatage du temps  *********/
61          to_string(current.s1, sector1_time);
62          to_string(current.s2, sector2_time);
63          to_string(current.s3, sector3_time);
64          to_string(current.lap_time, lap_time);
65          to_string(current.best_lap_time, best_lap_time);
66
67          /********  Affichage des données dans la variable table
                *********/
68          ft_printf_ln(table, "%d|%d|%.6s|%.6s|%.6s|%d|%d|%d|%.7s|%.7s",
```

```
            i + 1,
69                      current.id, sector1_time, sector2_time,
                           sector3_time, current.out,
70                      current.stand, current.lap, lap_time,
                           best_lap_time);
71
72          /********  Style pour le(s) voiture(s) au pit  *********/
73          (current.stand)
74          ? ft_set_cell_prop(table, i + 1, FT_ANY_COLUMN,
               FT_CPROP_CONT_FG_COLOR, FT_COLOR_DARK_GRAY)
75          : ft_set_cell_prop(table, i + 1, 6, FT_CPROP_CONT_FG_COLOR,
               FT_COLOR_LIGHT_GRAY);
76      }
77      /********  Style pour les premiers voitures *********/
78      ft_set_cell_prop(table, 1, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
           FT_COLOR_LIGHT_GREEN);
79      ft_set_cell_prop(table, 2, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
           FT_COLOR_LIGHT_BLUE);
80      ft_set_cell_prop(table, 3, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
           FT_COLOR_LIGHT_YELLOW);
81
82      /********  Deuxiéme table à afficher *********/
83
84      ft_table_t *second_table = ft_create_table();
85      ft_write_ln(second_table, "SECTORS", "NAME", "TIME");
86      ft_set_border_style(second_table, FT_DOUBLE2_STYLE);
87
88      ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN, FT_CPROP_ROW_TYPE,
            FT_ROW_HEADER);
89      ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN,
           FT_CPROP_CELL_TEXT_STYLE, FT_TSTYLE_BOLD);
90      ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN,
           FT_CPROP_CONT_FG_COLOR, FT_COLOR_CYAN);
91
92      char s1_time[10], s2_time[10], s3_time[10], winner[10];
93
94      to_string(car_array[best_sector("S1")].best_s1, s1_time);
95      to_string(car_array[best_sector("S2")].best_s2, s2_time);
96      to_string(car_array[best_sector("S3")].best_s3, s3_time);
97      to_string(car_array[best_car()].best_lap_time, winner);
98
99      ft_printf_ln(second_table, "%s|%d|%s", "S1", car_array[best_sector(
           "S1")].id, s1_time);
100     ft_printf_ln(second_table, "%s|%d|%s", "S2", car_array[best_sector(
           "S2")].id, s2_time);
101     ft_printf_ln(second_table, "%s|%d|%s", "S3", car_array[best_sector(
           "S3")].id, s3_time);
102
103     /********  Affichage du gagnant lors de la course de dimanche
```

```
         *********/
104      (!strcmp(circuit.step_name, "RACE")) ?
105      ft_printf_ln(second_table, "%s|%d|%.7s", "Winner", car_array[
            best_car()].id, winner) : 0;
106
107      /********  Rafraichissement des données *********/
108      clear();
109
110      /********  Affichage de la variable table et second_table en
            console *********/
111      printf("%s", ft_to_string(table));
112      printf("%s", ft_to_string(second_table));
113
114      /******** Destruction de deux tables  *********/
115      ft_destroy_table(table);
116      ft_destroy_table(second_table);
117  }
118
119  /** la fonction best_sector sert à trouver qui a le meilleur temps dans
         chacun des secteurs.
120
121  *@param char sector[] c'est nom du sector.
122  *@return int id c'est une variable qui contient l'id de la voiture qui
        a le meilleur temps dans un sector donné.
123  */
124
125  int best_sector(char sector[]) {
126      int sector_number = 0, id = 0;
127      for (int i = 0; i < circuit.number_of_cars; i++) {
128
129          if (!strcmp(sector, "S1")) {
130              if (sector_number == 0 || car_array[i].best_s1 <
                    sector_number) {
131                  sector_number = car_array[i].best_s1;
132                  id = i;
133              }
134          } else if (!strcmp(sector, "S2")) {
135              if (sector_number == 0 || car_array[i].best_s2 <
                    sector_number) {
136                  sector_number = car_array[i].best_s2;
137                  id = i;
138              }
139          } else if (!strcmp(sector, "S3")) {
140              if (sector_number == 0 || car_array[i].best_s3 <
                    sector_number) {
141                  sector_number = car_array[i].best_s3;
142                  id = i;
143              }
144          }
```

```
145     }
146     return id;
147 }
148 
149 /** la fonction best_car sert à trouver qui a le tour le plus rapide à
       la fn du grand prix.
150 
151 *@return int id c'est une variable qui contient l'id de la voiture qui
       a le meilleur tour.
152 */
153 
154 int best_car() {
155     int win = 0, id = 0;
156     for (int i = 0; i < circuit.number_of_cars; i++) {
157         if (win == 0 || car_array[i].best_lap_time < win) {
158             win = car_array[i].best_lap_time;
159             id = i;
160         }
161     }
162     return id;
163 }
164 
165 /** la fonction best_car sert à trouver quel voiture a crash/ out de la
       course.
166 
167 *@return une variable positif s'il y a une voiture est out sinon la
       valeur 0 est retournée.
168 */
169 
170 int finished() {
171     for (int i = 0; i < circuit.number_of_cars; ++i) {
172         if (car_array[i].out) {
173             return 1;
174         }
175     }
176     return 0;
177 }
178 
179 /** la fonction display sert à afficher et trier les données se
       trouvant dans la mémoire partagée.
180  *  Avant de trier les données on fait une copie, puis on trie la copie
       , à la fin on sauvegarde
181  *  le fichier qui aura comme nom l'étape en exécution.
182 
183 *@param sem_t *sem c'est un sémaphore pour sécuriser les données lors
       de la copie.
184 *@param F1_Car *data c'est la variable de type F1_Car qui pointe vers
       la mémoire partagée.
185 */
```

```
186
187 void display(sem_t *sem, F1_Car *data) {
188
189     init_window();
190
191     while (1) {
192         sem_wait(sem);
193         memcpy(car_array, data, sizeof(F1_Car) * circuit.number_of_cars
                );
194         sem_post(sem);
195         qsort(car_array, circuit.number_of_cars, sizeof(F1_Car),
                compare);
196         if (finished() || car_array[9].done) {
197             break;
198         }
199         print_table();
200         sleep(1);
201     }
202     sleep(1);
203     save_ranking();
204     terminate_window();
205 }
```

**Listing 6 :** display.h

```
1  //
2  // Created by danny on 5/10/19.
3  //
4
5  #pragma once
6
7  #include <semaphore.h>
8  #include "child.h"
9  #include "window.h"
10 #include "time.h"
11 #include "files.h"
12 #include "../lib/fort.h"
13 #include <stdio.h>
14 #include <string.h>
15 #include <unistd.h>
16
17 void display(sem_t *_sem, F1_Car *data);
18 int compare(const void *left, const void *right);
19 int best_sector(char sector[]);
20 int best_car();
```

**Listing 7 :** files.c

```c
//
// Created by danny on 26/10/19.
//

#include "files.h"


Circuit circuit;
F1_Car car_array[20];

/** la fonction save_ranking créer un fichier qui a comme nom l'étape
 *  en exécution. Ce fichier est créé à la fin de l'étape.
*/

void save_ranking() {

    FILE *file = fopen(circuit.step_name, "w");

    if (file == NULL) perror("fopen failed !"), exit(EXIT_FAILURE);

    for (int i = 0; i < circuit.number_of_cars; i++) {
        char best_lap_str[10];
        to_string(car_array[i].best_lap_time, best_lap_str);
        fprintf(file, "%d --> %s\n", car_array[i].id, best_lap_str);
    }

    if (fclose(file) != 0) perror("fclose failed !"), exit(EXIT_FAILURE
       );
}

/** la fonction read_files lit un fichier passé en paramétre pour
   stocker les qualifiés et
*   les élimninés dans les tableaux passé en paramétre également.

*@param int qualified_cars[] un tableau pour stocker les voiutures
   qualifiés
*@param int race_ranking[] un tableau pour stocker les voiutures en
   fonction du classement de dimanche
*@param int last_cars_of_Q1[] un tableau pour stocker les élimninés de
   Q1
*@param int last_cars_of_Q2[] un tableau pour stocker les élimninés de
   Q2
*@param char file_to_read[] le fichier à lire.
*@param int lines_to_read le nombre de lignes à lire dans ce fichier
   passé en paramétre.

*/

void
```

```
43 read_files(int qualified_cars[], int race_ranking[], int
      last_cars_of_Q1[], int last_cars_of_Q2[], char file_to_read[],
44             int lines_to_read) {
45 
46     int file_size = find_size(file_to_read);
47     char absolute_path[file_size];
48     getcwd(absolute_path, file_size);
49     char full_absolute_path[file_size];
50     sprintf(full_absolute_path, "%s/%s", absolute_path, file_to_read);
51 
52     FILE *cmd;
53     char result[NUMBER_OF_CARS];
54     char grep_file_result[file_size];
55     sprintf(grep_file_result, "egrep -o '^[0-9]{1,2}' '%s'",
          full_absolute_path);
56 
57     cmd = popen(grep_file_result, "r");
58     if (cmd == NULL) perror("popen failed !"), exit(EXIT_FAILURE);
59 
60     int i = 0, j = 0, k = 0;
61     while (fgets(result, sizeof(result), cmd)) {
62 
63         if (i < lines_to_read) {
64             qualified_cars[i] = atoi(result);
65             if (strcmp(file_to_read, "Q3") == 0) {
66                 race_ranking[i] = atoi(result);
67             }
68             i++;
69         } else {
70             if (strcmp(file_to_read, "Q1") == 0) {
71                 last_cars_of_Q1[j] = atoi(result);
72                 j++;
73             } else if (strcmp(file_to_read, "Q2") == 0) {
74                 last_cars_of_Q2[k] = atoi(result);
75                 k++;
76             }
77         }
78     }
79 
80     if (pclose(cmd) != 0) perror("pclose failed !"), exit(EXIT_FAILURE)
          ;
81 }
82 
83 /** la fonction find_size calcule la taille du fichier passé en paramé
      tre
84 
85 *@param char *file_name le fichier à lire.
86 *@return int size qui est la taille du fichier.
87 */
```

```
88
89  int find_size(char *file_name) {
90
91      FILE *file = fopen(file_name, "r");
92
93      if (file == NULL) {
94          printf("%s '%s' %s", "File previous to", circuit.step_name, "
                NOT found !\n"), exit(EXIT_FAILURE);
95      }
96
97      fseek(file, 0L, SEEK_END);
98
99      int size = ftell(file);
100     if (fclose(file) != 0) perror("fclose failed !"), exit(EXIT_FAILURE
            );
101     return size;
102 }
103
104 /** la fonction save_eliminated_cars sauvegarde les voitures élimninés
        dans un fichier.
105
106 *@param char file_to_save[] le fichier qui va contenir les voitures é
        liminés.
107 *@param int array[] le tableau qui contient les voitures éliminés.
108 */
109
110 void save_eliminated_cars(char file_to_save[], int array[]) {
111     FILE *file = fopen(file_to_save, "w");
112
113     if (file == NULL)
114         perror("fopen failed !"), exit(EXIT_FAILURE);
115
116     for (int i = 0; i < 5; i++) {
117         fprintf(file, "%d\n", array[i]);
118     }
119
120     if (fclose(file) != 0)
121         perror("fclose failed !"), exit(EXIT_FAILURE);
122 }
123
124 /** la fonction read_eliminated_cars lit les voitures élimninés depuis
        un fichier
125  *  vers un tableau qui va contenir le classement de la course de
        dimanche.
126
127 *@param char file_to_read[] le fichier à lire qui contient les éliminés
128 *@param int array[] le tableau qui contient le classmeent de la course
        de dimanche.
129 */
```

```
130
131 void read_eliminated_cars(char file_to_read[], int array[]) {
132
133     char results[5];
134
135     FILE *file = fopen(file_to_read, "r");
136
137     if (file == NULL) perror("fopen failed !"), exit(EXIT_FAILURE);
138
139     int i = 15, j = 10;
140     while (fgets(results, sizeof(results), file)) {
141
142         if (strcmp(file_to_read, "lastQ1") == 0) {
143             array[i] = atoi(results);
144             i++;
145         }
146
147         if (strcmp(file_to_read, "lastQ2") == 0) {
148             array[j] = atoi(results);
149             j++;
150         }
151     }
152
153     if (fclose(file) != 0)
154         perror("fclose failed !"), exit(EXIT_FAILURE);
155 }
```

**Listing 8 :** files.h

```
1  #pragma once
2
3  #include "child.h"
4  #include "window.h"
5  #include "time.h"
6  #include "../lib/fort.h"
7  #include <stdio.h>
8  #include <string.h>
9  #include <unistd.h>
10 #include <semaphore.h>
11
12 void save_ranking();
13
14 int find_size(char *file_name);
15
16 void read_files(int qualified_cars[], int race_ranking[], int
       last_cars_of_Q1[], int last_cars_of_Q2[], char file[], int
       lines_to_read);
17
18 void save_eliminated_cars(char file_to_save[], int array[]);
```

```
19
20 void read_eliminated_cars(char file_to_read[], int array[]);
```

**Listing 9 :** main.c

```
1  #include "child.h"
2  #include "display.h"
3  #include <getopt.h>
4  #include <locale.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <sys/wait.h>
8  #include <unistd.h>
9
10 Circuit circuit;
11 F1_Car *car;
12
13 /********  Tableau par défaut des id des voitures si on est ni au Q2,
      Q3, RACE  *********/
14 int car_names[NUMBER_OF_CARS] = {44, 77, 5, 7, 3, 33, 11, 31, 18, 35,
15                                 27, 55, 10, 28, 8, 20, 2, 14, 9, 16};
16
17 /**
18 *  qualified_cars va stocker les voitures qualifiés.
19 *  race_ranking va stocker le classement désiré pour la course de
      dimanche.
20 *  last_cars_of_Q1 va stocker les élimninés au Q1.
21 *  last_cars_of_Q2 va stocker les élimninés au Q2.
22 */
23 int qualified_cars[15], race_ranking[20], last_cars_of_Q1[15],
      last_cars_of_Q2[10];
24
25 /********  Gestion d'erreur dans le paramétrage du programme   ********
      */
26
27 void print_usage() {
28     printf("%s", "Usage: ./prog --day [dayName] --step [stepName]\n");
29     printf("%s", "Usage: For race you can specify the lap length, by
          default it's 7km !\n");
30     printf("%s", "Usage: ./prog --day [dayName] --step [stepName] --
          length [number]\n");
31     printf("%s", "Use the --help command for more information. \n");
32     exit(EXIT_FAILURE);
33 }
34
35 /********  Manuel du programme  *********/
36
37 void help() {
38     printf("\n%s\n\n", "These are some commands used to run this
```

```
        program.");
39      printf("%s\n", "For P sessions : \t There are run on fridays but P3
            on sat. Use the --day command.");
40      printf("%s\n", "\t\t\t followed by a day name and which step needs
        to be runned for the P sessions.");
41      printf("%s\n\n", "\t\t\t --day fri --step P2 for instance.");
42      printf("%s\n", "For Q sessions : \t There are run on saturdays, use
            the --day command.");
43      printf("%s\n", "\t\t\t followed by a day name and which step needs
        to be runned for the Q sessions.");
44      printf("%s\n\n", "\t\t\t --day sat --step Q3 for instance.");
45      printf("%s\n", "For the RACE session : \t It's run on sundays, use
        the --day command.");
46      printf("%s\n", "\t\t\t followed by a day name. Here you can specify
            the race's lap length.");
47      printf("%s\n", "\t\t\t by default it's 7km, the --length command is
            optional.");
48      printf("%s\n\n", "\t\t\t --day sun --step RACE --length 10 for
        instance. ");
49      exit(EXIT_SUCCESS);
50  }
51
52  int main(int argc, char **argv) {
53
54      signal(SIGINT, return_cursor);
55
56      //valeurs par défaut
57      circuit.lap_km = 7;
58      circuit.race_km = 305;
59
60      /*************************************************
61      *            Paramétrage du programme            *
62      *************************************************/
63
64      int user_km = 0;
65      char day_name[5], step_name[5];
66
67      static struct option long_options[] = {{"day",    required_argument
        , NULL, 'd'},
68                                             {"step",   required_argument
                                                , NULL, 's'},
69                                             {"length", required_argument
                                                , NULL, 'l'},
70                                             {"help",   no_argument, 0,
                                                 'h'},
71                                             {NULL, 0,
                                                                 NULL,
                                                     0}};
72
```

```
73      char opt;
74      while ((opt = getopt_long(argc, argv, "hd:s:l:", long_options, NULL
            )) != EOF) {
75          switch (opt) {
76              case 'h':
77                  help();
78                  break;
79              case 'd':
80                  strcpy(day_name, optarg);
81                  break;
82              case 's':
83                  strcpy(step_name, optarg);
84                  break;
85              case 'l':
86                  user_km = atoi(optarg);
87                  break;
88              default:
89                  print_usage();
90          }
91      }
92
93      /******** Friday *********/
94      if (!strcmp(day_name, "fri")) {
95
96          if (!strcmp(step_name, "P1")) {
97
98              /********  Assignation du nbr de voitures, du nom de l'é
                    tape et le temps de l'étape *********/
99              circuit = (Circuit) {.number_of_cars = 20, .step_name = "P1
                    ", .step_total_time = minutes_to_ms(90)};
100
101         } else if (!strcmp(step_name, "P2")) {
102
103             circuit = (Circuit) {.number_of_cars = 20, .step_name = "P2
                    ", .step_total_time = minutes_to_ms(90)};
104
105         } else {
106             print_usage();
107         }
108
109         /******** Saturday *********/
110     } else if (!strcmp(day_name, "sat")) {
111
112         if (!strcmp(step_name, "P3")) {
113
114             circuit = (Circuit) {.number_of_cars = 20, .step_name = "P3
                    ", .step_total_time = minutes_to_ms(60)};
115
116         } else if (!strcmp(step_name, "Q1")) {
```

```
117
118             circuit = (Circuit) {.number_of_cars = 20, .step_name = "Q1
                    ", .step_total_time = minutes_to_ms(18)};
119
120         } else if (!strcmp(step_name, "Q2")) {
121
122             circuit = (Circuit) {.number_of_cars = 15, .step_name = "Q2
                    ", .step_total_time = minutes_to_ms(15)};
123
124             /********  Lecture des 15 premiers voitures au Q1  ********
                    */
125             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q1", 15);
126
127         } else if (!strcmp(step_name, "Q3")) {
128
129             circuit = (Circuit) {.number_of_cars = 10, .step_name = "Q3
                    ", .step_total_time = minutes_to_ms(12)};
130
131             /********  Lecture des 10 premiers voitures au Q2  ********
                    */
132             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q2", 10);
133
134         } else {
135             print_usage();
136         }
137         /******** Sunday *********/
138     } else if (!strcmp(day_name, "sun")) {
139
140         if (!strcmp(step_name, "RACE")) {
141
142             /********  Assignation du nbr de voitures, du nom de l'é
                    tape et le temps de l'étape *********/
143             circuit.number_of_cars = 20;
144             circuit.step_name = "RACE";
145             circuit.step_total_time = minutes_to_ms(120);
146
147             /********  Lecture des 10 premiers voitures au Q3  ********
                    */
148             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q3", 10);
149
150             /********  Lecture du fichier lastQ2 et attribution de la
                    10iéme à la 15iéme place  *********/
151             read_eliminated_cars("lastQ2", race_ranking);
152
153             /********  Lecture du fichier lastQ1 et attribution de la
                    15iéme à la 20iéme place   *********/
```

```
154             read_eliminated_cars("lastQ1", race_ranking);
155
156             /********  Lecture du fichier Q3 et attribution de la 1ére
                   à la 10iéme place   *********/
157             read_eliminated_cars("Q3", race_ranking);
158
159             /********  La longueur du circuit est 7km *********/
160             if (user_km == 0) {
161                 circuit.number_of_laps = circuit.race_km / circuit.
                       lap_km;
162
163                 /********  La longueur du circuit a été changé par l'
                       utilisateur *********/
164             } else if (user_km > 0) {
165                 circuit.number_of_laps = circuit.race_km / user_km;
166             } else {
167                 print_usage();
168             }
169         } else {
170             print_usage();
171         }
172     } else {
173         print_usage();
174     }
175
176     /*************************************************
177     *              Sauvegarde des fichiers            *
178     *************************************************/
179
180     /******* Si on est au Q2, les éliminés du Q1 sont sauvegardés dans
           le fichier lastQ1 ******/
181     !strcmp(circuit.step_name, "Q2") ?
182     save_eliminated_cars("lastQ1", last_cars_of_Q1) :
183
184     /******* Si on est au Q3, les éliminés du Q2 sont sauvegardés dans
           le fichier lastQ2 ******/
185     !strcmp(circuit.step_name, "Q3") ?
186     save_eliminated_cars("lastQ2", last_cars_of_Q2) :
187     NULL;
188
189     /*************************************************
190     *           Création de la mémoire partagée        *
191     *************************************************/
192
193     int struct_shm_id = shmget(IPC_PRIVATE, sizeof(F1_Car) * circuit.
           number_of_cars, 0600 | IPC_CREAT);
194     if (struct_shm_id == -1) {
195         perror("shmget failed !");
196         exit(EXIT_FAILURE);
```

```
197     }
198
199     car = shmat(struct_shm_id, NULL, 0);
200     if (car == (void *) (-1)) {
201         perror("shmat failed !");
202         exit(EXIT_FAILURE);
203     }
204
205     /*******************************************************
206     *                 Création des sémaphores              *
207     *******************************************************/
208
209     int sem_shm_id = shmget(IPC_PRIVATE, sizeof(sem_t), 0600 |
            IPC_CREAT);
210     if (sem_shm_id == -1) {
211         perror("shmget failed !");
212         exit(EXIT_FAILURE);
213     }
214     sem_t *sem = shmat(sem_shm_id, NULL, 0);
215     if (sem == (void *) (-1)) {
216         perror("shmat failed !");
217         exit(EXIT_FAILURE);
218     }
219
220     sem_init(sem, 1, 1);
221
222     /*********************************************************
223      *                Création des fils/voitures              *
224      *********************************************************/
225
226     int i;
227     pid_t pid = 0;
228     for (i = 0; i < circuit.number_of_cars; i++) {
229         pid = fork();
230         if (pid == 0)
231             break;
232     }
233
234     switch (pid) {
235
236         case -1:
237             /********  échec du fork *********/
238             fprintf(stderr, "fork failed !");
239             exit(EXIT_FAILURE);
240
241         case 0:
242             /********  Si on est au Q2 ou Q3 attribution des id par le
                    tableau des qualifiés  *********/
243             (!strcmp(circuit.step_name, "Q2") || !strcmp(circuit.
```

```
                step_name, "Q3")) ?
244             child(sem, &car[i], &qualified_cars[i]) :
245
246             /********  Si on est au RACE attribution des id par le
                   tableau race_ranking  *********/
247             !strcmp(circuit.step_name, "RACE") ?
248             child(sem, &car[i], &race_ranking[i]) :
249
250             /********  Si on est aux autres étapes attribution des id
                   par le tableau car_names  *********/
251             child(sem, &car[i], &car_names[i]);
252
253             exit(EXIT_SUCCESS);
254
255         default:
256             /********  Appel de la fonction display qui va afficher les
                    données  *********/
257             display(sem, car);
258
259             /********  wait for children to finish  *********/
260             for (int j = 0; j < circuit.number_of_cars; j++) {
261                 wait(NULL);
262             }
263     }
264     /********  Détachament des segments de mémoire  *********/
265     shmdt(car);
266
267     /********  Supprimer la mémoire partagée  *********/
268     shmctl(struct_shm_id, IPC_RMID, NULL);
269
270     /********  Destruction des sémaphores  *********/
271     sem_destroy(sem);
272     shmdt(sem);
273     shmctl(sem_shm_id, IPC_RMID, NULL);
274     exit(EXIT_SUCCESS);
275 }
```

**Listing 10 :** prng.c

```
1
2  #include "prng.h"
3
4  /********  Création des temps des différentes pour les voitures
       *********/
5  void random_seed(unsigned int seed) { srand(seed); }
6
7  /********  la probabilité d'aller au stand  *********/
8  int sector_range(int min, int max, int crashing_probability) {
9    car_crashed(crashing_probability);
```

```
10    return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
11  }
12
13  /********  le temps passé au stand  *********/
14  int stand_duration(int min, int max) {
15    return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
16  }
17
18  /********  la probabilité d'aller au stand  *********/
19  int stand_probability(int seed) { return rand() % seed == 0; }
20
21  /********  runs in a certain probability, like 1/seed  *********/
22  int car_crashed(unsigned int seed) { return rand() % seed == 0; }
```

**Listing 11 :** prng.h

```
1  //
2  // Created by danny on 4/10/19.
3  //
4
5  #pragma once
6
7  #include <stdlib.h>
8
9  void random_seed(unsigned int seed);
10
11 int sector_range(int min, int max, int crashing_probability);
12
13 int stand_duration(int min, int max);
14
15 int stand_probability(int seed);
16
17 int car_crashed(unsigned int seed);
```

**Listing 12 :** time.c

```
1  //
2  // Created by danny on 19/10/19.
3  //
4
5  #include "time.h"
6
7  /********  Conversion des données en temps réel  *********/
8  Time time_to_ms(int msec) {
9      Time formated_time;
10     div_t result;
11
12     result = div(msec, 60000);
```

```
13      formated_time.min = result.quot;
14      msec = result.rem;
15
16      result = div(msec, 1000);
17      formated_time.sec = result.quot;
18      msec = result.rem;
19
20      formated_time.msec = msec;
21      return formated_time;
22  }
23
24  int minutes_to_ms(int minutes) { return minutes * 60000; }
25
26  /********  Formatage du temps  *********/
27  void to_string(int msec, char *str) {
28      Time time = time_to_ms(msec);
29      (time.min) ? sprintf(str, "%d':%d\"%d", time.min, time.sec, time.
           msec)
30                 : sprintf(str, "%d\":%d", time.sec, time.msec);
31  }
```

**Listing 13 :** time.h

```
1  #pragma once
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  typedef struct Time {
7      int min;
8      int sec;
9      int msec;
10 } Time;
11
12 Time time_to_ms(int msec);
13
14 int minutes_to_ms(int minutes);
15
16 void to_string(int msec, char *str);
```

**Listing 14 :** window.c

```
1
2  #include "window.h"
3
4  void init_window() { printf("\e[?1049h\e[?7l\e[?25l\e[2J\e[1;52r"); }
5
6  void clear() { printf("\e[55H\e[9999C\e[1J\e[1;55r"); }
```

```
7
8  void terminate_window() { printf("\e[?7h\e[?25h\e[2J\e[;r\e[?1049l"); }
9
10 void return_cursor() {
11   clear();
12   terminate_window();
13   exit(EXIT_SUCCESS);
14 }
```

**Listing 15 :** window.h

```
1  #pragma once
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  void init_window();
7
8  void terminate_window();
9
10 void clear();
11
12 void return_cursor();
```