EPHEC

Rapport de projet “Formule 1”
Groupe 4

OS travaux pratiques

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

07 dec 2019

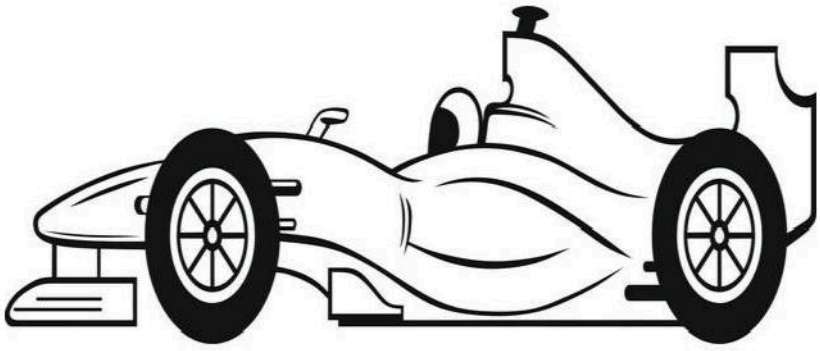


Table des matières

Liste des tableaux

Table des figures

Listings

1 Rapport du projet : F1

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 et simuler cela par le bais du langage de programmation C. Nous devons utiliser la ligne de commande afin d'afficher la grille des positions tout au long des séances d'essais libres, des qualifications ainsi que de la course.

Nous afficherons dès lors :

- le meilleur temps au tour par voiture
- les noms(id) de voitures ainsi que leurs positions
- qui a le meilleur temps dans chacun des secteurs
- le nombre de tours(lap) effectué par voiture
- les voitures aux stand(pit)
- …

De plus, nous devons appliquer des concepts vus en cours comme :

- utilisation de processus père-fils via la fonction `fork()` (cours de 1ère année)
- utilisation de la mémoire partagée pour le partage de données entre processus
- utilisation de sémaphores pour gérer la synchronisation des processus.

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.

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 :

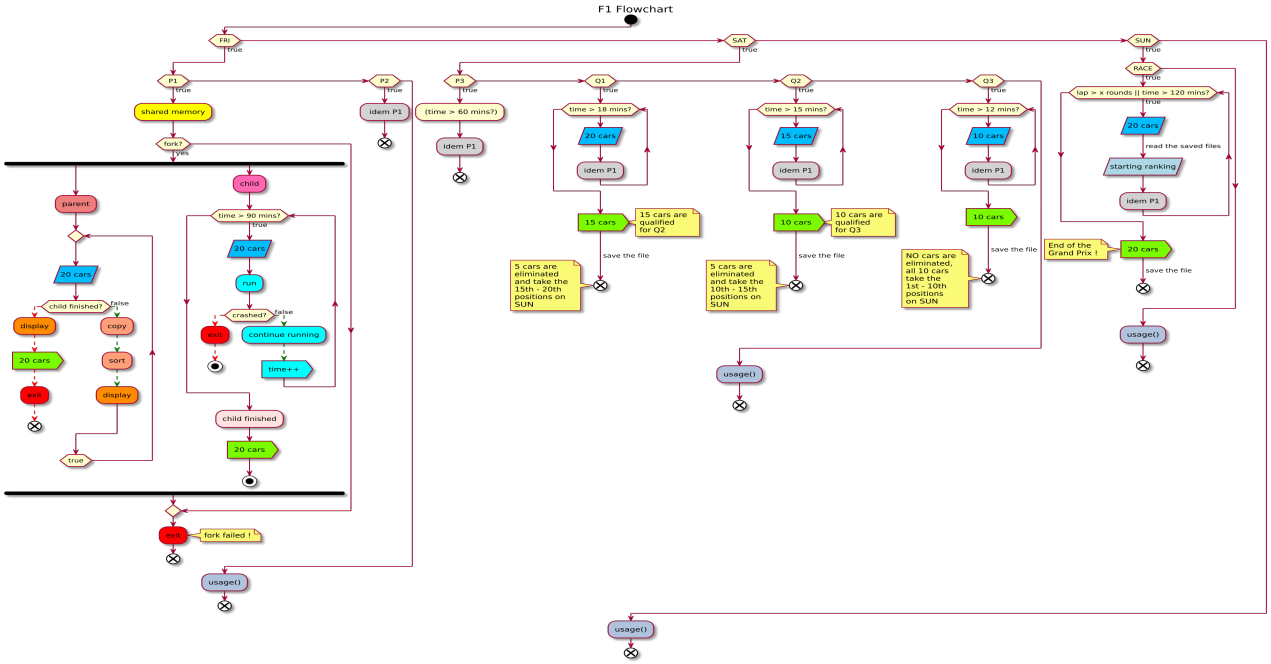
- ☒ Réaliser le programme en C sous Linux
- ☒ Utiliser la mémoire partagée comme moyen de communication inter-processus
- ☒ Utiliser les sémaphores pour synchroniser l'accès à la mémoire partagée

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



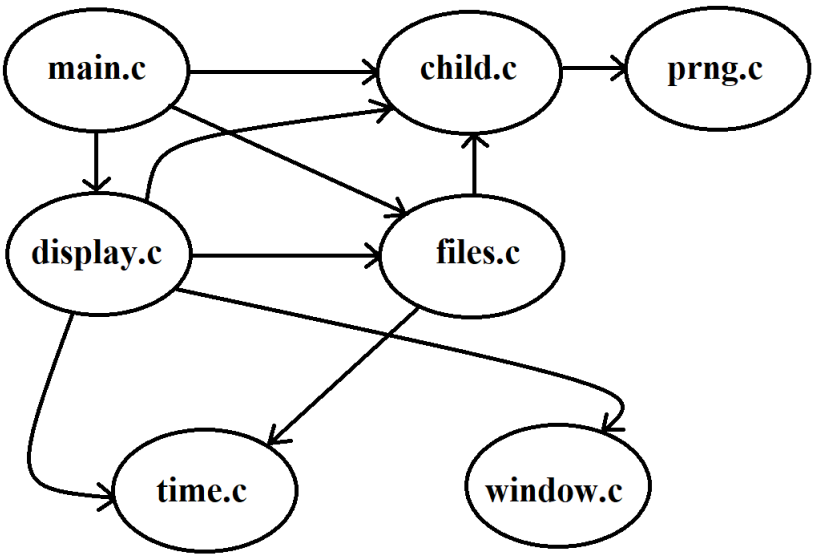
Malheureusement vu la taille du l'image, on était obligé de l'imprimer à part sur une feuille A3 qui se trouve en annexe du rapport. Comme vous pouvez le voir sur l'image le projet est divisé en 3 parties. Vendredi, samedi et dimanche. P1 et P2 sont exécutés le vendredi, samedi on a les étapes suivantes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

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

1.3.2 Découpage en plusieurs fichiers

Tous les fichiers C du projet

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 découpé le code en plusieurs fichiers nous a beaucoup aidé lors du débogage des problèmes rencontrés au fur à mesure de notre avancement.

1.3.3 Fichiers principaux

Sans surprise, le fichier le plus important 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 participeront au Grand Prix. Le fichier **display.c**. sert principalement à afficher les données triées en console.

Le fichier **child.c** contient quant à lui tout ce qui est propre à une voiture. C'est dans ce fichier que la voiture va "s'exécuter" une fois créée par le `fork()` du **main.c**. Finalement on a aussi un fichier **files.c** qui se charge de la creation, écriture et lecture des fichiers texte contenant les données des classements après chaque session.

1.3.4 Description de la méthode de travail

Premièrement, nous avons décidé de travailler avec un logiciel de gestion de version 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 :

1. **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.

2. **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 au début du projet qu'il était primordiale d'avoir d'abord une voiture qui tourne et affiche convenablement 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 ne supportent pas le rafraîchissement.

L'un de plus gros challenge rencontré était le rafraîchissement 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.

Après avoir réussi à obtenir une voiture qui tournait correctement et un affichage qui nous convenait, on est passé à la création de la mémoire partagée. Évidemment créer cette dernière sans avoir au moins 2 processus qui tournent n'a pas de de sens. On est resté bloqué sur la mémoire partagée pendant plusieurs semaines.

Quelquels problèmes rencontrés

- des processus qui ne terminent jamais (le père et les fils restaient bloqué après à la fin d'une session)
- des processus zombie
- des voitures qui tournaient plus

Néanmoins, tous ces problèmes ont été résolu.

À ce stade, on avait une mémoire partagée fonctionnelle et un affichage digne de ce nom. L'étape suivante était donc de trier les données afin d'obtenir un classement correct pour la course du dimanche. Ceci implique plusieurs choses :

- La mise en place du paramétrage de notre programme.
- La création d'un fichier par classement en fonction de l'étape.

1.4 Explication des particularités du code

1.4.1 Fonctionnalités du code

Pour le paramétrage du programme, nous avons décidé d'utiliser un "parser" de 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.

Voir code en annexe dans le fichier **main.c** pour l'implémentation.

Notre programme prend 4 options d'argument :

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.

Voici quelques exemples de lancement de notre programme avec des arguments

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
```

Pour avoir le manuel (help) :

`./prog --help` ou `./prog -h`

Concrètement en exécutant notre programme, on lance la phase sélectionnée pour chacune des voitures participantes. Les voitures participantes vont alors générer des temps aléatoires pour chaque secteur, toutes les informations relatives à la voiture sont, à chaque tour, 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 en console. Les informations représentée dans ce dernier dépendent de l'étape concernée. Ce tableau est également trié en fonction du meilleur temps au tour par voiture 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, la position des pilotes dans le ranking est sauvegardée dans un fichier. Ce fichier sera lu 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 cette dernière 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(...)`.

Voir code dans le fichier **main.c**. pour l'implémentation de ces appels systèmes.

Notre mémoire partagée est constituée d'un `struct` comportant les informations propre à une voiture :

Le struct partagée où les différents pilotes vont écrire leurs données

Listing 1 : shared struct

```
typedef struct F1_Car {
    int id;
    double lap_time;
    double s1;
    double s2;
    double s3;
    int best_s1;
    int best_s2;
    int best_s3;
    int stand;
    int out;
    int lap;
    int best_lap_time;
    int done;
} 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 donc à nous de régler ce problème de concurrence inter-processus.

Dans notre cas un processus fils peut être vu comme un "**rédacteur**" tandis que le processus père comme un "**lecteur**". Deux processus fils ne poserons jamais de problème de concurrence car ceux-ci

écrivent dans une adresse mémoire différente. Il y a donc seulement un risque de concurrence si le père lit une adresse mémoire en cours d'écriture par un fils. Pour résoudre ce problème, nous avons utilisé des sémaphores.

Il y a plusieurs variété de sémaphores, les sémaphores du *System V* et les sémaphores *POSIX*. On peut encore différencier 2 type de sémaphore POSIX :

1. **unnamed semaphores** (`sem_init (...)`, `sem_destroy (...)`, `sem_wait (...)`, `sem_post (...)`)
2. **named semaphores** (`sem_open (...)`, `sem_unlink (...)`, `sem_wait (...)`, `sem_post (...)`)

Nous avons décidé d'utilise des sémaphores POSIX de type **unnamed semaphores** disponibles dans la librairie standard C (GNU).

Pour pouvoir utiliser un sémaphore, il faut d'abord l'initialiser. Cela se fait en utilisant la fonction `sem_init(...)` qui prend comme arguments :

- un pointeur vers le sémaphore à initialiser
- un flag 'pshared' indiquant si ce sémaphore sera partagé entre les threads d'un processus ou entre plusieurs processus
- 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.

Voir le code en annexe dans le fichier **main.c** pour l'implémentation de ces fonctions.

1.4.4 Libération des ressources de l'ordinateur

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

I. Détachement" de la mémoire partagée et suppression de celui-ci

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. Lorsqu'un processus se termine, tous les segments auxquels il était attaché sont détachés lors de l'appel `exit(...)`.

Néanmoins, 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 :

1. un identifiant de segment de mémoire partagée (retourné par `shmget(...)`).
2. 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.
3. le troisième est un pointeur *buf* sur une structure paratagée. Avec la commande *IPC_RMID*, buf pourra être positionné à NULL dans ce cas.

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.

II. Suppression des sémaphore

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 bous permettant 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.

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 une copie du programme au moment de l'appel.

Voir le code en annexe dans le fichier **main.c** pour la création des processus fils.

I. Rôle du processus père

Dans notre cas, nous avons un processus père qui donne naissance à un nombre de processus fils en fonction de l'étape choisie.

Le processus père est chargé de lire les données stockées en mémoire partagée. Il s'occupe également du tri des voitures ainsi que de l'affichage. En fin de session, il sauvegarde les informations dans les fichiers correspondant.

II. Rôle des processus fils

Les fils sont seulement chargés de "courir". C'est à dire, d'exécuter les étapes d'un week-end complet de grand prix de Formule 1. Pour ce faire nous avons utilisé une boucle `while(...)` qui tourne tant que la voiture n'a pas dépassé le temps de session autorisé ou, si nous sommes en course, tant que la voiture n'a pas fait les X tours de circuit demandé.

Le nombre de tours à faire est déterminé par la longueur du circuit qui varie en fonction l'option ***–length*** passé en argument du programme. Si ce dernier n'est pas fourni une valeur par défaut de 7km lui est attribuée.

Le code du fils se trouve dans le fichier **child.c**.

III. Affichage

Pour pouvoir afficher les données proprement dans un tableaux, nous avons utilisé la librairie public **libfort** disponible sur github : https://github.com/seleznevae/libfort. Voici un exemple du tableau des résultats 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 utilisé dans notre programme sont des entiers. Afin de rendre les choses plus lisible, nous les avons converties vers un format temporel.

Nous avons également un deuxième tableau affichant le meilleur temps de chaque secteur ainsi que la voiture qui l’a fait.

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

Voir le code en annexe dans le fichier **display.c** pour l’implémentation de ces tableaux.

IV. Le tri du classement

Avant de trier on fait une copie des données de la mémoire partagée par le biais de 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 faut évidemment allouer suffisamment de mémoire pour le bloc de destination afin

que celui-ci puisse contenir toutes les données.

Pour pouvoir classer les voitures en fonction de leur tour le plus rapide, ou en fonction de leur position par rapport aux autres, 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 *));
```

Voici une petite explication des arguments de cette fonction :

Argument n° :

1. un pointeur vers le début de la zone mémoire à trier
2. le nombre d'éléments à trier
3. la taille des éléments stockés dans le tableau
4. un pointeur vers la fonction permettant 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.
5. les paramètres suivant sont expliqué ci-dessous

Les const de qsort

Les deux paramètres de type `(const void *)` font appel à l'utilisation de pointeurs `(void *)` qui sont 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.

Voir le code en annexe dans le fichier **display.c** pour l'implémentation de ces fonctions.

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 voitures 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.

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 cette fois ci, ce sont les 10 premiers lignes du fichier Q2 qui sont lues afin de déterminer les participants. Finalement le fichier Q3 est lu avant le début de la course du dimanche.

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

Le fichier lastQ1 contient les 5 dernières voitures du Q1, lastQ2 contient les 5 dernières voitures du Q2. On utilise le fichier Q3 généré à la fin de Q3 afin de déterminer les 10 premières 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**.

Voir le code en annexe dans le fichier **files.c** pour l'implémentation de ces fonctions.

1.4.7 Sécurité du programme

Pour éviter de rendre le code trop lourd on a décidé de ne pas obliger l'utilisateur à lancer toutes les séances d'essais. On peut donc passer au P2 sans avoir exécuté P1. Par contre pour les séances de qualifications on a ajouté de la sécurité.

Nous avons fait en sorte qu'une qualification ne puisse pas être lancée si la précédente n'a pas été lancée et terminée auparavant. Q2 ne sera donc jamais exécuté avant Q1 tout comme Q3 ne sera jamais exécuté avant Q2 ou Q1. Il en va de même pour la course, celle-ci ne peut être lancée si les 3 qualifications ont été terminées.

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 les arguments attendu.

Voir le code en annexe dans le fichier **main.c** pour l'implémentation de la sécurité

1.4.8 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. Comme la majorité d'entre nous ne suit pas la formule 1, nous avons dû nous renseigner a ce 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 au tour le plus rapide
- La ou les voiture(s) au stands
- La ou les voiture(s) ayant abandonné la course (OUT).

1.5.2 Affichage cliquable : TODO !

Comme pour le tableaux généré par `htop` dans Linux, la possibilité de cliquer sur un des en-têtes de colonne afin de trier automatiquement l'affichage du tableau en fonction de cette colonne pourrait s'avérer intéressante.

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.

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

2 Conclusion

2.1 Daniel Olivier

Avis de Daniel sur le projet

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 Morgan Valentin

Avis de Morgan sur le projet

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.

2.3 Martin Michotte

Avis de Martin sur le projet

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.

2.4 Martin Perdaens

Avis de Martin sur le projet

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

Références utilisé pour ce porjet

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

```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", "S3", "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 get_resources_file récupérer les fichiers dans le dossier src/resources un fichier
 *
*@param char file_name nom du fichier qui se trouve dans le dossier resources.
 */

char *get_resources_file(char *file_name) {

    /*** static array in the function, to avoid lose of the array when the function ends ***/
    static char resource_files[20];
    strcpy(resource_files, "src/resources/");
    strcat(resource_files, file_name);
    return resource_files;

}

void save_ranking() {

    char rsrc_file[20];
    strcpy(rsrc_file, get_resources_file(circuit.step_name));

    FILE *file = fopen(rsrc_file, "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
```

```
45 *   les élimninés dans les tableaux passé en paramétre également.
46 
47 *@param int qualified_cars[] un tableau pour stocker les voiutures
      qualifiés
48 *@param int race_ranking[] un tableau pour stocker les voiutures en
      fonction du classement de dimanche
49 *@param int last_cars_of_Q1[] un tableau pour stocker les élimninés de
      Q1
50 *@param int last_cars_of_Q2[] un tableau pour stocker les élimninés de
      Q2
51 *@param char file_to_read[] le fichier à lire.
52 *@param int lines_to_read le nombre de lignes à lire dans ce fichier
      passé en paramétre.
53 
54 */
55 
56 void
57 read_files(int qualified_cars[], int race_ranking[], int
      last_cars_of_Q1[], int last_cars_of_Q2[], char file_to_read[],
58            int lines_to_read) {
59 
60     int file_size = find_size(file_to_read);
61     char absolute_path[file_size];
62     getcwd(absolute_path, file_size);
63 
64     char rsrc_file[20];
65     strcpy(rsrc_file, get_resources_file(file_to_read));
66 
67     char full_absolute_path[file_size];
68     sprintf(full_absolute_path, "%s/%s", absolute_path, rsrc_file);
69 
70     FILE *cmd;
71     char result[NUMBER_OF_CARS];
72     char grep_file_result[file_size];
73     sprintf(grep_file_result, "egrep -o '^[0-9]{1,2}' '%s'",
          full_absolute_path);
74 
75     cmd = popen(grep_file_result, "r");
76     if (cmd == NULL) perror("popen failed !"), exit(EXIT_FAILURE);
77 
78     int i = 0, j = 0, k = 0;
79     while (fgets(result, sizeof(result), cmd)) {
80 
81         if (i < lines_to_read) {
82             qualified_cars[i] = atoi(result);
83             if (strcmp(file_to_read, "Q3") == 0) {
84                 race_ranking[i] = atoi(result);
85             }
86             i++;
```

```
87          } else {
88              if (strcmp(file_to_read, "Q1") == 0) {
89                  last_cars_of_Q1[j] = atoi(result);
90                  j++;
91              } else if (strcmp(file_to_read, "Q2") == 0) {
92                  last_cars_of_Q2[k] = atoi(result);
93                  k++;
94              }
95          }
96      }
97
98      if (pclose(cmd) != 0) perror("pclose failed !"), exit(EXIT_FAILURE)
            ;
99  }
100
101 /** la fonction find_size calcule la taille du fichier passé en paramé
        tre
102
103 *@param char *file_name le fichier à lire.
104 *@return int size qui est la taille du fichier.
105 */
106
107 int find_size(char *file_name) {
108
109     char rsrc_file[20];
110     strcpy(rsrc_file, get_resources_file(file_name));
111
112     FILE *file = fopen(rsrc_file, "r");
113
114     if (file == NULL) {
115         printf("%s '%s' %s", "File previous to", circuit.step_name, "
            NOT found !\n"), exit(EXIT_FAILURE);
116     }
117
118     fseek(file, 0L, SEEK_END);
119
120     int size = ftell(file);
121     if (fclose(file) != 0) perror("fclose failed !"), exit(EXIT_FAILURE
            );
122     return size;
123 }
124
125 /** la fonction save_eliminated_cars sauvegarde les voitures élimninés
        dans un fichier.
126
127 *@param char file_to_save[] le fichier qui va contenir les voitures é
        liminés.
128 *@param int array[] le tableau qui contient les voitures éliminés.
129 */
```

```
130
131 void save_eliminated_cars(char file_to_save[], int array[]) {
132
133     char rsrc_file[20];
134     strcpy(rsrc_file, get_resources_file(file_to_save));
135
136     FILE *file = fopen(rsrc_file, "w");
137
138     if (file == NULL)
139         perror("fopen failed !"), exit(EXIT_FAILURE);
140
141     for (int i = 0; i < 5; i++) {
142         fprintf(file, "%d\n", array[i]);
143     }
144
145     if (fclose(file) != 0)
146         perror("fclose failed !"), exit(EXIT_FAILURE);
147 }
148
149 /** la fonction read_eliminated_cars lit les voitures élimninés depuis
       un fichier
150  *  vers un tableau qui va contenir le classement de la course de
       dimanche.
151
152 *@param char file_to_read[] le fichier à lire qui contient les éliminés
153 *@param int array[] le tableau qui contient le classmeent de la course
       de dimanche.
154 */
155
156 void read_eliminated_cars(char file_to_read[], int array[]) {
157
158     char results[5];
159
160     char rsrc_file[20];
161     strcpy(rsrc_file, get_resources_file(file_to_read));
162
163     FILE *file = fopen(rsrc_file, "r");
164
165     if (file == NULL) perror("fopen failed !"), exit(EXIT_FAILURE);
166
167     int i = 15, j = 10;
168     while (fgets(results, sizeof(results), file)) {
169
170         if (strcmp(file_to_read, "lastQ1") == 0) {
171             array[i] = atoi(results);
172             i++;
173         }
174
175         if (strcmp(file_to_read, "lastQ2") == 0) {
```

```
176             array[j] = atoi(results);
177             j++;
178         }
179     }
180 
181     if (fclose(file) != 0)
182         perror("fclose failed !"), exit(EXIT_FAILURE);
183 }
184 
185 
186 /** la fonction read_resources_files permet de lire les fichiers dans
       le dossier src/resources
187  *
188 *@param char file_name nom du fichier qui se trouve dans le dossier
       resources.
189  */
190 
191 void read_resources_files(char *file_name) {
192 
193     int file_size = find_size(file_name);
194 
195     FILE *file;
196 
197     char result[file_size];
198 
199     char rsrc_file[20];
200     strcpy(rsrc_file, get_resources_file(file_name));
201 
202     file = fopen(rsrc_file, "r");
203 
204     if (file == NULL)
205         perror("fopen failed !"), exit(EXIT_FAILURE);
206 
207     while (fgets(result, sizeof(result), file)) {
208 
209         printf("%s", result);
210     }
211 }
```

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[]);
21
22 void read_resources_files(char *file_name);
23
24 char* get_resources_file(char *file_name);
```

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     read_resources_files("usage");
29     exit(EXIT_FAILURE);
30 }
31
32 /********  Manuel du programme  *********/
33
34 void help() {
35     read_resources_files("help");
36     exit(EXIT_SUCCESS);
37 }
38
39 /********  Version du programme  *********/
40
41 void version() {
42     read_resources_files("version");
43     exit(EXIT_SUCCESS);
44 }
45
46 int main(int argc, char **argv) {
47
48     signal(SIGINT, return_cursor);
49
50     //valeurs par défaut
51     circuit.lap_km = 7;
52     circuit.race_km = 305;
53
54     /***********************************************
55     *             Paramétrage du programme         *
56     ***********************************************/
57
58     int user_km = 0;
59     char day_name[5], step_name[5];
60
61     static struct option long_options[] = {{"day",
          required_argument, NULL, 'd'},
62                                            {"step",
                                               required_argument, NULL,
                                               's'},
63                                            {"length",
                                               required_argument, NULL,
                                               'l'},
64                                            {"version", no_argument, 0,
                                                        'v'},
65                                            {"help",    no_argument, 0,
                                                        'h'},
66                                            {NULL, 0,
```

```
                                                                    NULL
                                                , 0}};
67
68      char opt;
69      while ((opt = getopt_long(argc, argv, "vhd:s:l:", long_options,
           NULL)) != EOF) {
70          switch (opt) {
71              case 'h':
72                  help();
73                  break;
74              case 'v':
75                  version();
76                  break;
77              case 'd':
78                  strcpy(day_name, optarg);
79                  break;
80              case 's':
81                  strcpy(step_name, optarg);
82                  break;
83              case 'l':
84                  user_km = atoi(optarg);
85                  break;
86              default:
87                  print_usage();
88          }
89      }
90
91      /******** Friday *********/
92      if (!strcmp(day_name, "fri")) {
93
94          if (!strcmp(step_name, "P1")) {
95
96              /********  Assignation du nbr de voitures, du nom de l'é
                   tape et le temps de l'étape *********/
97              circuit = (Circuit) {.number_of_cars = 20, .step_name = "P1
                   ", .step_total_time = minutes_to_ms(90)};
98
99          } else if (!strcmp(step_name, "P2")) {
100
101             circuit = (Circuit) {.number_of_cars = 20, .step_name = "P2
                   ", .step_total_time = minutes_to_ms(90)};
102
103         } else {
104             print_usage();
105         }
106
107         /******** Saturday *********/
108     } else if (!strcmp(day_name, "sat")) {
109
```

```
110         if (!strcmp(step_name, "P3")) {
111
112             circuit = (Circuit) {.number_of_cars = 20, .step_name = "P3
                    ", .step_total_time = minutes_to_ms(60)};
113
114         } else if (!strcmp(step_name, "Q1")) {
115
116             circuit = (Circuit) {.number_of_cars = 20, .step_name = "Q1
                    ", .step_total_time = minutes_to_ms(18)};
117
118         } else if (!strcmp(step_name, "Q2")) {
119
120             circuit = (Circuit) {.number_of_cars = 15, .step_name = "Q2
                    ", .step_total_time = minutes_to_ms(15)};
121
122             /********  Lecture des 15 premiers voitures au Q1  ********
                    */
123             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q1", 15);
124
125         } else if (!strcmp(step_name, "Q3")) {
126
127             circuit = (Circuit) {.number_of_cars = 10, .step_name = "Q3
                    ", .step_total_time = minutes_to_ms(12)};
128
129             /********  Lecture des 10 premiers voitures au Q2  ********
                    */
130             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q2", 10);
131
132         } else {
133             print_usage();
134         }
135         /******** Sunday *********/
136     } else if (!strcmp(day_name, "sun")) {
137
138         if (!strcmp(step_name, "RACE")) {
139
140             /********  Assignation du nbr de voitures, du nom de l'é
                    tape et le temps de l'étape *********/
141             circuit.number_of_cars = 20;
142             circuit.step_name = "RACE";
143             circuit.step_total_time = minutes_to_ms(120);
144
145             /********  Lecture des 10 premiers voitures au Q3  ********
                    */
146             read_files(qualified_cars, race_ranking, last_cars_of_Q1,
                    last_cars_of_Q2, "Q3", 10);
147
```

```
148             /********  Lecture du fichier lastQ2 et attribution de la
                   10iéme à la 15iéme place  *********/
149             read_eliminated_cars("lastQ2", race_ranking);
150
151             /********  Lecture du fichier lastQ1 et attribution de la
                   15iéme à la 20iéme place   *********/
152             read_eliminated_cars("lastQ1", race_ranking);
153
154             /********  Lecture du fichier Q3 et attribution de la 1ére
                   à la 10iéme place   *********/
155             read_eliminated_cars("Q3", race_ranking);
156
157             /********  La longueur du circuit est 7km *********/
158             if (user_km == 0) {
159                 circuit.number_of_laps = circuit.race_km / circuit.
                       lap_km;
160
161                 /********  La longueur du circuit a été changé par l'
                       utilisateur *********/
162             } else if (user_km > 0) {
163                 circuit.number_of_laps = circuit.race_km / user_km;
164             } else {
165                 print_usage();
166             }
167         } else {
168             print_usage();
169         }
170     } else {
171         print_usage();
172     }
173
174     /***************************************************
175     *              Sauvegarde des fichiers             *
176     ***************************************************/
177
178     /******* Si on est au Q2, les éliminés du Q1 sont sauvegardés dans
           le fichier lastQ1 ******/
179     !strcmp(circuit.step_name, "Q2") ?
180     save_eliminated_cars("lastQ1", last_cars_of_Q1) :
181
182     /******* Si on est au Q3, les éliminés du Q2 sont sauvegardés dans
           le fichier lastQ2 ******/
183     !strcmp(circuit.step_name, "Q3") ?
184     save_eliminated_cars("lastQ2", last_cars_of_Q2) :
185     NULL;
186
187     /***************************************************
188     *           Création de la mémoire partagée        *
189     ***************************************************/
```

```
190
191     int struct_shm_id = shmget(IPC_PRIVATE, sizeof(F1_Car) * circuit.
            number_of_cars, 0600 | IPC_CREAT);
192     if (struct_shm_id == -1) {
193         perror("shmget failed !");
194         exit(EXIT_FAILURE);
195     }
196
197     car = shmat(struct_shm_id, NULL, 0);
198     if (car == (void *) (-1)) {
199         perror("shmat failed !");
200         exit(EXIT_FAILURE);
201     }
202
203     /*******************************************************
204     *                 Création des sémaphores              *
205     *******************************************************/
206
207     int sem_shm_id = shmget(IPC_PRIVATE, sizeof(sem_t), 0600 |
            IPC_CREAT);
208     if (sem_shm_id == -1) {
209         perror("shmget failed !");
210         exit(EXIT_FAILURE);
211     }
212     sem_t *sem = shmat(sem_shm_id, NULL, 0);
213     if (sem == (void *) (-1)) {
214         perror("shmat failed !");
215         exit(EXIT_FAILURE);
216     }
217
218     sem_init(sem, 1, 1);
219
220     /*********************************************************
221      *                 Création des fils/voitures             *
222      *********************************************************/
223
224     int i;
225     pid_t pid = 0;
226     for (i = 0; i < circuit.number_of_cars; i++) {
227         pid = fork();
228         if (pid == 0)
229             break;
230     }
231
232     switch (pid) {
233
234         case -1:
235             /********  échec du fork *********/
236             fprintf(stderr, "fork failed !");
```

```
237             exit(EXIT_FAILURE);
238
239         case 0:
240             /********  Si on est au Q2 ou Q3 attribution des id par le
                   tableau des qualifiés  *********/
241             (!strcmp(circuit.step_name, "Q2") || !strcmp(circuit.
                   step_name, "Q3")) ?
242             child(sem, &car[i], &qualified_cars[i]) :
243
244             /********  Si on est au RACE attribution des id par le
                   tableau race_ranking  *********/
245             !strcmp(circuit.step_name, "RACE") ?
246             child(sem, &car[i], &race_ranking[i]) :
247
248             /********  Si on est aux autres étapes attribution des id
                   par le tableau car_names  *********/
249             child(sem, &car[i], &car_names[i]);
250
251             exit(EXIT_SUCCESS);
252
253         default:
254             /********  Appel de la fonction display qui va afficher les
                    données  *********/
255             display(sem, car);
256
257             /********  wait for children to finish  *********/
258             for (int j = 0; j < circuit.number_of_cars; j++) {
259                 wait(NULL);
260             }
261     }
262     /********  Détachament des segments de mémoire  *********/
263     shmdt(car);
264
265     /********  Supprimer la mémoire partagée  *********/
266     shmctl(struct_shm_id, IPC_RMID, NULL);
267
268     /********  Destruction des sémaphores  *********/
269     sem_destroy(sem);
270     shmdt(sem);
271     shmctl(sem_shm_id, IPC_RMID, NULL);
272     exit(EXIT_SUCCESS);
273 }
```

Listing 10 : prng.c

```
1
2 #include "prng.h"
3
4 /********  Création des temps des différentes pour les voitures
```

```
    *********/
void random_seed(unsigned int seed) { srand(seed); }

/********  la probabilité d'aller au stand  *********/
int sector_range(int min, int max, int crashing_probability) {
  car_crashed(crashing_probability);
  return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
}

/********  le temps passé au stand  *********/
int stand_duration(int min, int max) {
  return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
}

/********  la probabilité d'aller au stand  *********/
int stand_probability(int seed) { return rand() % seed == 0; }

/********  runs in a certain probability, like 1/seed  *********/
int car_crashed(unsigned int seed) { return rand() % seed == 0; }
```

Listing 11 : prng.h

```
//
// Created by danny on 4/10/19.
//

#pragma once

#include <stdlib.h>

void random_seed(unsigned int seed);

int sector_range(int min, int max, int crashing_probability);

int stand_duration(int min, int max);

int stand_probability(int seed);

int car_crashed(unsigned int seed);
```

Listing 12 : time.c

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

#include "time.h"

```

```
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

```c

#include "window.h"

void init_window() { printf("\e[?1049h\e[?7l\e[?25l\e[2J\e[1;52r"); }

void clear() { printf("\e[55H\e[9999C\e[1J\e[1;55r"); }

void terminate_window() { printf("\e[?7h\e[?25h\e[2J\e[;r\e[?1049l"); }

void return_cursor() {
  clear();
  terminate_window();
  exit(EXIT_SUCCESS);
}
```

Listing 15 : window.h

```c
#pragma once

#include <stdio.h>
#include <stdlib.h>

void init_window();

void terminate_window();

void clear();

void return_cursor();
```