
Rapport de projet “Formule 1” Groupe 4

OS travaux pratiques

Daniel Olivier, Martin Michotte, Morgan
Valentin, Martin Perdaens



07 dec 2019

Table des matières

Rapport du projet : F1-of-Linux	3
Introduction et présentation du projet	3
Cahier des charges	4
Première partie : gestion des séances d’essai, qualifications et course	4
Explication des particularités du code	6
Fonctionnalités du code	6
Mémoire partagée et communication entre processus	6
Libération des ressources de l’ordinateur	8
Création et gestion des processus	8
Rôle du processus père	8
Difficultés rencontrées et solutions	9
Compréhension du cahier des charges	9
Évolutions futures	10
Intégration de codes couleurs dans l’affichage	10
Affichage cliquable	10
Options lié à la pression d’une touche de clavier	10
Phase d’essai entièrement libre	10
Conclusion	12
Exemplaire du code	13
child.c	13
child.h	14
display.c	15
display.h	19
files.c	20
files.h	22
main.c	23
prng.c	28
prng.h	28
time.c	28
time.h	29
window.c	29
window.h	30

Rapport du projet : F1-of-Linux

Groupe 4

Notre groupe est constitué de 4 personnes :

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

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 gèrera les séances d'essais libres, les qualifications ainsi que la course. De plus, certaines informations doivent être disponibles : 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).

Cahier des charges

Nous voulons organiser un grand prix contenant 20 voitures, à la manière de la Formule 1. 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 Formule 1 est établi de la manière suivante :

- Vendredi matin, une séance d'essais libres d'une durée de 1h30 (P1)
- Vendredi après-midi, une deuxième séance d'essais libres d'une durée de 1h30 (P2)
- Samedi matin, une troisième et dernière séance d'essais libres d'une durée de 1h (P3)
- Samedi après-midi, une séance de qualifications est organisée et divisée en 3 parties :
 - Q1, d'une durée 18 minutes et visant à éliminer les 5 dernières voitures qui occuperont les places 16 à 20 sur la grille de départ de la course en fonction de leur meilleur temps au tour,
 - Q2, d'une durée 15 minutes, qui éliminera les 5 dernières voitures suivantes et qui occuperont les places 11 à 16 sur la grille de départ de la course en fonction de leur meilleur temps au tour,
 - Q3, d'une durée 12 minutes et permettant de classer les 10 voitures restantes pour établir les 10 premières places sur la grille de départ de la course en fonction de leur meilleur temps au tour.
- Dimanche après-midi, la course en elle-même visant à obtenir un podium ainsi qu'un classement général typique des courses de voitures.

Ce projet devra prendre en charge plusieurs particularités qui seront développées dans les points ci-dessous.

Première partie : gestion des séances d'essai, qualifications et course

Lors des séances d'essais (P1, P2, P3) : Il est nécessaire de relever les temps dans les différents secteurs (au nombre de 3) à chaque passage de chacune des voitures.

De plus, dans l'affichage des séances d'essais, il est important de connaître le meilleur temps à chaque secteur ainsi que d'autres informations pertinentes comme si la voiture est au stand (PIT) ou si elle abandonne la séance (OUT). Malgré que les voitures soient au stand où ait abandonné la séance, on conserve toujours le meilleur temps de la voiture ainsi que son classement.

Pendant la séance d'essais, le classement des voitures se fait en fonction de leur tour complet le plus rapide. À la fin des séances d'essais, on conserve le classement final.

Lors des qualifications (Q1, Q2, Q3) : Lors des qualifications, le temps des 3 secteurs à chaque passage pour chaque voiture est à relever.

De plus, dans l'affichage des qualifications, il est important de connaître le meilleur temps à chaque secteur ainsi que d'autres informations pertinentes comme si la voiture est au stand (PIT) ou si elle abandonne la séance (OUT). Malgré que les voitures soient au stand où ait abandonné la séance, on conserve toujours le meilleur temps de la voiture ainsi que son classement.

Le classement des voitures se fait en fonction de leur tour complet le plus rapide.

La particularité avec les qualifications est un temps réduit et l'importance de conserver le classement de chacune des séances afin d'en définir l'ordre de départ de la course.

- À la fin de la première qualification, 15 voitures resteront qualifiées pour la 2ème séance et les 5 dernières sont placées à la fin de la grille de départ (places 16 à 20),
- À la fin de la deuxième qualification, il reste 10 voitures qualifiées pour la 3ème séance et les 5 dernières sont placées dans les places 11 à 15 de la grille de départ,
- Le classement de la troisième qualification attribue les places 1 à 10 de la grille de départ.

Lors de la course : L'affichage de la course présente le classement de l'ordre sur la grille de départ. Le classement doit toujours être maintenu durant la course, même s'il y a des dépassements. Il est important de savoir qui a le meilleur temps dans chacun des secteurs et également qui a le tour le plus rapide.

Comme pour les essais libres et les qualifications, il est nécessaire de relever les temps dans les différents secteurs à chaque passage de chacune des voitures.

- Si une voiture est en abandon de course (out), elle sera classée en fin de classement.
- Si la voiture est aux stands (PIT), le temps au stand est comptabilisé dans son temps et elle ressort à sa place dans la course. Par ailleurs, pour ce point, il y a généralement 2 ou 3 PIT par voiture par course.

À la fin de la course, on conserve le classement final et le tour le plus rapide.

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

De plus, il est demandé de paramétrer le programme. En effet, les circuits peuvent être de longueur variable et 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 interprocessus ;
- Utiliser les sémaphores pour synchroniser l'accès à la mémoire partagée.

Explication des particularités du code

Fonctionnalités du code

Le programme prend en tant qu'arguments le nom d'une étape du week-end de Formule 1 ainsi que la longueur d'un tour en kilomètres. Si ce dernier n'est pas fourni, une valeur par défaut est attribuée.

On lance la phase sélectionnée pour chacune des voitures participantes. Lors de la simulation, les voitures participantes vont générer des temps aléatoires à chaque secteur.

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.

Mémoire partagée et communication entre processus

La mémoire partagée contient un tableau de structure comportant les informations de secteurs entre autres choses.

```
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;
```

- `shmget (cle,taille,flag)` retourne l'identificateur d'un segment à partir de sa clé(cle) ou -1 en cas d'échec. Le segment sera créé s'il n'existait pas encore. On peut utiliser la clé `IPC_PRIVATE` pour la création quand il n'est pas utile ensuite d'acquérir l'identificateur. Le paramètre `taille` donne le nombre d'octets du segment (s'il a déjà été créé, la taille doit être inférieure ou égale à la taille

de création). Le paramètre option est une combinaison (par OU bit à bit) de constantes (telles que IPC_CREAT pour la création) et de droits d'accès (comme 0666). Par exemple pour créer un segment on utilisera typiquement l'option IPC_CREAT|0666, et pour l'acquisition simplement 0666

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4
5 int shmget(key_t key, int size, int shmflg);
```

```
1 Fl_Car *car;
2
3 int struct_shm_id = shmget(IPC_PRIVATE, sizeof(Fl_Car) * circuit.
    number_of_cars, 0600 | IPC_CREAT);
4 if (struct_shm_id == -1) {
5     perror("shmget failed !");
6     exit(EXIT_FAILURE);
7 }
```

- **shmat** (identificateur,NULL,option) sert à attacher un segment, c'est à dire à obtenir une fois que l'on connaît son identificateur, un pointeur vers la zone de mémoire partagée. L'option sera SHM_RDONLY pour un segment en lecture seule ou 0 pour un segment en lecture/écriture. Cette primitive retourne l'adresse de la zone de mémoire partagée ou (void *)(-1) en cas d'échec

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4
5 void *shmat(int shmid, void *shmaddr, int shmflg);
```

```
1 car = shmat(struct_shm_id, NULL, 0);
2 if (car == (void *) (-1)) {
3     perror("shmat failed !");
4     exit(EXIT_FAILURE);
5 }
```

- **shmdt** (adresse) sert à détacher le segment attaché à l'adresse passée en paramètre. Retourne 0 en cas de succès, ou -1 en cas d'échec. Lorsqu'un processus n'utilise plus un segment de mémoire partagée, il peut le détacher de son espace adresses par shmdt. Attention, l'argument de shmdt est l'adresse à laquelle le segment a été attaché, pas le semid du segment ! shmdt ne détruit pas le segment. Pour cela, il faut utiliser shmctl.

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4
```

```
5  int shmdt(void *shmaddr);
```

```
1  shmdt(car);
```

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.

La sémaphore nous permettent de garantir l'accès exclusif à la mémoire partagée. Les opérations `sem_wait(sem_t *sem)` et `sem_post(sem_t *sem)` permettent respectivement de verrouiller et déverrouiller une sémaphore.

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 « destruction » de la sémaphore par le biais de l'opération `sem_destroy(sem_t *sem)`.

Ensuite, on se détache des zones de mémoire partagée et ensuite on les supprime.

```
1  shmdt(data);
2  shmctl(struct_shm_id, IPC_RMID, NULL);
3
4  sem_destroy(sem);
5  shmdt(sem);
6  shmctl(sem_shm_id, IPC_RMID, NULL);
```

Création et gestion des processus

Chaque voiture correspond à un processus fils, tandis que le père s'occupe de la gestion des étapes et de l'affichage.

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

Nos `fork` sont 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 la sauvegarde des informations sur fichier.

Difficultés rencontrées et solutions

Concernant les difficultés rencontrées, suite à la reprise de ce projet pour la seconde session, il n'y a pas eu particulièrement de nouveaux problèmes rencontrés.

Comme indiqué dans le point traitant la compréhension du cahier des charges, grâce à la communication avec la professeur ainsi que du travail réalisé par chacun des membres pour la prise de note, la création d'un flowchart et d'un tableau détaillé reprenant les informations importantes de façon claire et concise, nous avons su éviter la plupart des difficultés possiblement rencontrées.

Compréhension du cahier des charges

Au vu du cahier des charges reçu, nous avons eu des difficultés à comprendre plus concrètement comment mettre en œuvre certaines implémentations demandées tant lors de la première session que lors de la seconde session. À force de recherche et également de questions posées dans le cadre du temps consacré spécifiquement au projet en cours lors de la première session, nous avons accumulé différentes notes nous permettant de mieux visualiser ce qui nous avait posé problème.

Deux rendez-vous ont été convenu avec la professeur (2 avril 2019 à 15h et 25 juin 2019 à 11h) afin de mieux visualiser les demandes, de comprendre ce qui n'avait pas été lors de la première session et également les détails qui n'avaient pas été correctement compris ou nous paraissant tout simplement trop flou.

Suite à ces rendez-vous avec la professeur, le flowchart et le tableau détaillé contenant les informations importantes des demandes dans le cahier des charges fourni, cette difficulté a été résolue.

Évolutions futures

Intégration de codes couleurs dans l’affichage

Il s’agit certes d’une implémentation de moindre importance, mais cela pourrait s’avérer pratique pour ressortir de manière plus rapide les informations les plus importantes. Par exemple, on pourrait réaliser un code couleur pour :

- Les 3 premières places dans le classement,
- Le temps le plus rapide au tour,
- 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).

Affichage cliquable

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.

Options lié à la pression d’une touche de clavier

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
- F10 : Quitter

Phase d’essai entièrement libre

Par souci de facilité (et pour se concentrer sur d’autres parties nécessitant plus de temps et de travail), nous avons décidé que les voitures présentes lors d’une séance d’essai libre démarrent toutes comme s’il s’agissait d’une étape classique (une qualification ou une course).

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.

Conclusion

L'avantage de ce projet est l'application de concepts multiples vue en cours théorique au courant du premier quadrimestre. Cela nous 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.).

Lors de la première session, ce projet nous avait permis d'apprendre à programmer de façon plus assidue. Lors de l'écriture d'une nouvelle méthode, nous testions systématiquement le projet et en cas de problème, nous prenions le temps de relire le code (et si nécessaire, nous testions différentes méthodes pour déboguer et avancer dans le projet). Nous avons rencontré plusieurs difficultés de compréhension par rapport au cahier des charges ainsi que d'autres difficultés rencontrées, nous avons accumulé un retard par rapport au planning que nous avons fixé au départ, mais l'avons rattrapé en courant de quadrimestre. Malheureusement, la programmation présentée ne correspondant et ne remplissant pas toutes les demandes, cela nous a entraînés dans une seconde tentative pour ce projet.

En cette seconde session, nous avons changé de méthodologie et avons porté une importance quasi capitale sur le fait de réaliser les tests de méthodes. Nous avons appris de nos erreurs et avons eu des moments constructifs d'échange avec la professeur afin de réussir au mieux ce projet. Nous avons également découvert l'utilité de l'utilisation de quelques librairies, ainsi que d'une documentation disponible en ligne, nous permettant de mieux comprendre certaines implémentations nécessaires.

Exemplaire du code

child.c

```
1  #include "child.h"
2
3  int time_passed = 0;
4  int current_lap = 0;
5  F1_Car *vehicle;
6  Circuit circuit;
7
8  void car_crash() {
9      if (car_crashed(100000000))
10         vehicle->out = 1;
11     else
12         vehicle->out = 0;
13 }
14
15 int finished_running() {
16     if (!strcmp(circuit.step_name, "RACE")) {
17         return current_lap == circuit.number_of_laps;
18     } else {
19         return time_passed >= circuit.step_total_time;
20     }
21 }
22
23 int msleep(unsigned int tms) {
24     return usleep(tms * 1000);
25 }
26
27 void child(sem_t *sem, F1_Car *car, int *car_names) {
28
29     random_seed(getpid());
30     vehicle = car;
31     vehicle->id = *car_names;
32
33     while (!finished_running()) {
34
35         /*(!strcmp(circuit.step_name, "RACE")) ? sleep(10) : 0;
36
37         sem_wait(sem);
38         vehicle->s1 = sector_range(30, 45, 100000000);
39         if (vehicle->best_s1 == 0 || vehicle->best_s1 > vehicle->s1) {
40             vehicle->best_s1 = vehicle->s1;
41         }
42         car_crash();
43         sem_post(sem);
44
45         sem_wait(sem);
46         vehicle->s2 = sector_range(30, 45, 100000000);
```

```
47     if (vehicle->best_s2 == 0 || vehicle->best_s2 > vehicle->s2) {
48         vehicle->best_s2 = vehicle->s2;
49     }
50     car_crash();
51     sem_post(sem);
52
53     sem_wait(sem);
54     vehicle->s3 = sector_range(30, 45, 10000000);
55
56     int i = 1;
57     vehicle->stand = 0;
58     while (stand_probability(10)) {
59
60         vehicle->s3 += stand_duration(1, 100);
61         i++;
62         vehicle->stand = 1;
63     }
64     if (vehicle->best_s3 == 0 || vehicle->best_s3 > vehicle->s3) {
65         vehicle->best_s3 = vehicle->s3;
66     }
67     car_crash();
68     msleep(80);
69
70     vehicle->lap_time = vehicle->s1 + vehicle->s2 + vehicle->s3;
71     time_passed += vehicle->lap_time;
72
73     if (vehicle->best_lap_time == 0 ||
74         vehicle->best_lap_time > vehicle->lap_time)
75         vehicle->best_lap_time = vehicle->lap_time;
76     vehicle->lap++;
77     current_lap = vehicle->lap;
78     (time_passed >= circuit.step_total_time || current_lap ==
79         circuit.number_of_laps) ? vehicle->done = 1 : 0;
80     sem_post(sem);
81     sleep(1);
82 }
```

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
21 #define NUMBER_OF_CARS 20
22
23 typedef struct Circuit {
24     char *step_name;
25     int step_total_time;
26     int number_of_laps;
27     int lap_km;
28     int number_of_cars;
29     int race_km;
30 } Circuit;
31
32
33 typedef struct F1_Car {
34     int id;
35     double lap_time;
36     double s1;
37     double s2;
38     double s3;
39     int best_s1;
40     int best_s2;
41     int best_s3;
42     int stand;
43     int out;
44     int lap;
45     int best_lap_time;
46     int done;
47 } F1_Car;
48
49 void child(sem_t *sem, F1_Car *car, int *car_names);
50
51 void car_crash();
52
53 int finished_running();
54
55 int msleep(unsigned int tms);
```

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 int compare(const void *left, const void *right) {
13     const F1_Car *process_a = (F1_Car *) left;
14     const F1_Car *process_b = (F1_Car *) right;
15
16     if (strcmp(circuit.step_name, "RACE")) {
17         if (process_a->best_lap_time < process_b->best_lap_time)
18             return -1;
19         else if (process_a->best_lap_time > process_b->best_lap_time)
20             return 1;
21         else
22             return 0;
23     } else {
24         if (process_a->lap < process_b->lap)
25             return 1;
26         else if (process_a->lap > process_b->lap)
27             return -1;
28         else
29             return 0;
30     }
31 }
32
33 void print_table() {
34
35     ft_table_t *table = ft_create_table();
36
37     ft_set_border_style(table, FT_DOUBLE2_STYLE);
38
39     ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_ROW_TYPE,
40                     FT_ROW_HEADER);
41     ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_CELL_TEXT_STYLE,
42                     FT_TSTYLE_BOLD);
43     ft_set_cell_prop(table, 0, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
44                     FT_COLOR_CYAN);
45
46     ft_write_ln(table, "POSITION", "NAME", "S1", "S2", "S3", "OUT", "
47                     PIT", "LAP", "LAP TIME", "BEST LAP TIME");
48
49     for (int i = 0; i < circuit.number_of_cars; i++) {
50         F1_Car current = car_array[i];
51     }
52 }
```



```
48     char sector1_time[10], sector2_time[10], sector3_time[10],
49         lap_time[10], best_lap_time[10];
50     to_string(current.s1, sector1_time);
51     to_string(current.s2, sector2_time);
52     to_string(current.s3, sector3_time);
53     to_string(current.lap_time, lap_time);
54     to_string(current.best_lap_time, best_lap_time);
55
56     ft_printf_ln(table, "%d|%d|%.6s|%.6s|%.6s|%d|%d|%d|%.7s|%.7s",
57         i + 1,
58         current.id, sector1_time, sector2_time,
59         sector3_time, current.out,
60         current.stand, current.lap, lap_time,
61         best_lap_time);
62
63     (current.stand)
64     ? ft_set_cell_prop(table, i + 1, FT_ANY_COLUMN,
65         FT_CPROP_CONT_FG_COLOR, FT_COLOR_DARK_GRAY)
66     : ft_set_cell_prop(table, i + 1, 6, FT_CPROP_CONT_FG_COLOR,
67         FT_COLOR_LIGHT_GRAY);
68 }
69
70 ft_set_cell_prop(table, 1, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
71     FT_COLOR_LIGHT_GREEN);
72 ft_set_cell_prop(table, 2, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
73     FT_COLOR_LIGHT_BLUE);
74 ft_set_cell_prop(table, 3, FT_ANY_COLUMN, FT_CPROP_CONT_FG_COLOR,
75     FT_COLOR_LIGHT_YELLOW);
76
77 ft_table_t *second_table = ft_create_table();
78 ft_write_ln(second_table, "SECTORS", "NAME", "TIME");
79 ft_set_border_style(second_table, FT_DOUBLE2_STYLE);
80
81 ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN, FT_CPROP_ROW_TYPE,
82     FT_ROW_HEADER);
83 ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN,
84     FT_CPROP_CELL_TEXT_STYLE, FT_TSTYLE_BOLD);
85 ft_set_cell_prop(second_table, 0, FT_ANY_COLUMN,
86     FT_CPROP_CONT_FG_COLOR, FT_COLOR_CYAN);
87
88 char s1_time[10], s2_time[10], s3_time[10], winner[10];
89
90 to_string(car_array[best_sector("S1")].best_s1, s1_time);
91 to_string(car_array[best_sector("S2")].best_s2, s2_time);
92 to_string(car_array[best_sector("S3")].best_s3, s3_time);
93 to_string(car_array[best_lap_time()].best_lap_time, winner);
94
95 ft_printf_ln(second_table, "%s|%d|%s", "S1", car_array[best_sector(
96     "S1")].id, s1_time);
```

```
85     ft_printf_ln(second_table, "%s|%d|%s", "S2", car_array[best_sector(  
86         "S2")].id, s2_time);  
87     ft_printf_ln(second_table, "%s|%d|%s", "S3", car_array[best_sector(  
88         "S3")].id, s3_time);  
89     (!strcmp(circuit.step_name, "RACE")) ?  
90     ft_printf_ln(second_table, "%s|%d|%.7s", "Winner", car_array[  
91         best_lap_time()].id, winner) : 0;  
92  
93     clear();  
94     printf("%s", ft_to_string(table));  
95     printf("%s", ft_to_string(second_table));  
96     ft_destroy_table(table);  
97     ft_destroy_table(second_table);  
98 }  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129
```

```
int best_sector(char sector[]) {  
    int sector_number = 0, id = 0;  
    for (int i = 0; i < circuit.number_of_cars; i++) {  
        if (!strcmp(sector, "S1")) {  
            if (sector_number == 0 || car_array[i].best_s1 <  
                sector_number) {  
                sector_number = car_array[i].best_s1;  
                id = i;  
            }  
        } else if (!strcmp(sector, "S2")) {  
            if (sector_number == 0 || car_array[i].best_s2 <  
                sector_number) {  
                sector_number = car_array[i].best_s2;  
                id = i;  
            }  
        } else if (!strcmp(sector, "S3")) {  
            if (sector_number == 0 || car_array[i].best_s3 <  
                sector_number) {  
                sector_number = car_array[i].best_s3;  
                id = i;  
            }  
        }  
    }  
    return id;  
}  
  
int best_lap_time() {  
    int winner = 0, id = 0;  
    for (int i = 0; i < circuit.number_of_cars; i++) {  
        if (winner == 0 || car_array[i].best_lap_time < winner) {  
            winner = car_array[i].best_lap_time;  
            id = i;  
        }  
    }  
    return id;  
}
```

```
130 }
131
132 int finished() {
133     for (int i = 0; i < circuit.number_of_cars; ++i) {
134         if (car_array[i].out) {
135             return 1;
136         }
137     }
138     return 0;
139 }
140
141 void display(sem_t *sem, F1_Car *data) {
142     init_window();
143
144     while (1) {
145         sem_wait(sem);
146         memcpy(car_array, data, sizeof(F1_Car) * circuit.number_of_cars);
147     };
148     sem_post(sem);
149     qsort(car_array, circuit.number_of_cars, sizeof(F1_Car),
150         compare);
151     if (finished() || car_array[9].done) {
152         break;
153     }
154     print_table();
155     sleep(1);
156 }
157 sleep(1);
158 save_ranking();
159 terminate_window();
160 }
```

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_lap_time();
```

files.c

```
1 //
2 // Created by danny on 26/10/19.
3 //
4
5 #include "files.h"
6
7
8 Circuit circuit;
9 F1_Car car_array[20];
10
11 void save_ranking() {
12
13     FILE *file = fopen(circuit.step_name, "w");
14
15     if (file == NULL) perror("fopen failed !"), exit(EXIT_FAILURE);
16
17     for (int i = 0; i < circuit.number_of_cars; i++) {
18         char best_lap_str[10];
19         to_string(car_array[i].best_lap_time, best_lap_str);
20         fprintf(file, "%d --> %s\n", car_array[i].id, best_lap_str);
21     }
22
23     if (fclose(file) != 0) perror("fclose failed !"), exit(EXIT_FAILURE);
24 }
25
26 void
27 read_files(int qualified_cars[], int race_ranking[], int
28     last_cars_of_Q1[], int last_cars_of_Q2[], char file_to_read[],
29     int lines_to_read) {
30
31     int path_size = findSize(file_to_read);
32
33     char path_name[path_size];
34     getcwd(path_name, path_size);
35     char file_path_name[path_size];
36     sprintf(file_path_name, "%s/%s", path_name, file_to_read);
37
38     FILE *cmd;
39     char result[NUMBER_OF_CARS];
```

```
39     char command_file_path_name[path_size];
40     sprintf(command_file_path_name, "egrep -o '^[0-9]{1,2}' '%s'",
41             file_path_name);
42
43     cmd = popen(command_file_path_name, "r");
44     if (cmd == NULL) perror("popen failed !"), exit(EXIT_FAILURE);
45
46     int i = 0, j = 0, k = 0;
47     while (fgets(result, sizeof(result), cmd)) {
48         if (i < lines_to_read) {
49             qualified_cars[i] = atoi(result);
50             if (strcmp(file_to_read, "Q3") == 0) {
51                 race_ranking[i] = atoi(result);
52             }
53             i++;
54         } else {
55             if (strcmp(file_to_read, "Q1") == 0) {
56                 last_cars_of_Q1[j] = atoi(result);
57                 j++;
58             } else if (strcmp(file_to_read, "Q2") == 0) {
59                 last_cars_of_Q2[k] = atoi(result);
60                 k++;
61             }
62         }
63     }
64
65     if (pclose(cmd) != 0) perror("pclose failed !"), exit(EXIT_FAILURE);
66 }
67
68 int findSize(char file_name[]) {
69     FILE *file = fopen(file_name, "r");
70
71     if (file == NULL) {
72         printf("%s : %s", circuit.step_name, "file not Found!\n");
73         return -1;
74     }
75
76     fseek(file, 0L, SEEK_END);
77
78     int res = ftell(file);
79     if (fclose(file) != 0) perror("fclose failed !"), exit(EXIT_FAILURE);
80     return res;
81 }
82
83 void save_eliminated_cars(char file_to_save[], int array[]) {
84     FILE *file = fopen(file_to_save, "w");
```

```
87
88     if (file == NULL)
89         perror("fopen failed !"), exit(EXIT_FAILURE);;
90
91     for (int i = 0; i < 5; i++) {
92         fprintf(file, "%d\n", array[i]);
93     }
94
95     if (fclose(file) != 0)
96         perror("fclose failed !"), exit(EXIT_FAILURE);;
97 }
98
99 void read_eliminated_cars(char file_to_read[], int array[]) {
100
101     char results[5];
102
103     FILE *file = fopen(file_to_read, "r");
104
105     if (file == NULL) perror("fopen failed !"), exit(EXIT_FAILURE);;
106
107     int i = 15, j = 10;
108     while (fgets(results, sizeof(results), file)) {
109
110         if (strcmp(file_to_read, "lastQ1") == 0) {
111             array[i] = atoi(results);
112             i++;
113         }
114
115         if (strcmp(file_to_read, "lastQ2") == 0) {
116             array[j] = atoi(results);
117             j++;
118         }
119     }
120
121     if (fclose(file) != 0)
122         perror("fclose failed !"), exit(EXIT_FAILURE);;
123 }
```

files.h

```
1 #pragma once
2
3
4 #include "child.h"
5 #include "window.h"
6 #include "time.h"
7 #include "../lib/fort.h"
8 #include <stdio.h>
9 #include <string.h>
```

```
10 #include <unistd.h>
11 #include <semaphore.h>
12
13
14 void save_ranking();
15
16 int findSize(char file_name[]);
17
18 void read_files(int qualified_cars[], int race_ranking[], int
    last_cars_of_Q1[], int last_cars_of_Q2[], char file[],
19                 int lines_to_read);
20
21 void save_eliminated_cars(char file_to_save[], int array[]);
22
23 void read_eliminated_cars(char file_to_read[], int array[]);
```

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 Fl_Car *car;
12
13
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 int qualified_cars[15], race_ranking[20], last_cars_of_Q1[15],
    last_cars_of_Q2[10];
18
19 void print_usage() {
20     printf("%s", "Usage: ./prog --day [dayName] --step [stepName]\n");
21     printf("%s", "Usage: For race you can specify the lap length, by
        default it's 7km !\n");
22     printf("%s", "Usage: ./prog --day [dayName] --step [stepName] --
        length [number]\n");
23     printf("%s", "Use the --help command for more information. \n");
24     exit(EXIT_FAILURE);
25 }
26
27 void help() {
```

```
28     printf("\n%s\n\n", "These are some commands used to run this
        program.");
29     printf("%s\n", "For P sessions : \t There are run on fridays & P3
        on sat. Use the --day command.");
30     printf("%s\n", "\t\t\t followed by a day name and which step needs
        to be runned for the P sessions.");
31     printf("%s\n\n", "\t\t\t --day fri --step P2 for instance.");
32     printf("%s\n", "For Q sessions : \t There are run on saturdays, use
        the --day command.");
33     printf("%s\n", "\t\t\t followed by a day name and which step needs
        to be runned for the Q sessions.");
34     printf("%s\n\n", "\t\t\t --day sat --step Q3 for instance.");
35     printf("%s\n", "For the RACE session : \t It's run on sundays, use
        the --day command.");
36     printf("%s\n", "\t\t\t followed by a day name. Here you can specify
        the race's lap length.");
37     printf("%s\n", "\t\t\t by default it's 7km, the --length command is
        optional.");
38     printf("%s\n\n", "\t\t\t --day sun --step RACE --length 10 for
        instance. ");
39     exit(EXIT_SUCCESS);
40 }
41
42 int main(int argc, char **argv) {
43
44     signal(SIGINT, return_cursor);
45
46     circuit.lap_km = 7;
47     circuit.race_km = 305;
48
49     int user_km = 0;
50     char day_name[5], step_name[5];
51
52     static struct option long_options[] = {"day",      required_argument
        , NULL, 'd'},
53
60         {"step",      required_argument
        , NULL, 's'},
54         {"length", required_argument
        , NULL, 'l'},
55         {"help",      no_argument, 0,
        'h'},
56         {NULL, 0,
        NULL,
        0}};
57
58     char opt;
59     while ((opt = getopt_long(argc, argv, "hd:s:l:", long_options, NULL
        )) != EOF) {
60         switch (opt) {
61             case 'h':
62                 help();
```



```
63         break;
64     case 'd':
65         strcpy(day_name, optarg);
66         break;
67     case 's':
68         strcpy(step_name, optarg);
69         break;
70     case 'l':
71         user_km = atoi(optarg);
72         break;
73     default:
74         print_usage();
75     }
76 }
77
78 if (!strcmp(day_name, "fri")) {
79     if (!strcmp(step_name, "P1")) {
80         circuit = (Circuit) {.number_of_cars = 20, .step_name = "P1",
81                             .step_total_time = minutes_to_ms(90)};
82     } else if (!strcmp(step_name, "P2")) {
83         circuit = (Circuit) {.number_of_cars = 20, .step_name = "P2",
84                             .step_total_time = minutes_to_ms(90)};
85     } else {
86         print_usage();
87     }
88 } else if (!strcmp(day_name, "sat")) {
89     if (!strcmp(step_name, "P3")) {
90         circuit = (Circuit) {.number_of_cars = 20, .step_name = "P3",
91                             .step_total_time = minutes_to_ms(60)};
92     } else if (!strcmp(step_name, "Q1")) {
93         circuit = (Circuit) {.number_of_cars = 20, .step_name = "Q1",
94                             .step_total_time = minutes_to_ms(18)};
95     } else if (!strcmp(step_name, "Q2")) {
96         circuit = (Circuit) {.number_of_cars = 15, .step_name = "Q2",
97                             .step_total_time = minutes_to_ms(15)};
98         read_files(qualified_cars, race_ranking, last_cars_of_Q1,
99                   last_cars_of_Q2, "Q1", 15);
100     } else if (!strcmp(step_name, "Q3")) {
101         circuit = (Circuit) {.number_of_cars = 10, .step_name = "Q3",
102                             .step_total_time = minutes_to_ms(12)};
103         read_files(qualified_cars, race_ranking, last_cars_of_Q1,
104                   last_cars_of_Q2, "Q2", 10);
105     } else {
106         print_usage();
107     }
108 } else if (!strcmp(day_name, "sun")) {
109     if (!strcmp(step_name, "RACE")) {
110         circuit.number_of_cars = 20;
111         circuit.step_name = "RACE";
112         circuit.step_total_time = minutes_to_ms(120);
113     }
```

```

105         read_files(qualified_cars, race_ranking, last_cars_of_Q1,
106                     last_cars_of_Q2, "Q3", 10);
107         read_eliminated_cars("lastQ2", race_ranking);
108         read_eliminated_cars("lastQ1", race_ranking);
109         read_eliminated_cars("Q3", race_ranking);
110
111         if (user_km == 0) {
112             circuit.number_of_laps = circuit.race_km / circuit.
113                 lap_km;
114         } else if (user_km > 0) {
115             circuit.number_of_laps = circuit.race_km / user_km;
116         } else {
117             print_usage();
118         }
119     } else {
120         print_usage();
121     }
122 }
123
124 !strcmp(circuit.step_name, "Q2") ? save_eliminated_cars("lastQ1",
125                                                         last_cars_of_Q1)
126                                                         : !strcmp(circuit.step_name, "Q3")
127                                                         ? save_eliminated_cars("lastQ2",
128                                                         last_cars_of_Q2)
129                                                         : NULL;
130
131 int struct_shm_id = shmget(IPC_PRIVATE, sizeof(F1_Car) * circuit.
132                             number_of_cars, 0600 | IPC_CREAT);
133 if (struct_shm_id == -1) {
134     perror("shmget failed !");
135     exit(EXIT_FAILURE);
136 }
137
138 car = shmat(struct_shm_id, NULL, 0);
139 if (car == (void *) (-1)) {
140     perror("shmat failed !");
141     exit(EXIT_FAILURE);
142 }
143
144 int sem_shm_id = shmget(IPC_PRIVATE, sizeof(sem_t), 0600 |
145                         IPC_CREAT);
146 if (sem_shm_id == -1) {

```

```

143     perror("shmget failed !");
144     exit(EXIT_FAILURE);
145 }
146 sem_t *sem = shmat(sem_shm_id, NULL, 0);
147 if (sem == (void *) (-1)) {
148     perror("shmat failed !");
149     exit(EXIT_FAILURE);
150 }
151
152 sem_init(sem, 1, 1);
153
154 int i;
155 pid_t pid = 0;
156 for (i = 0; i < circuit.number_of_cars; i++) {
157     pid = fork();
158     if (pid == 0)
159         break;
160 }
161
162 switch (pid) {
163     case -1:
164         fprintf(stderr, "fork failed !");
165         exit(EXIT_FAILURE);
166     case 0:
167         (!strcmp(circuit.step_name, "Q2") || !strcmp(circuit.
168             step_name, "Q3")) ?
169             child(sem, &car[i], &qualified_cars[i]) : !strcmp(circuit.
170                 step_name, "RACE") ?
171                 child(sem, &car[i], &
172                     race_ranking[i]) :
173                     child(sem, &car[i], &car_names[i]);
174
175         exit(0);
176     default:
177         display(sem, car);
178         for (int j = 0; j < circuit.number_of_cars; j++) {
179             wait(NULL);
180         }
181
182         shmctl(struct_shm_id, IPC_RMID, NULL);
183
184         sem_destroy(sem);
185         shmdt(sem);
186         shmctl(sem_shm_id, IPC_RMID, NULL);
187         exit(EXIT_SUCCESS);
188 }

```

prng.c

```
1
2 #include "prng.h"
3
4 void random_seed(unsigned int seed) { srand(seed); }
5
6 int sector_range(int min, int max, int crashing_probability) {
7     car_crashed(crashing_probability);
8     return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
9 }
10
11 int stand_duration(int min, int max) {
12     return rand() % (max * 1000 + 1 - min * 1000) + min * 1000;
13 }
14
15 int stand_probability(int seed) { return rand() % seed == 0; }
16
17 // runs in a certain probability, like 1/seed
18 int car_crashed(unsigned int seed) { return rand() % seed == 0; }
```

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);
```

time.c

```
1 //
2 // Created by danny on 19/10/19.
3 //
4
```

```
5  #include "time.h"
6
7  Time time_to_ms(int msec) {
8      Time formatted_time;
9      div_t result;
10
11     result = div(msec, 60000);
12     formatted_time.min = result.quot;
13     msec = result.rem;
14
15     result = div(msec, 1000);
16     formatted_time.sec = result.quot;
17     msec = result.rem;
18
19     formatted_time.msec = msec;
20     return formatted_time;
21 }
22
23 int minutes_to_ms(int minutes) { return minutes * 60000; }
24
25 void to_string(int msec, char *str) {
26     Time time = time_to_ms(msec);
27     (time.min) ? sprintf(str, "%d':%d\"%d", time.min, time.sec, time.
28                     msec)
29                 : sprintf(str, "%d\"%d", time.sec, time.msec);
30 }
```

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);
```

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 }
```

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();
```