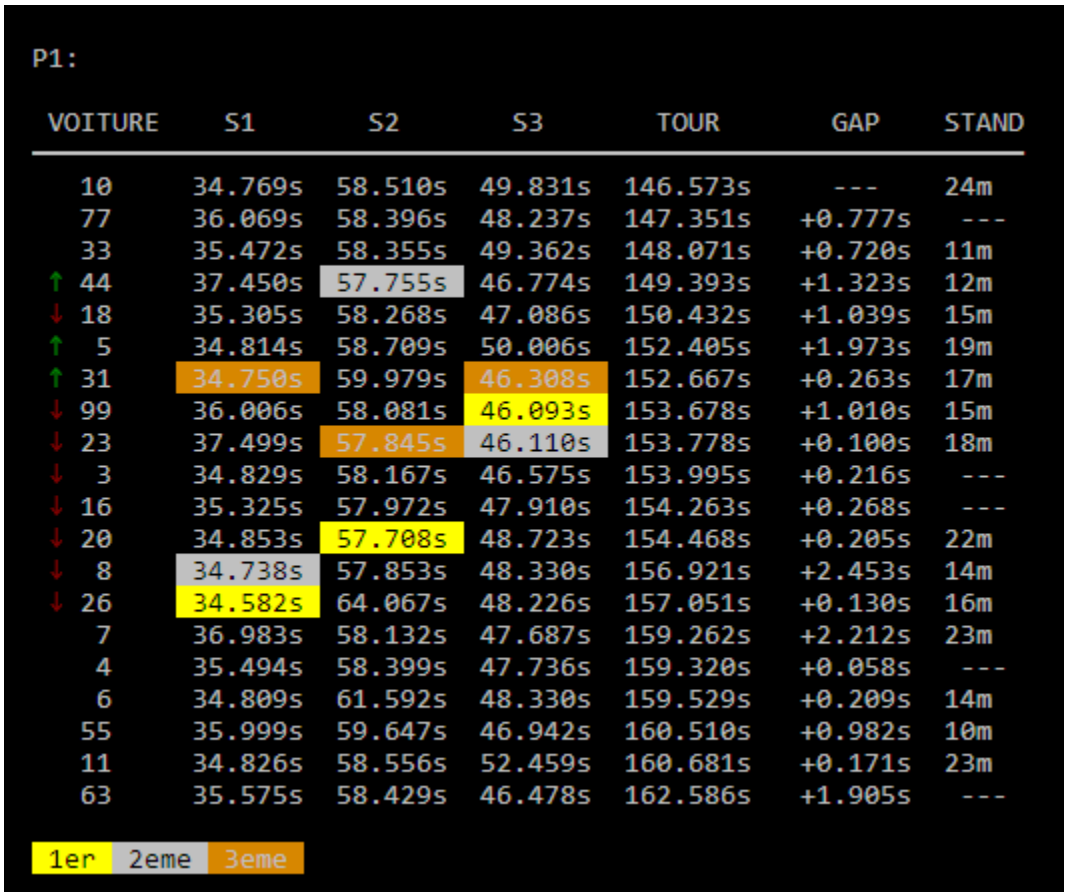


P1:

	VOITURE	S1	S2	S3	TOUR	GAP	STAND
	10	34.769s	58.510s	49.831s	146.573s	---	24m
	77	36.069s	58.396s	48.237s	147.351s	+0.777s	---
	33	35.472s	58.355s	49.362s	148.071s	+0.720s	11m
↑	44	37.450s	57.755s	46.774s	149.393s	+1.323s	12m
↓	18	35.305s	58.268s	47.086s	150.432s	+1.039s	15m
↑	5	34.814s	58.709s	50.006s	152.405s	+1.973s	19m
↑	31	34.750s	59.979s	46.308s	152.667s	+0.263s	17m
↓	99	36.006s	58.081s	46.093s	153.678s	+1.010s	15m
↓	23	37.499s	57.845s	46.110s	153.778s	+0.100s	18m
↓	3	34.829s	58.167s	46.575s	153.995s	+0.216s	---
↓	16	35.325s	57.972s	47.910s	154.263s	+0.268s	---
↓	20	34.853s	57.708s	48.723s	154.468s	+0.205s	22m
↓	8	34.738s	57.853s	48.330s	156.921s	+2.453s	14m
↓	26	34.582s	64.067s	48.226s	157.051s	+0.130s	16m
	7	36.983s	58.132s	47.687s	159.262s	+2.212s	23m
	4	35.494s	58.399s	47.736s	159.320s	+0.058s	---
	6	34.809s	61.592s	48.330s	159.529s	+0.209s	14m
	55	35.999s	59.647s	46.942s	160.510s	+0.982s	10m
	11	34.826s	58.556s	52.459s	160.681s	+0.171s	23m
	63	35.575s	58.429s	46.478s	162.586s	+1.905s	---

1er 2eme 3eme

Projet Francorchamps

SIMULATION DE CIRCUIT RÉDIGÉE EN C

François Girondin (2TL2), Louis De Wilde (2TL1), Carlos Emilliano Ruiz Herrera (2TL1), Mathieu Walravens (2TL1)

Systèmes d'exploitation II (Pratique) | 13-01-21

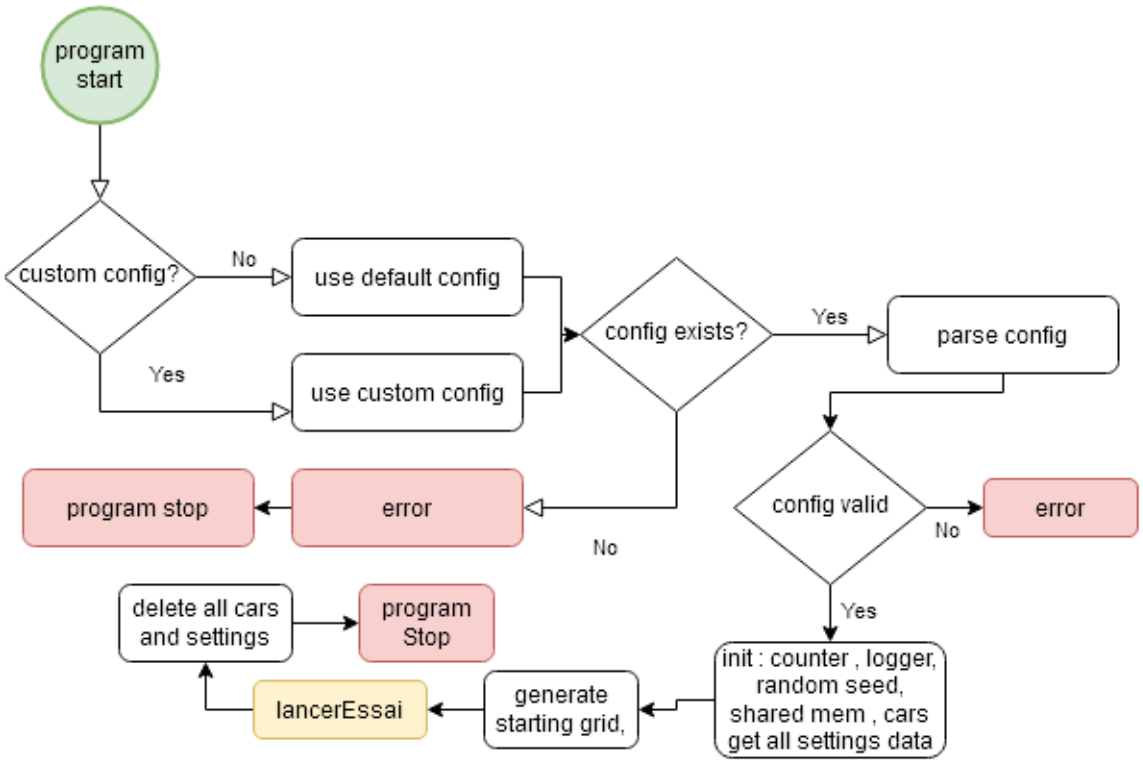
Introduction

Le but de ce projet est de pouvoir simuler, au travers de tableaux de classements rafraîchis à intervalles réguliers, un weekend entier de Grand Prix de Formule 1 sur le circuit de Francorchamps, des séances d'essai à la finale du dimanche.

Pour cela, nous avons écrit en langage C un programme utilisant un processus parent et des processus fils, la mémoire partagée et les sémaphores afin de pouvoir faire tourner toutes les voitures en même temps.

Analyse

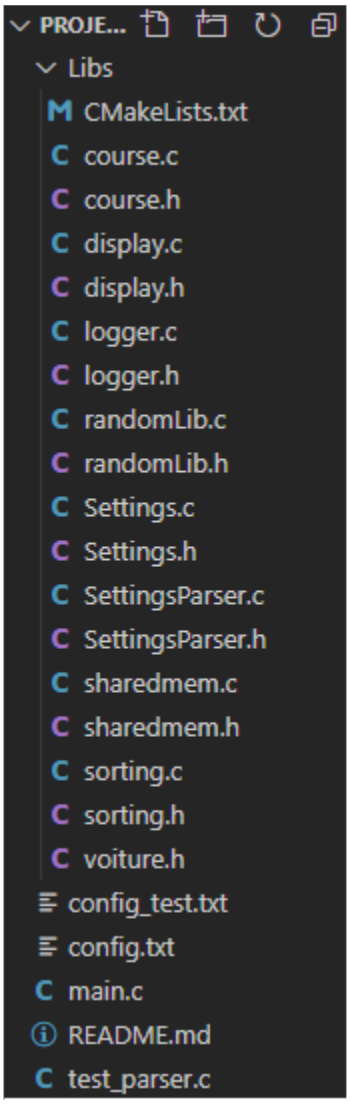
1. Plan de l'application.



Lorsqu'on lance le programme, il passe d'abord par plusieurs étapes de vérification des paramètres. Après cela, il initie ses variables en fonction des paramètres qui lui ont été passés via un fichier de config (passé en paramètre lors du lancement du programme, appelé config.txt par défaut), et crée également la mémoire partagée et son tableau de

sémaphores, ainsi que l'état initial des voitures. Ensuite, on génère la première frame du tableau, avant de lancer les essais, chacun avec leurs propres paramètres, jusqu'à la finale. Enfin, on détruit toutes les voitures et les paramètres avant de clore le programme.

2. Découpage en modules.



Afin de simplifier le code, le programme a été divisé en 10 modules (dont le main.c) et un fichier de paramètres :

-Le module course : Contient les fonctions responsables de la simulation des tours par les voitures : vérification qu'elles se crashent (ou pas), génération des temps de course, usure des pneus, etc...

-Le module display : Concerne tout ce qui concerne l'affichage du déroulement ainsi que des résultats de la course.

-Le module logger : Contient les fonctions permettant de gérer le fichier logs.

-Le module randomLib : Responsable de toute la partie aléatoire du projet par le biais de la création de seeds et de ranges, utilisée pour l'usure des pneus et les temps aléatoires.

-Le module Settings : Contient les fonctions relatives aux paramètres : vérification, traitement et stockage de ces derniers.

-Le module SettingsParser : Contient les fonctions de lecture de fichier permettant d'interpréter le fichier des paramètres et en faire une structure Settings.

-Le module sharedmem : Contient toutes les fonctions liées à la mémoire partagée et aux sémaphores.

-Le module sorting : Contient toutes les fonctions permettant d'effectuer divers tris sur des arrays.

-Le module voiture (.h uniquement) : Contient la définition de la structure des voitures et de leur état (usure et kilométrage).

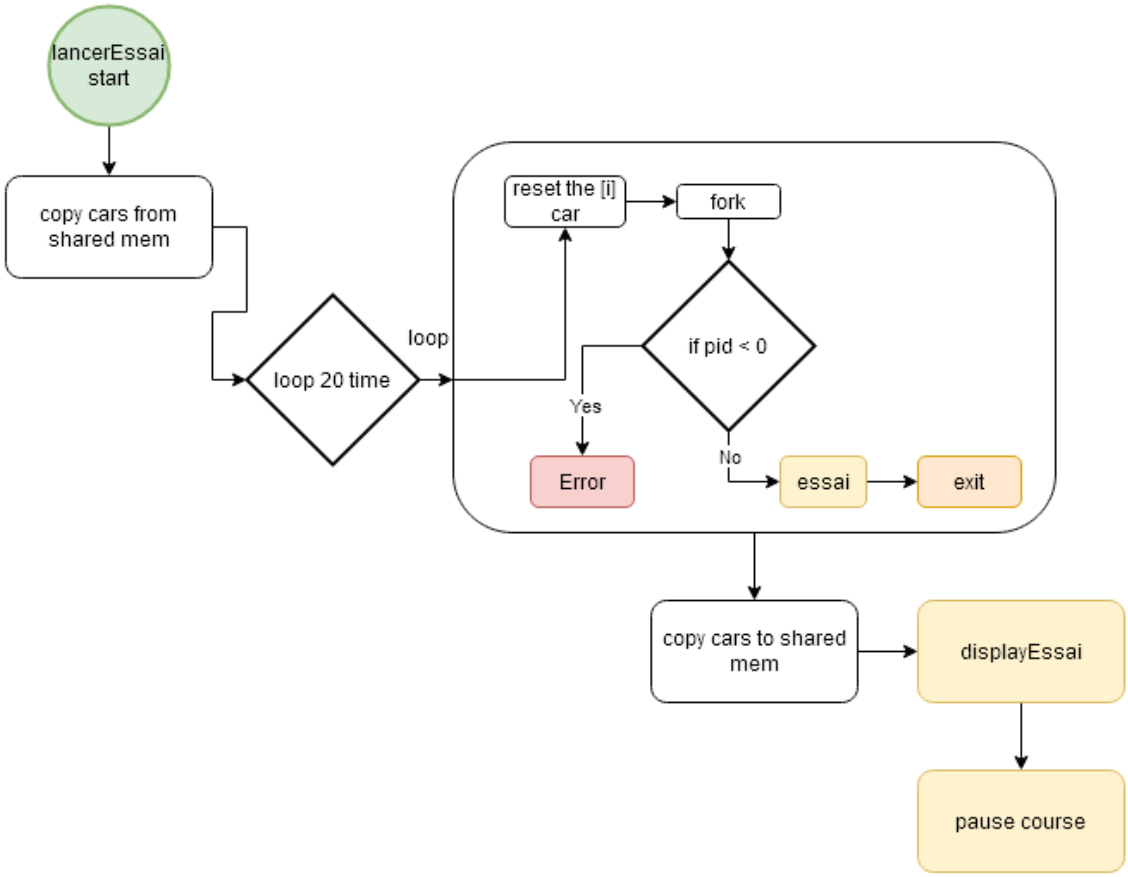
-Le module main : Le programme en tant que tel, qui créée les voitures, lance les courses, les affiche, gère les accès à la mémoire, vérifie et utilise les paramètres, etc...

-Le fichier config.txt : Il contient les paramètres de course tels que les longueurs des sections, le temps de rafraichissement du tableau ou les noms des voitures en course.

3. Plan des modules principaux.

lancerEssai/lancerFinale : les fonctions lancerEssai/lancerFinale ont pour rôle de lancer toutes les voitures dans leur course en créant pour chacune un processus et en lançant la simulation dans ceux-ci, et une fois ceci fait, lancer également la boucle qui gère l'affichage dans le processus parent.

Dans un premier temps, la fonction copie les voitures depuis la mémoire partagée. Ensuite, pour chaque voiture, elle réinitialise ses paramètres avant de créer un processus fils qui exécutera essai pour cette voiture. Puis, les voitures sont recopiées dans la mémoire partagée, pour nous assurer qu'elles sont prêtes avec le début de la simulation et l'affichage, et on lance displayEssai/displayFinale qui boucle chaque seconde afin d'actualiser son affichage pendant que les voitures dans les processus fils continuent leur parcours.



essai/finale : les fonctions essai/finale ont pour rôle de simuler les tours de circuit pour une voiture qui lui est attribuée, en gérant ses temps de course, son usure, ses probabilités de crash, ses pauses au stand, etc…

Dans un premier temps, ces fonctions récupèrent les paramètres du config.txt qui lui seront utiles (taux d'usure, longueur des sections, le délai de sleep…). Ensuite, les fonctions vont copier la voiture depuis la mémoire partagée, pour avoir une copie locale et

travailler sur celle-ci, avant de lancer une boucle while qui tourne jusqu'à ce que la voiture soit à court de temps/de tours.

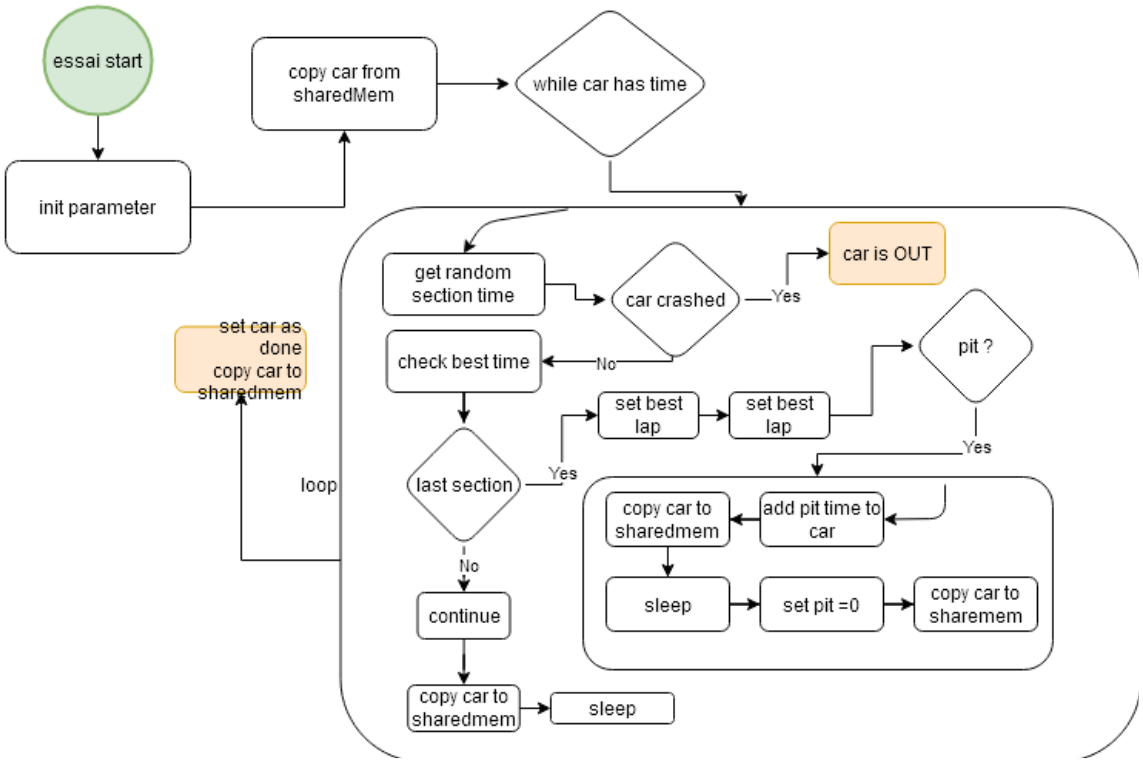
À chaque passage dans la boucle, la voiture calcule un temps de section aléatoire réaliste en nous basant sur des paramètres fournis dans confix.txt. Elle passe ensuite un test pour voir si elle se crashe. Si c'est le cas, la voiture est OUT. Dans le cas contraire, on vérifie si le temps de section est dépasse le meilleur temps actuel de la voiture et on le mets à jour si besoin.

Si ce n'est pas la dernière section, alors on copie la voiture dans la mémoire partagée pour la mettre à jour, et le processus est mis en sleep pendant le temps défini par le paramètre delay du config.txt pour que la course ne se finisse pas trop vite.

Si c'est la dernière, en revanche, on établit le meilleurs temps de la voiture et on calcule aléatoirement si elle doit aller au stand.

Si oui, alors on ajoute le temps de pit au temps de parcours de la voiture, on recopie la voiture dans la mémoire partagée pour que l'affichage en soit au courant, avant de mettre le processus en sleep, puis on met le pit à 0 et on copie à nouveau la voiture dans la mémoire partagée.

Lorsque la voiture a usé tout son temps de course, son statut est mis à DONE et on la copie dans la mémoire partagée.



Sharedmem : Pour gérer la mémoire partagée, on n'a pas utilisé d'alogorithme particulier, vu que, même si le tableau de voitures a 20 écrivains/-ecteurs (fils) et 1 lecteur (affichage), pour chaque voiture individuellement, on n'a que un processus fils et le processus d'affichage qui y ont accès, et donc entourer les zones critiques part des sémaphores -1 et +1 suffit à assure l'exclusion mutuelle et nécessite juste d'un sémaphore par voiture. Un sémaphore supplémentaire a aussi été ajouté pour gérer le fichiers de logs, pour assurer qu'un seul processus n'écrive à la fois.

Pour minimiser les accès à la zone critique qui pourraient ralentir le programme et causer des soucis si ils étaient mal gérés, des fonctions sont définies pour : obtenir une copie locale d'une voiture depuis la mémoire partagée, et mettre à jour la voiture en mémoire partagée avec une voiture locale, donc la plupart du travail se fait en mémoire locale et on n'accède à la mémoire partagée que à des moments spécifiques avec des fonctions utilisant correctement les sémaphores.

Conclusion

Nous sommes parvenus à réaliser l'entièreté du travail demandé, en utilisant mémoire partagée et sémaphores pour faire tourner les différentes voitures. En allant même plus loin que les consignes elles-mêmes, en mettant en place des paramètres extraits d'un fichier texte, en ayant un système de logs en parallèle à l'affichage général et en compliquant certains aspects pour rendre l'application plus réaliste et intéressante.

Les difficultés rencontrées :

Les difficultés rencontrées ont été nombreuses mais pas insurmontables, et parmi les plus notables, il y avait le fait que lors de la finale, les voitures n'étaient pas synchronisées, bien qu'elles tournaient en même temps. Pour que les voitures restent au même stade lors de l'affichage, il a donc fallu rajouter de la logique.

Il y a également eu des problèmes de « fuite de mémoire », mais nous avons pu régler ce problème à l'aide de Valgrind.

D'un ordre moindre, il y avait le fait qu'il a fallu jongler avec les chiffres pour trouver un équilibre dans la probabilité de crash, afin d'éviter une hécatombe à chaque circuit.

Conclusions personnelles :

Carlos Emilliano Ruiz Herrera : Une fois les premières difficultés surmontées, c'était pas vraiment compliqué, ça demandait juste beaucoup du temps.

Mathieu Walravens : Le projet était surtout long, mais assez simple malgré les consignes un peu plus floues, ça rappelle au final le projet de l'année passée.

Louis De Wilde : J'ai trouvé les consignes assez vagues, mais une fois qu'on a compris le cœur du projet, il était assez simple.

François Girondin : Tout comme Louis, je trouve que les consignes manquaient de clarté, mais pas pour le projet en lui-même (car tous les éléments pour le comprendre nous ont été donnés en cours) mais pour le rapport. J'ai trouvé difficile de comprendre ce qui était demandé à certains points sans exemples.