Cours de Systèmes d'exploitation II

Championnat de Formule 1 : Gestion d'un Grand Prix

Rapport de Projet – Projet F1

Julien WILLEMS

Charles ROBERT

Hippolyte AMORY

Quentin HENRARD

Projet

Introduction

Notre travail consiste à faire un programme en C linux qui permet la gestion d'un Championnat de Formule 1, sous forme d'un ensemble de Grand-Prix. Le programme permet la gestion d'un grand-prix allant généralement du vendredi au dimanche et incluant une ou plusieurs phases d'essais, des phases de qualifications et une course finale. Aussi, il doit pouvoir gérer la possibilité d'un weekend spéciale, incluant des modifications dans les horaires.

Un weekend classique se divise en trois séances d'essais libres, la séance de qualifications divisée en trois parties et enfin la course finale. Un weekend spécial lui, n'inclus qu'une séance d'essais libres, des sessions de qualifications identiques au weekend classique mais plus courtes pour une course sprint, une course sprint, une qualification normale et une course normale.

L'élimination aux qualifications signifie une place plus basse dans la grille de départ pour la course finale. Les places 20 à 16 sont occupées par les voitures éliminées au premier tour de qualification. Les voitures éliminées au second tour ont les places 15 à 11. Les Q3, elles, décident des dix premières places sur la grille de départ.

Le programme doit prendre en compte la possibilité d'un crash et doit prendre en compte les passages au PIT, ainsi que le temps que cela fait perdre sur un tour. Il doit aussi suivre la logique du weekend et pouvoir reprendre ou il était après l'interruption entre deux sessions, par exemple reprendre au Q2 après le Q1.

Tous les résultats demandés dans le projet doivent être stockés dans des fichiers adéquats. Les classements et les données relatives à la session dans laquelle le programme se trouve, eux, doivent êtres affichés sous forme de tableaux en console.

Analyse du travail

Structure générale du projet

Le projet est scindé en plusieurs fichiers sources pour mieux séparer les différentes étapes d'un week-end de F1 :

1. **main.c** : Gère la création et l'initialisation d'un nouveau circuit ou la reprise d'un circuit en cours via le fichier etat.
2. **qualif.c** : S'occupe des trois phases de qualifications (Q1, Q2, Q3) pour déterminer les voitures les plus rapides et constituer la grille de départ.
3. **f1_race.c** : Simule la course finale en utilisant la mémoire partagée et les sémaphores pour la synchronisation des données.
4. **Fichiers auxiliaires** :
    - **circuits.csv** : Liste des circuits disponibles avec leurs caractéristiques.

- **etat** : Fichier pivot indiquant le circuit en cours et la phase actuelle.
- **taillecircuit.txt**, **qualifies.txt**, **classement.txt** : Fichiers spécifiques à chaque circuit pour stocker les données nécessaires.

Cette organisation nous a permis de structurer le projet de manière modulaire, facilitant ainsi la gestion des différentes phases du week-end de course.

Gestion des fichiers et des données

Une part significative du projet a été consacrée à la gestion des fichiers :

- **Création et gestion des dossiers de circuits** : Lors de la création d'un nouveau circuit, un dossier dédié est généré contenant les fichiers nécessaires (taillecircuit.txt, qualifies.txt, etc.).
- **Lecture et écriture de fichiers** :
  - **circuits.csv** : Permet de vérifier l'existence d'un circuit et de récupérer ses caractéristiques.
  - **etat** : Maintient l'état actuel du week-end de course pour permettre la reprise.
  - **classement.txt** : Stocke les résultats finaux de la course.

Cette gestion complexe des fichiers a nécessité une attention particulière pour assurer la cohérence des données entre les différentes phases du projet, ce qui a parfois détourné notre attention des mécanismes de synchronisation et d'IPC.

Synchronisation et communication inter-processus

Mémoire partagée et sémaphores (dans f1_race.c)

Nous avons utilisé :

- **Mémoire partagée** : Via shmget et shmat pour stocker les données globales de la course (états des voitures, temps, etc.).
- **Sémaphores** : Via semget, semop et semctl pour protéger l'accès à la mémoire partagée et éviter les conditions de course.

Ces mécanismes sont au cœur de la gestion concurrente des processus représentant les voitures, assurant une synchronisation correcte des données en temps réel.

Communication via des pipes (dans qualif.c)

Pour les qualifications, nous avons utilisé des **pipes** pour communiquer les résultats des processus enfants (voitures) au processus parent, permettant ainsi de collecter et de trier les meilleurs temps pour chaque phase de qualification.

Processus de qualifications (qualif.c)

La gestion des qualifications (Q1, Q2, Q3) est réalisée ainsi :

1. **Q1** (20 voitures) : Création de 20 processus enfants simulant chaque voiture, génération des temps de tours, et élimination des 5 voitures les plus lentes.

2. **Q2** (15 voitures restantes) : Répétition du processus avec les 15 meilleures voitures, élimination de 5 supplémentaires.
3. **Q3** (10 voitures restantes) : Simulation des derniers tours pour déterminer la pole position.

Les résultats des qualifications sont stockés dans des fichiers (qualifies.txt) pour être utilisés lors de la course finale.

Processus de course (f1_race.c)

La course finale est gérée avec :

- **Création de processus enfants** : Un processus par voiture qualifiée.
- **Simulation des tours** : Chaque voiture simule les trois secteurs par tour, avec gestion des événements aléatoires (abandon, passage aux stands).
- **Synchronisation via mémoire partagée et sémaphores** : Mise à jour des temps et des états des voitures en temps réel.
- **Affichage** : Suivi de la course avec affichage des écarts et des statuts des voitures à chaque tour terminé.
- **Classement final** : Collecte et stockage des résultats dans classement.txt.

Programme principal (main.c)

Le fichier **main.c** orchestre la logique globale :

- **Avec argument (./projet <NomCircuit>)** : Création d'un nouveau circuit, vérification dans circuits.csv, création des fichiers nécessaires, et initialisation de l'état à 1 (essais).
- **Sans argument (./projet)** : Lecture du fichier etat pour déterminer la phase actuelle et lancer la phase correspondante (essais, qualifications, course). Après la course, suppression du fichier etat pour signifier la fin du week-end.

Cette automatisation permet de gérer facilement les différentes phases du week-end sans intervention manuelle continue.

Conclusion

Bilan de la réalisation

Nous avons réussi à implémenter les principales fonctionnalités du projet :

- **Phases de qualifications** (Q1, Q2, Q3) avec éliminations progressives.
- **Simulation de la course finale** avec gestion des passages aux stands, des abandons, et affichage des résultats intermédiaires.
- **Utilisation des mécanismes IPC** (sémaphores, mémoire partagée, pipes) pour synchroniser les processus et partager les données.
- **Gestion complète des fichiers** pour stocker et reprendre l'état du week-end de course.

Cependant, malgré ces réussites, nous avons constaté que la gestion des fichiers et la structuration du projet ont pris une part disproportionnée du temps et des ressources, au détriment de l'approfondissement des concepts de synchronisation et de communication inter-processus.

Difficultés rencontrées

1. **Complexité et lourdeur du projet**
   Le projet s'est révélé beaucoup plus lourd que prévu pour un cours de deuxième année. La nécessité de gérer de nombreux fichiers, processus et communications IPC a rendu le projet difficile à maîtriser entièrement dans le temps imparti.

2. **Gestion des fichiers prédominante**
   La gestion extensive des fichiers (création, lecture, écriture, synchronisation avec les processus) a souvent détourné notre attention des mécanismes de synchronisation (sémaphores, mémoire partagée), qui étaient pourtant les points clés à apprendre dans ce cours.

3. **Complexité du langage C**
   La manipulation de la mémoire partagée et des sémaphores en C, combinée à la gestion dynamique des structures de données, a introduit des bugs complexes à résoudre. La rigueur requise par le langage C a parfois freiné notre progression.

4. **Organisation et charge de travail**
   Avec seulement deux membres du groupe partageant les mêmes horaires, la coordination des tâches et la répartition du travail se sont avérées difficiles, surtout en parallèle avec d'autres projets académiques.

Pistes d'amélioration

- **Réduction de la portée du projet**
  Simplifier le projet pour se concentrer davantage sur les mécanismes IPC (sémaphores, mémoire partagée) et moins sur la gestion des fichiers pourrait permettre une meilleure compréhension et une application plus efficace des concepts enseignés.

- **Modularisation accrue**
  Séparer davantage la logique de gestion des fichiers de la logique de synchronisation et de communication inter-processus aiderait à clarifier les différentes responsabilités du code et à faciliter les modifications futures.

- **Interface plus avancée**
  Intégrer une interface plus intuitive (par exemple, via ncurses) pour suivre la course en temps réel pourrait rendre le projet plus engageant et moins axé sur la simple gestion de fichiers.

- **Documentation et tests**
  Une meilleure documentation du code et la mise en place de tests automatisés auraient pu aider à détecter et résoudre plus rapidement les bugs, réduisant ainsi le temps passé sur le débogage.

- **Répartition des tâches**
  Une meilleure organisation au sein du groupe, avec une répartition claire des

responsabilités et une planification plus efficace des réunions, aurait pu améliorer notre productivité et la qualité du projet final.

Conclusion personnelle de groupe

En définitive, ce projet s'est avéré beaucoup trop lourd pour réellement assimiler et mettre en pratique les concepts de sémaphores et de mémoire partagée. Bien que nous ayons pu toucher à ces mécanismes de synchronisation et de gestion de processus en C, ils sont devenus secondaires face à la complexité imposée par la gestion extensive des fichiers et la structuration générale du projet. L'ampleur du projet a souvent obscurci les objectifs pédagogiques principaux, rendant difficile l'apprentissage ciblé des concepts essentiels du cours. Pour de futurs projets, une meilleure délimitation des objectifs et une simplification des tâches pourraient permettre aux étudiants de se concentrer plus efficacement sur les notions clés, facilitant ainsi une compréhension et une application plus approfondies des mécanismes de synchronisation et de communication inter-processus.