Rapport d'Implémentation : Simulateur d'Ordonnancement FIFO

1 Introduction et Objectifs

Ce document décrit l'architecture et le fonctionnement d'un simulateur de gestion de processus (CPU Scheduler) développé en langage C.

L'objectif principal de cette version du projet est de mettre en place l'infrastructure de chargement des données et d'implémenter l'algorithme d'ordonnancement **First-In-First-Out (FIFO)**. Le simulateur lit une configuration de processus depuis un fichier externe et modélise leur exécution séquentielle sur un processeur unique.

2 Architecture Logicielle

Le projet est structuré de manière modulaire pour séparer la définition des données, le chargement des fichiers et la logique algorithmique.

2.1 Structure de Données (`scheduler.h`)

Le fichier d'en-tête définit la structure centrale `Scheduler` qui encapsule l'état global de la simulation.

- **Rôle :** Maintenir la liste des processus et l'horloge système.
- **Définition :**

```
typedef struct {
    Process* processes; // Tableau dynamique des processus
    int processCount;   // Nombre total de processus chargés
    int currentTime;    // Temps actuel du processeur
} Scheduler;
```

Cette structure permet de passer l'état complet du système via un pointeur unique aux différentes fonctions d'ordonnancement.

3 Module de Chargement (`scheduler.c`)

Ce module gère la persistance des données et l'allocation mémoire. Il est conçu pour être robuste face aux erreurs de formatage du fichier d'entrée.

3.1 Fonctionnalité : Parsing du Fichier

La fonction `load_processes_from_file` opère en deux phases pour garantir une allocation mémoire optimale :

1. **Phase d'Analyse (Scanning) :**
   — Le fichier est parcouru une première fois pour compter le nombre de processus valides.
   — Le filtre ignore automatiquement les lignes vides et les commentaires (lignes débutant par '#').
2. **Phase d'Allocation et Lecture :**
   — La mémoire est allouée dynamiquement pour la structure `Scheduler` et le tableau de processus.
   — Le fichier est relu (`rewind`) pour extraire les attributs de chaque processus : *Nom, Temps d'arrivée, Temps d'exécution, Priorité.*

4 Logique Algorithmique : FIFO (`fifo.c`)

Ce fichier contient le cœur de la simulation actuelle. L'algorithme FIFO (Premier Arrivé, Premier Servi) est implémenté selon une approche non-préemptive.

4.1 Pré-traitement : Tri Chronologique

Pour respecter la contrainte FIFO, les processus doivent être traités strictement dans l'ordre de leur arrivée. Le module effectue un tri (Tri à bulles) sur le tableau des processus avant le lancement de la simulation :

$$\text{Condition de tri :} \quad T_{\text{arrivée}}(P_i) \leq T_{\text{arrivée}}(P_j) \quad \forall i < j$$

4.2 Cycle d'Exécution

La fonction `fifo_scheduler` simule le temps CPU. Pour chaque processus P dans la liste triée :

1. **Attente active :** Si le processus arrive dans le futur par rapport à l'horloge actuelle (T_{actuel}), le processeur avance le temps :

$$\text{Si } T_{\text{actuel}} < T_{\text{arrivée}}(P) \implies T_{\text{actuel}} = T_{\text{arrivée}}(P)$$

2. **Exécution :** Le processus s'exécute sans interruption pour toute sa durée (T_{burst}).

$$T_{\text{fin}} = T_{\text{actuel}} + T_{\text{burst}}$$

3. **Mise à jour :** L'horloge globale est mise à jour et les bornes d'exécution sont affichées à la console.

5 Programme Principal (`main.c`)

Le fichier principal orchestre le flux d'exécution de l'application :

1. **Initialisation :** Validation des arguments de la ligne de commande (chemin du fichier de configuration).
2. **Chargement :** Appel au parseur pour construire la structure `Scheduler` en mémoire.
3. **Audit :** Affichage des processus chargés pour vérification par l'utilisateur.
4. **Simulation :** Lancement de l'ordonnanceur FIFO.
5. **Nettoyage :** Libération explicite de la mémoire (free) pour éviter les fuites de mémoire.