Rapport TP3 - Ordonnancement de Processus

NF16 - Automne 2025
Binôme : Amine Saidna - Herald Nkounkou

Introduction

Dans ce TP, nous avons implémenté deux politiques d'ordonnancement : FCFS (First-Come First-Served) et SJF (Shortest Job First). Pour FCFS on utilise une file FIFO optimisée où les processus sont exécutés dans l'ordre d'arrivée. Pour SJF on utilise une liste triée par durée pour toujours choisir le processus le plus court. Les deux sont non préemptifs, donc un processus une fois commencé va jusqu'au bout.

---

1. Analyse de la complexité

1.1 Fonctions communes

creer_processus : **O(1)** - Juste un malloc et initialisation des champs.

free_processus : **O(1)** - Un seul free.

charger_processus : **O(n)** - Un seul parcours avec realloc. Capacité dynamique qui double quand nécessaire, coût amorti O(1) par insertion.

1.2 Fonctions FIFO (FCFS)

fifo_init : **O(1)** - Alloue la structure t_file.

fifo_clear : **O(n)** - Parcourt n éléments pour les libérer.

fifo_vide : **O(1)** - Test de taille.

fifo_enfiler : **O(1)** - Grâce au pointeur queue, insertion directe à la fin. Dans la version de base c'était O(n).

fifo_defiler : **O(1)** - Retrait en tête.

simuler_fcfs : **O(n + T)** - Grâce à l'optimisation de fifo_enfiler, on passe de O(n×T) à O(n + T).

1.3 Fonctions Liste triée (SJF)

lsorted_init : **O(1)** - Retourne NULL.

lsorted_clear : **O(n)** - Parcours pour libérer.

lsorted_vide : **O(1)** - Test de nullité.

lsorted_inserer_trie : **O(k)** où k = taille de la liste. Pire cas : **O(n)**. On utilise <= pour insérer APRÈS les éléments de même durée.

lsorted_extraire_premier : **O(1)** - Retrait en tête.

simuler_sjf : **O(n² + T)** - n insertions en O(n) chacune.

1.4 Fonctions du Gantt

simuler_fcfs_gantt et **simuler_sjf_gantt** : **O(n + T)** et **O(n² + T)** - Même complexité que les simulations, on stocke les résultats.

afficher_gantt : **O(n² + T)** - Dominé par la simulation SJF.

---

2. Structures et fonctions supplémentaires

2.1 Structure t_file (optimisée)

```
typedef struct {
    t_processus* tete;
    t_processus* queue;
    int taille;
} t_file;
```

Justification : Dans la version de base, enfiler prenait O(n) car il fallait parcourir toute la liste. Avec le pointeur queue, insertion en O(1). Impact majeur : FCFS passe de O(n×T) à O(n+T).

2.2 Structure t_execution

```
typedef struct {
    int pid;
    int debut;
    int fin;
} t_execution;
```

Justification : Pour faire le diagramme de Gantt, il faut enregistrer quand chaque processus commence et finit. Sans cette structure, on aurait dû soit refaire les simulations (inefficace), soit gérer plein de tableaux séparés (compliqué).

2.3 Fonctions pour le Gantt

simuler_fcfs_gantt et **simuler_sjf_gantt** - Font les simulations sans afficher, remplissent un tableau de t_execution et retournent le nombre d'événements.

afficher_gantt - Appelle les deux simulations, détermine le temps max, puis affiche trois lignes : arrivées (^), FCFS, et SJF. Couleurs ANSI pour différencier les processus (PID modulo 6). Légende en fin.

Justification : Le diagramme permet de voir directement la différence entre FCFS et SJF. C'est pédagogique et montre visuellement pourquoi SJF réduit les temps d'attente.

2.4 Modification du menu

Ajout de l'option 4 "Afficher le Gantt" et décalage de "Quitter" en 5. Plus logique pour comparer les algorithmes.

---

3. Choix d'implémentation et difficultés

3.1 Optimisation de fifo_enfiler

Problème : Version de base en O(n). Solution : Structure t_file avec pointeur queue. Résultat : O(1) par insertion, FCFS en O(n+T) au lieu de O(n×T).

3.2 Chargement optimisé avec realloc

Au lieu de deux parcours (compter puis remplir), un seul avec realloc. Capacité double quand dépassée. Coût amorti O(1) par insertion, plus efficace en I/O.

3.3 Gestion du temps

Quand la file/liste est vide mais qu'il reste des processus, on saute au prochain temps d'arrivée : temps_actuel = tableau[idx].arrivee. Évite les boucles vides.

3.4 Insertion triée avec égalité

En cas d'égalité de durée, on insère APRÈS. Condition : while (current->suivant!=NULL && current->suivant->duree<=p->duree). Le <= fait qu'on avance jusqu'après tous les éléments égaux.

3.5 Gestion mémoire

Attention aux fuites. Processus libérés dès la fin, appels à fifo_clear/lsorted_clear, libération du tableau dans le main. Testé avec valgrind : aucune fuite.

---

4. Tests et résultats

Exemple 1 (3 processus) : FCFS fait P1→P2→P3. SJF fait P1→P3→P2 car P3 (durée 1) est plus court que P2 (durée 2). Temps total : 8 ticks.

Exemple 2 (5 processus) : SJF optimise en priorisant les courts. Traces conformes au sujet.

Le Gantt montre visuellement que FCFS est simple mais pas optimal, SJF réduit les temps d'attente.

Conclusion

Ce TP nous a permis de maîtriser les listes chaînées et de comprendre FCFS vs SJF. L'optimisation majeure : structure t_file avec pointeur queue (FCFS de O(n×T) à O(n+T)). Utilisation de realloc pour le chargement. Principales difficultés : gestion des pointeurs, cas d'égalité, synchronisation du temps. Code fonctionnel, optimisé, sans fuites, conforme aux specs. Le Gantt apporte une vraie valeur pédagogique.