CY Paris Université

RAPPORT

pour le projet Système d'Exploitation
Licence d'Informatique troisième année

sur le sujet

Simulateur d'ordonnancement de processus

rédigé par

Da Cruz Mathis - Mian Farooq

Avril 2022

Table des matières

Table des figures

Liste des tableaux

Remerciements

Les auteurs du projet tenaient à remercier sincèrement M.Laroque, qui en tant que professeur du module Systèmes d'Exploitation, s'est toujours montré à l'écoute et disponible pour notre groupe tout au long de ce semestre.En effet grâce à sa dévotion en tant que superviseur, il nous a fournis beaucoup d'aide durant les cours de TD. Nous remercions également toute l'équipe des professeurs de l'Université de Cergy pour nous avoir suivie et soutenue jusqu'ici.

1 Introduction

Dans cette section, nous allons présenter brièvement l'objectif du projet.

1.1 Objectif du projet

Le but du projet étant l'implantation d'un logiciel de simulation d'ordonnancement de processus dans un système d'exploitation. Pour ce faire, il faut utiliser au minimum les algorithmes FIFO, SJF, et round-robin. Nous devons donc créer une entrée de simulation (cycles E/S et CPU) ainsi qu'une sortie textuelle ou graphique.

1.2 Principe du projet

Un ordinateur possède plusieurs processus. Ainsi dans les systèmes d'exploitation, l'ordonnaceur est le composant du noyau de ce dernier choissant l'ordre d'exécution des processus. Par conséquent, l'algorithme d'ordonnancement permet l'identification du processus qui permettera la meilleure performance du système.

1.3 Objectif du rapport

Ce rapport peut se définir comme le guide du développeur. En effet, il précise l'algorithme utilisé ainsi que les structures de données. Ce document pourra donc aider un programmeur à améliorer notre projet. /

2 Spécification du projet

Nous avons présenté l'objectif du projet dans la section 1. Dans cette section, nous présentons la spécification de notre système réalisé. Ceci correspond principalement au document de spécification du projet (cahier des charges).

2.1 Notions de base et contraintes du projet

2.1.1 Fonctionnement général du logiciel

L'utilisateur envoie des informations de type données au simulateur. Ce dernier va les traiter afin d'en extraire des résultats. Ils seront affichés via l'écran de l'utilisateur. Par conséquent, il pourra donc les analyser. Ce cycle se répétera autant de fois que nécessaire.

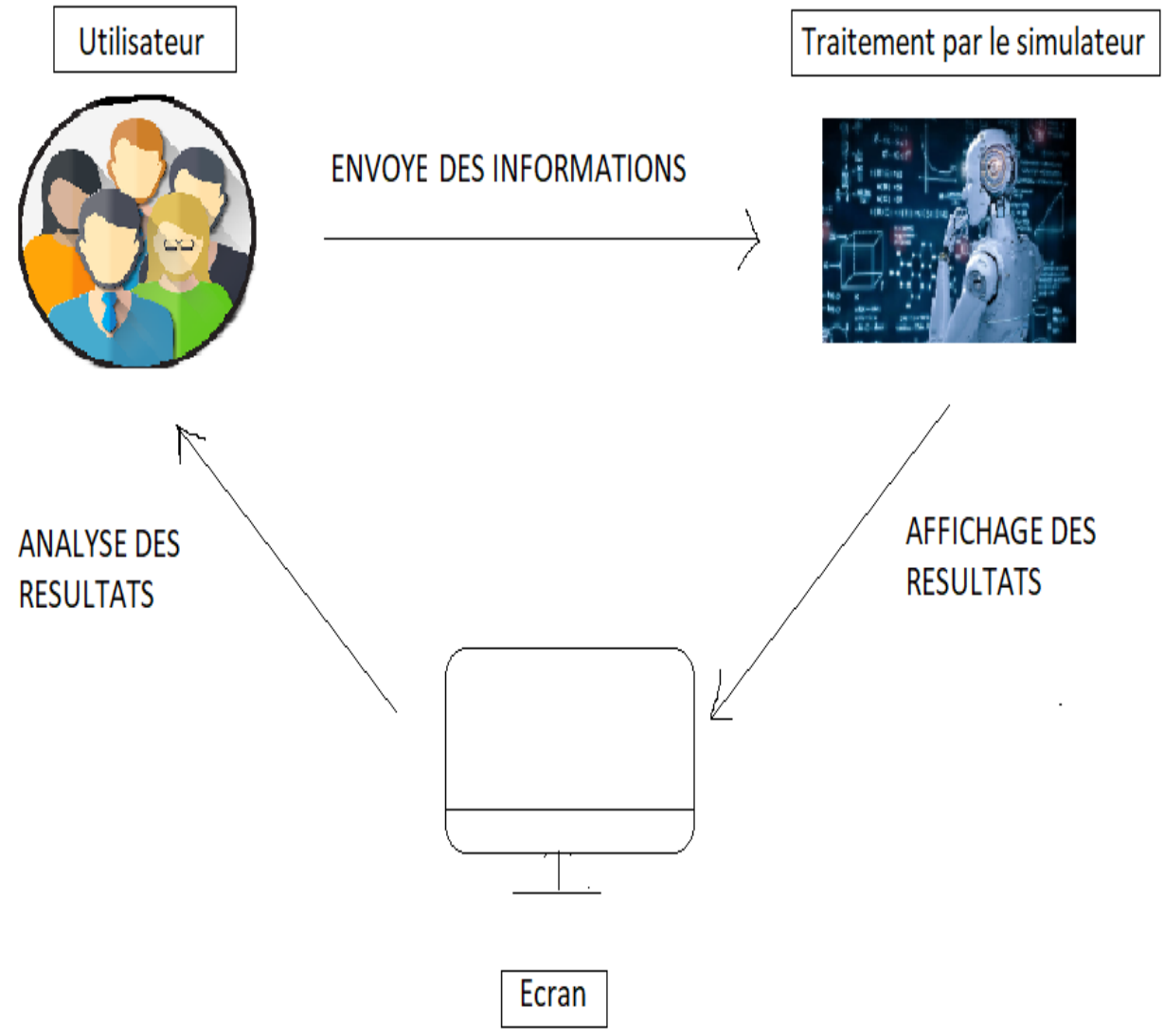

FIGURE 1 – Idée Générale

/

2.1.2 Notions et termonologies de base

Processus : Programme en cours d'exécution par un ordinateur. C'est donc un ensemble d'instructions à exécuter.

Algorithme FIFO (First In First Out) : Chaque processus d'exécute jusqu'à son terme par ordre d'arrivée. Le premier arrivé est donc le processus qui est en tête.

Algorithme SJF (Shortest Job First) : Correspond à l'ordonnancement par ordre inverse du temps d'exécution, c'est-à-dire le processus le plus court s'exécute en premier.

Algorithme Round Robin : Le processus est déplacé vers le processus suivant après un temps (quantium). Le processus qui est interrompu est ajouté à la fin de la file d'attente.

2.1.3 Contraintes et limitations connues

Contraintes :

Gestions des processus : Algorithmes d'ordonnancement, entrée/sortie

Durée : Temps limité de réalisation qui correspondait à 3 mois.

Outils de développement :

1. Language C pour le système
2. Doxygen pour générer la documentation grâce aux commentaires
3. Latex pour le rapport

3 Conception et réalisation du projet

3.1 Structure d'un ordonnancement

Le système d'ordonnancement se compose de trois grandes parties :

1. La table des processus
2. La file d'attente des processus prêts
3. La file d'attente des processus bloqués

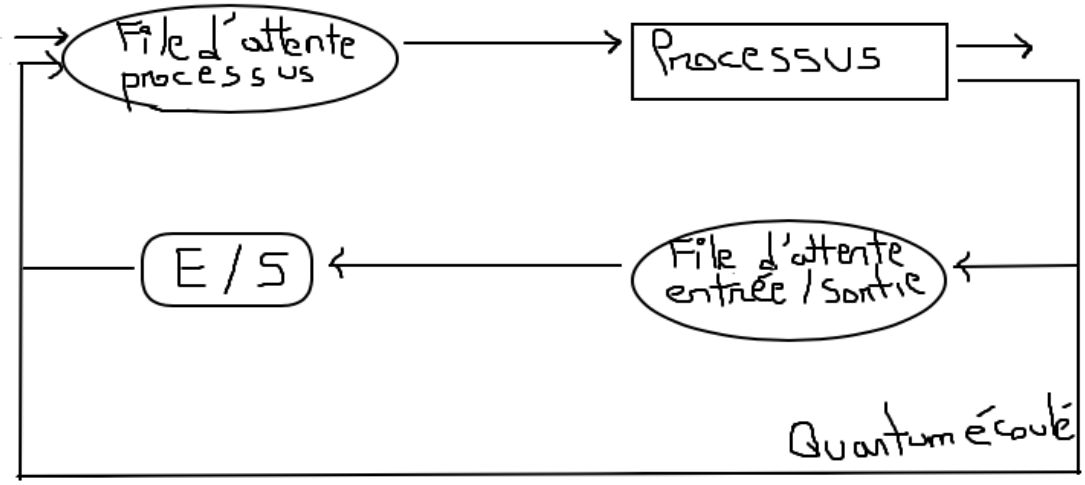

FIGURE 2 – Structure d'un ordonnancement

3.2 Structure de données

Nous avons choisi d'utiliser une liste chaînée. Cette dernière peut contenir plusieurs membres. Elle est codée en langage C.

Nous allons donc détailler la composition de la liste chaînée :

1. no : process
2. at : arrival time (temps d'arrivé)
3. bt : burst time (temmps d'éclatement)
4. tat : turn around time (délai d'exécution)
5. wt : waiting time (temps d'attente)
6. rt : response time (temps de réponse)
7. rd : relative delay (retard relatif)
8. *next : next cell (cellule suivante)

La structure de données est présentée ci-dessous :

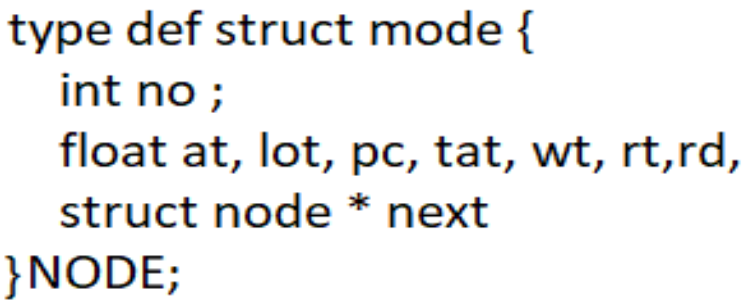

FIGURE 3 – Structure de données utilisée

Chaque algorithme d'ordonnancement est ainsi représenté par une liste chaînée de processus.

3.3 Conception des différents algorithmes

Dans cette partie, nous allons détailler la conception de chaque algorithmes utilisés pour le projet.

Commmencons par l'algorithme FIFO :

1. 1ère étape : Attribuer le nombre de processus à ordonnancer en FIFO, le temps d'éclatement (temps d'exécution), ainsi que son temps d'attente (temps qu'il arrive dans la file d'attente).
2. 2ème étape : Calculer le temps de rotation, le temps d'attente, ect... Pour ce faire, le temps de rotation correspond à la somme des périodes de temps passés à attendre d'entrer dans la mémoire, à exécuter sur le CPU ainsi que l'exécution des entrées/sorties.
3. 3ème étape : Remplir la liste dans le même ordre avec lequel les processus ont été saisis.
4. 4ème étape : Fin de l'algorithme FIFO.

Ensuite, détaillons l'algorithme SJF :

1. 1ère étape : Trier les différents processus en fonction de leur durée.
2. 2ème étape : Sélectionner le processus avec le moins de temps d'arrivée et d'éclatement (mininum de temps).
3. 3ème étape : Sélectionner dans la file d'attente, le processus qui a le moins de temps d'éclatement, après que le premier processus soit exécuter.
4. 4ème étape : Repéter ce cycle jusqu'à ce que les différents processus soit exécuter.

Décrivons l'algorithme Round Robin :

1. 1ère étape : Sélectionner le premier processus et arrêter son exécution (durée du quantum uniquement).
2. 2ème étape : Vérification d'éventuels nouveaux processus dans la file d'attente. Ils seront ajoutés dans la file circulaire, si ils arrivent pendant le temps d'exécution du premiers.
3. 3ème étape : Vérification la disponibilité de la file d'attente après l'exécution du processus pendant le quantum, continuer avec le processus qui précède. Si ce n'est pas le cas, ajouter le processus actuel à la fin de la file d'attente.
4. 4ème étape : Répéter toutes les étapes ci-dessus jusqu'à ce que tous les processus présents dans la file d'attente soient exécutés.

3.4 Insertion d'un nouvel algorithme : SRJF

S'il y a un besoin d'implémenter ce code, il est tout fait possible de le faire. Néanmoins, il faudra effectuer certains changement : le premier est devra se faire au niveau des fichiers on devra y ajouter les fichiers suivants, srjf.h ainsi que srjf.c. Le second serait de modifier le makefile en ajoutant srjf.h et srjf.c dans notre cas cela devrait être ainsi :

makefile

```
COMPILER = gcc

all : exec

exec : main.o node.o utils.o fifo.o sjf.o rr.o
	$(COMPILER) -o exec main.o node.o utils.o fifo.o sjf.o rr.o

main.o : main.c node.h fifo.h sjf.h rr.h
	$(COMPILER) -c -o main.o main.c

fifo.o : fifo.c fifo.h node.h
	$(COMPILER) -c -o fifo.o fifo.c

sjf.o : sjf.c sjf.h node.h
	$(COMPILER) -c -o sjf.o sjf.c

rr.o : rr.c rr.h node.h
	$(COMPILER) -c -o rr.o rr.c

utils.o : utils.c utils.h node.h
	$(COMPILER) -c -o utils.o utils.c

node.o : node.c node.h
	$(COMPILER) -c -o node.o node.c
```

srjf.o

srjf.o

srjf.h

srjf.o : srjf.c srjf.h node.h
$(COMPILER) -c-o srjf.o srjf.c

FIGURE 4 – Algorithme SRJF modifié

Décrivons cet algorithme :

1. 1ère étape : Trouver le processus avec le temps restant minimum à chaque tours.
2. 2ème étape : Réduire son temps de 1.
3. 3ème étape : Vérification du temps (devient 0).
4. 4ème étape : Incrémenter le compteur d'achèvement du processus.
5. 5ème étape : Temps d'achèvement du processus actuel = current-time +1.
6. 6ème étape : Calculer le temps d'attente pour chaque processus terminé : wt[i]= Temps d'achèvement - temps d'arrivée - temps de rafale.
7. 7ème étape : Incrémenter le tour de temps par un.
8. 8ème étape : Calculer le temps d'exécution : Temps dans la file d'attente + temps d'éclatement

Il convient de recoder une fonction "srjf-to-csv" qui permet de créer un fichier csv associé aux données, en lui indiquant le chemin d'accès du fichier (en dur).

3.5 Makefile

Le makefile contient tout les éléments permettant de mener a bien la compilation du programme. Il existe 2 commandes principales :

1. make : Compilation
2. make clean : Nettoyage

3.6 Commentaires et Doxygen

Chaque fichier.c et structures sont commentés ainsi que chaque fonction avec notamment le rôle de la fonction dans le programme, ses paramètres et éventuellement son type de retour. La documentation doxygen générées par ses commentaires est disponible dans les dossiers html et latex

3.7 Répartition des tâches

1. Farooq : FIFO, Structures, Affichages, Rapports...
2. Mathis : Commentaire, Doxygen, SJF, RR, CSV...

4 Conclusion et perspectives

Dans cette section, nous résumons la réalisation du projet et nous présentons également les extensions et améliorations possibles du projet.

4.1 Résumé du travail réalisé

Nous avons réalisé les différents algorithmes permettant la simulation d'un processus d'ordonnancement. Les structures de données sont codés par nos soins, en langage C.

4.2 Améliorations possibles du projet

Nous aurions pu proposer une interface graphique pour l'affichage des résultats, cela aurait permis une meilleure lisibilité pour l'analyse des résultats voir pour la vérification de ses derniers. Malheureusement par manque de temps, cette extention n'a pas été implémenter dans le code. Cependant, voici l'idée du processus de développement de cette extention : on crée une fenêtre graphique à l'aide GtkWidget *window pour la déclaration de la fenêtre ainsi que gtk-init (argc, argv) qui initialise le toolkit et récupère les paramètres entrés sur la ligne de commande. On pourra également rajouter un titre à cette fenêtre mais aussi lui changer de position,etc...

Mais, par ailleurs le plus important ce sont les sigaux et les évènements qui permet de faire un certain nombres de choses telle que la fermeture de la fenêtre par exemple. Autrement dit, défini par ses réactions aux différents événements qui peuvent se produire, c'est-à-dire des changements d'état de variable un déplacement ou un click de souris, une saisie au clavier... La programmation événementielle est une technique d'architecture logicielle où l'application a une boucle principale divisée en deux sections : la première section détecte les événements, la seconde les gère.