# 

# 

# 

# 

# **Scheduler Java - Rapport**

# 

# Conception OS - Projet 2ème semestre

# 3ème année, DLM 2015 - 2016

# 

|  |  |
| --- | --- |
| Date du document : | 09.05.2016 |
| Début du TP : | 07.03.2016 |
| Remise des délivrables : | 13.05.2016 |
| Présentation du projet : | 11.05.2016 |
|  |  |
| Informations |  |
| Etudiants : | Assunçao Jeshon & Di Stasio Leonardo |
| Professeur | Crotinovis Claudio |

# Introduction

Dans le cadre du cours de conception OS nous avons créé un programme qui tourne sur un système Linux et qui permet de représenter un scheduler. Pour se faire, nous avions le choix entre quelques langages et avons opté pour le langage Java.

Le programme représente donc un scheduler en action. Il est possible de créer quatre types de scheduler différents : FCFS, SJF non-préemptif, SJF préemptif, Round Robin. Il est également possible de créer un certain nombre de threads qui possède chacun un temps d'arrivée, un certain nombre de rafales ainsi qu’une couleur.

Ce programme est destiné à des étudiants de première année qui doivent apprendre comment fonctionne le scheduler du CPU. Cette méthode visuelle semble plus interactive et ludique que des simples tableaux.

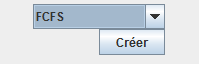
# Objectifs

L’objectif est de créer un programme représentant quatre différents types de scheduler :

* FCFS
* SJF non-préemptif
* SJF préemptif
* Round Robin

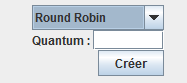
L’utilisateur pourra également créer un certain nombre de threads avec différents couleurs. Cela permettra de différencier tous les threads lors de l’affichage du résultat.

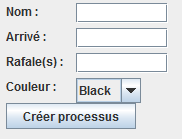
Il fallait également créer une interface intuitive et facile à comprendre pour l’utilisateur qui, rappelons-le, est un débutant dans le domaine.

La partie gauche du programme contiendra toutes les parties de création (scheduler et threads). L’utilisateur pourra également supprimer les threads créés. La partie de droite affichera le résultat selon le type d’ordonnanceur choisi. Les threads seront identifiables via la couleur sélectionnée lors de la création.

# Programme

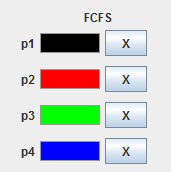
## Création des outils

La partie de gauche de l’application permet la création d’un ordonnanceur parmi les 4 décris ci-dessus. Lorsque l’on choisit le Round Robin un champ texte permet d’insérer un nombre de “Quantum”.

Une fois que l’on a créé un Scheduler on peut passer à la création de processus grâce au formulaire ci-contre.

Nous pouvons alors insérer :

* Un nom pour distinguer le processus dans la liste
* Une arrivé : la case d’arrivée du processus, elle doit être positive
* Un nombre de rafale
* Et une couleur qui le différenciera dans la partie de droite (nous verrons après)

Lorsque les processus sont créés, ils seront affichés dans un panel. De là on pourra les visualiser pour se rappeler des couleurs ou les supprimer s’ils ne nous conviennent plus.

La croix juste à côté permettra de les supprimer au cas où on ne les souhaiterait plus.

# Visualisation

Pour faire la visualisation nous avons mis en place un panel qui permet d’afficher les processus sous forme de rectangle. Nous allons reprendre l’exemple suivant pour l’ordonnanceur FCFS:

|  |  |  |
| --- | --- | --- |
| Processus | Arrivée | Nb cylces |
| P1 | 1 | 25 |
| P2 | 2 | 3 |
| P3 | 8 | 3 |

Ce qui devrait nous donner ce résultat-là :

|  |  |  |
| --- | --- | --- |
| P1 | P2 | P3 |

Et nous allons l’appliquer à notre programme. Voici le résultat que nous avons obtenu (P1 = noir, P2 = rouge et P3 = vert) :



On remarque que le résultat est correct en tout point. Il est également possible de tester les autres. Les couleurs seront les mêmes qu’on a cité avant. Voici les différents tests qu’on a effectués :

## SJF Préemptif

Résultat à obtenir :

|  |  |  |
| --- | --- | --- |
| Processus | Arrivée | Nb cylces |
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P2 | P1 |

Résultat obtenu :



## SJF Non-Préemptif

## Résultat à obtenir :

|  |  |  |
| --- | --- | --- |
| Processus | Arrivée | Nb cylces |
| P1 | 0 | 7 |
| P2 | 2 | 4 |
| P3 | 4 | 1 |

|  |  |  |
| --- | --- | --- |
| P1 | P3 | P2 |

Résultat obtenu :



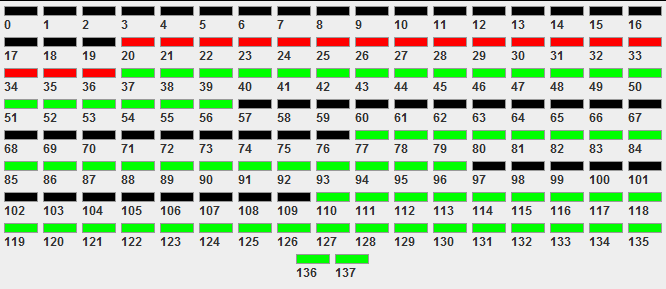
## Round Robin

Résultat à obtenir (Quantum de 20) :

|  |  |  |
| --- | --- | --- |
| Processus | Arrivée | Nb cylces |
| P1 | 0 | 53 |
| P2 | 2 | 17 |
| P3 | 4 | 68 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P1 | P3 | P1 | P3 |

Résultat obtenu :



On remarque que tous les résultats sont corrects dans ces cas-là et que le programme est assez fiable et il est utilisable.

# Bugs connus

* Les valeurs numérique insérer pour l’arrivé ou la rafale ne sont pas contrôlées, on peut ajouter des valeurs négatives. Il faudrait tester ça.

# Conclusion

Notre programme est fonctionnel et répond aux objectifs. Il a été validé grâce aux exemples du cours, cité ci-dessus.  
De plus, comme demandé, le programme se lance automatiquement au démarrage de la machine virtuelle (Ubuntu 14.04).  
Malheureusement, nous n’avons pas pu effectuer des tests plus approfondis, ce qui pourrait faire remonter des erreurs qui nous sont pour l’instant cachés.

Tous les objectifs ont été atteints. Néanmoins, de nombreuses améliorations sont possibles comme :

* Résoudre les différents bugs en effectuant plus de test sur le programme
* Afficher plus de détail sur les processus créés et pouvoir les modifier en cas d’erreurs
* Améliorer l’ergonomie du programme
* Ajouter de nouveaux Scheduler pour élargir les possibilités.
* Créer une animation pour afficher un à un les processus qui rentre en état actif et montrer les états de chaque processus.
* Afficher les calculs comme : le temps d’attente moyen, temps d’attente par processus, etc.

Assunçao Jeshon et Di Stasio Leonardo