UNIVERSITÉ de Cergy-Pontoise

figure RÉSEAU Cursus Master en Ingénierie

Explication des choix : projet ordonnanceur

pour le projet d'OS
Licence d'Informatique troisième année

Simulation d'un ordonnanceur

rédigé par

Loïc Bachelot & Amélie Rémond

Mars 2017

Table des matières

I Comment est organisé le projet ?

Nous avons décidé de réaliser ce projet en Java car il nous paraissait plus facile de réaliser une interface graphique dans ce langage plutôt qu'en langage C. De plus, il est plus facile de parser un document XML en Java. Le dossier du projet est divisé en 5 packages :

- Core : ce package contient notamment les différents algorithmes
- Data : ce package comprend la classe de données Processus
- Factory : ce package comprend la classe de parsing
- IHM : ce package comprend toutes les classes et ressources nécessaire à l'interface graphique
- Log : ce package sert à la mise en place des logs

II Les processus

Nous utilisons le langage Java, mis en œuvre grâce au mécanisme de la programmation orientée objet. De ce fait, nous avons créer une classe Processus permet de créer des objets processus. Nous avons mis les attributs suivants :

- Le nom du processus
- Le temps auquel le processus commence
- La liste des entrées sorties
- La liste des ressources
- L'index auquel on se trouve dans les listes précédentes
- Un ordre permettant d'ordonner les processus lors d'un algo FIFO
- Un temps d'attente
- Un deuxième temps auquel le processus commence pour effectuer une sauvegarde
- Le temps d'exécution total

III Les algorithmes

1) FIFO

L'algorithme FIFO (first in first out) a été implémenté en itératif. L'algorithme comporte une première phase d'initialisation des variables puis une phase d'exécution et enfin une partie calcul des grandeurs. La phase d'exécution est contenu dans une boucle while permettant de vérifier s'il reste des processus à exécuter. Dans cette boucle while, nous n'avons pas utilisé de File mais un système de "current Order". En effet, le current Order est un attribut de chaque processus et il permet de déterminer la place de chacun par rapport aux autres. De ce fait l'algorithme détermine selon l'ordre de chaque processus le suivant à exécuter. Une fois la boucle while terminée, les grandeurs sont calculées et sont mises en variables globales.

2) SJF

L'algorithme SJF (Shortest job first) a été également implémenté en itératif. L'attribut current Order n'est ici pas utilisé pour sélectionner le prochain processus à exécuter. En effet, l'algorithme compare simplement le temps de ressource de tous les processus et prend le plus court. L'exécution du processus ainsi sélectionné est similaire à l'algorithme FIFO.

3) SRJF

L'algorithme SRJF (Shortest remaining job first) a été également implémenté en itératif. Le principe global est le même que pour SJF, cependant comme le processus peut être interrompu, cela veut dire que la liste des temps de ressources peut être modifiée si le processus est interrompu pendant son exécution.

4) RoundRobin

L'algorithme RoundRobin (Tourniquet) a été également implémenté en itératif. L'utilisateur défini un quantum avant de lancer l'algorithme. Ce dernier reprend le principe qu'un FIFO sauf que le processus possède un temps défini (quantum) pour s'exécuter puis il est remis à la fin de la liste.

IV L'IHM

Nous avons décidé de créer l'IHM en utilisant la librairie JavaFX. Nous avons fait ce choix car cette librairie permet d'avoir une rendu esthétique tout en ayant une utilisation simple. Nous avons décider d'utiliser des JTable pour rentrer les valeurs pour avoir un résultat plus organisé.