Ministère de la Transition Numérique et de la Digitalisation

École Supérieure Africaine des Technologies de l'Information et de la Communication

Année académique : 2024-2025

ING 1

PROJET DE CLASSE

IMPLÉMENTATION D'UN ORDONNANCEUR DE PROCESSUS

Réalisé par

COULIBALY Ahmed Namiloho Aziz

KOUADIO Josué Noël Yao

DROH Sigbeu Ulruch

BARY Souleymane

Encadrant académique **: Dr COULIBALY Mamadou**

SOMMAIRE

1. Contexte du projet

Dans le cadre de la mise en pratique des acquis théoriques que nous avons eus durant le cours de « Introduction aux systèmes d'exploitation », nous avons réalisé le projet : implémentation d'un ordonnanceur de processus.

2. Objectifs du projet

Notre projet consiste à implémenter dans le langage Python un ordonnanceur de processus suivant les méthodes d'ordonnancement FCFS (First Come First Served), SJF (Shortest Job First), SRT (Shortest Remaining Time), RR (Round Robin), ainsi que de les comparer.

3. Conception du projet

Afin d'implémenter de manière optimale les méthodes d'ordonnancement, nous avons d'abord écrit les algorithmes correspondants dans le langage de description algorithmique (LDA).

(a) FCFS

FCFS : Premier arrivé, premier servi. Le système est dit ordonnanceur non préemptif. Dans ce cas, aucun processus ne peut interrompre un processus en cours d'exécution puis le processeur traite le premier processus prêt.

L'algorithme est disponible via le lien : [lien de l'algo]

(b) SJF

SJF: Shortest Job First. Nous sommes dans le cas d'un ordonnanceur non préemptif. Dans ce cas, le processeur évalue le temps d'exécution (T_e) des processus prêts puis choisit le processeur ayant le plus court T_e.

(c) SRT

SRT: Shortest Remaining Time. Il s'agit de la méthode d'ordonnancement SJF avec préemption, c'est-à-dire : durant l'exécution d'un processus par le processeur, si un autre processus est prêt, le temps restant du processus en cours d'exécution est comparé au temps d'exécution du processus prêt. Si le temps d'exécution du processus prêt est le plus court, le processeur met en suspension le processus en cours puis traite le processus prêt.

L'algorithme est disponible via le lien : [lien de l'algo]

(d) Round Robin

Round Robin : C'est une méthode utilisée dans un système à ordonnancement préemptif. Nous considérons un tourniquet qui représente une durée de temps pendant laquelle le processeur traite un processus prêt, puis ainsi de suite pour les autres processus prêts.

L'algorithme est disponible via le lien : [lien de l'algo]

4. Réalisation du projet

Nous touchons le cœur du projet. À l'aide de fonctions, procédures et de bibliothèques Python, nous avons implémenté du code permettant de simuler chaque méthode d'ordonnancement. Ensuite, nous évaluons trois critères à savoir le temps de séjour, le temps d'attente et le temps de réponse pour chacune des méthodes d'ordonnancement. Le temps de séjour pour chaque processus est obtenu en soustrayant le temps d'entrée du processus du temps de terminaison. Le temps d'attente est calculé en soustrayant le temps d'exécution du temps de séjour.

Tout ce travail a été réalisé dans l'environnement Jupyter Notebook, dans un *methode_ordonnancement_comparaison.ipynb* où chaque cellule représente un travail bien précis. Ci-dessous, nous expliquons les huit (8) cellules.

Cellule 1 : Nous importons les modules Python nécessaires à la réalisation du projet : *numpy* pour les calculs numériques, *pandas* pour la manipulation des données, et *matplotlib.pyplot* pour leur visualisation.

Cellule 2 : Nous écrivons l'algorithme en langage de description algorithmique (LDA) de la méthode d'ordonnancement FCFS ou PAPS. Nous avons utilisé principalement un tableau de structures, un tri par sélection et des boucles.

Cellule 3 : Nous écrivons l'algorithme en LDA de la méthode d'ordonnancement Round Robin. Nous avons utilisé principalement un tableau de structures, une liste chaînée et des boucles.

Cellule 4 : Nous écrivons l'algorithme en LDA de la méthode d'ordonnancement SJF. Nous avons utilisé principalement un tableau de structures et des boucles.

Cellule 5 : Nous écrivons l'algorithme en LDA de la méthode d'ordonnancement SJF. Nous avons utilisé principalement un tableau de structures et des boucles.

Cellule 6 : Nous implémentons en Python les fonctions correspondant aux quatre méthodes d'ordonnancement (FCFS, Round Robin, SJF non préemptif et SRT préemptif). Chaque fonction prend en entrée une liste de processus (et un quantum pour Round Robin), simule l'exécution selon la politique choisie, puis retourne les résultats de performance (temps de fin, séjour, attente, réponse) ainsi qu'un diagramme de Gantt décrivant l'ordre et la durée d'exécution des processus.

Cellule 7 : Nous définissons les fonctions d'affichage permettant de représenter graphiquement (diagramme de Gantt) et textuellement (tableau avec statistiques) les résultats produits par les méthodes d'ordonnancement.

Cellule 8 : Nous créons une fonction qui permet de saisir au clavier les données des processus (temps d'arrivée et temps d'exécution). Les processus sont enregistrés sous forme de dictionnaires puis regroupés dans une liste, utilisée ensuite pour les simulations.

Cellule 9 : Dans cette cellule, l'utilisateur peut choisir quel algorithme exécuter parmi les quatre disponibles. Pour cela, il suffit de décommenter la ligne correspondant à l'algorithme voulu. Le programme calcule alors les résultats, affiche le diagramme de Gantt et présente les statistiques de performance.

Cellule 10 : Nous exécutons les quatre algorithmes (FCFS, RR, SJF, SRT) sur les mêmes processus, affichons leurs Gantt et tableaux, comparons leurs performances via des graphiques

des temps moyens (attente, réponse, séjour), puis identifions automatiquement l'algorithme le plus performant pour chaque critère

Pour plus de détails sur le contenu de ces cellules, veuillez consulter le document « methode_ordonnancement_comparaison.pdf » ou exécuter le fichier « methode_ordonnancement_comparaison.ipynb»

5. Outils de réalisation du projet

Nous avons utilisé le langage de programmation Python dans l'univers Jupyter Notebook afin d'implémenter la logique algorithmique, de simuler le fonctionnement des méthodes d'ordonnancement, de représenter graphiquement les méthodes d'ordonnancement au travers d'un diagramme de Gantt. Jupyter Notebook est une application web open source qui permet aux utilisateurs de créer et de partager des documents contenant du code en direct, des équations, du texte et des visualisations.

Nous avons aussi utilisé GitHub et Git pour versionner (et organiser) l'ensemble du code produit, puis pour permettre à chacun d'entre nous de pouvoir collaborer sur le même projet.

6. Résultat et Discussion

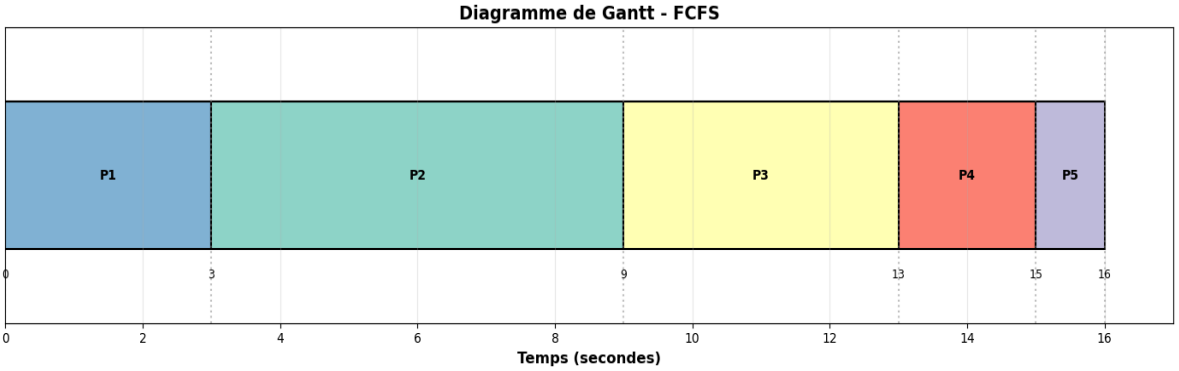
Afin de tester le programme python de simulation des méthodes d'ordonnancement, nous considérons plusieurs tableaux (jeu de données) qui représentent les types possibles de processus à gérer dans un ordonnanceur. Ainsi nous pourrons percevoir l'efficacité de chaque méthode d'ordonnancement.

(a) Jeu de données 1

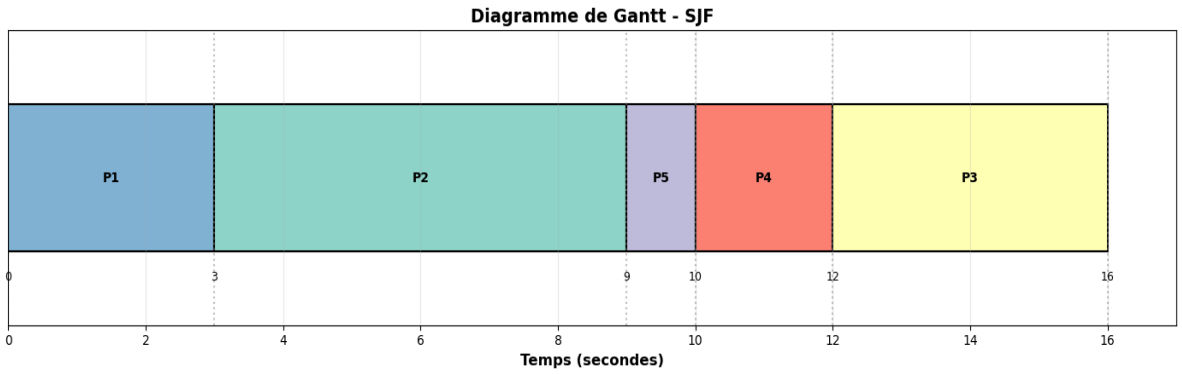
Les processus tels que présentés dans le tableau ci-dessous simulent un vrai OS dans le sens où nous avons un temps d'exécution varié et un ordre d'arrivé réaliste

Processus	Temps d'exécution	Temps d'arrivée
A	3	0
B	6	4
C	4	1
D	2	6
E	1	7

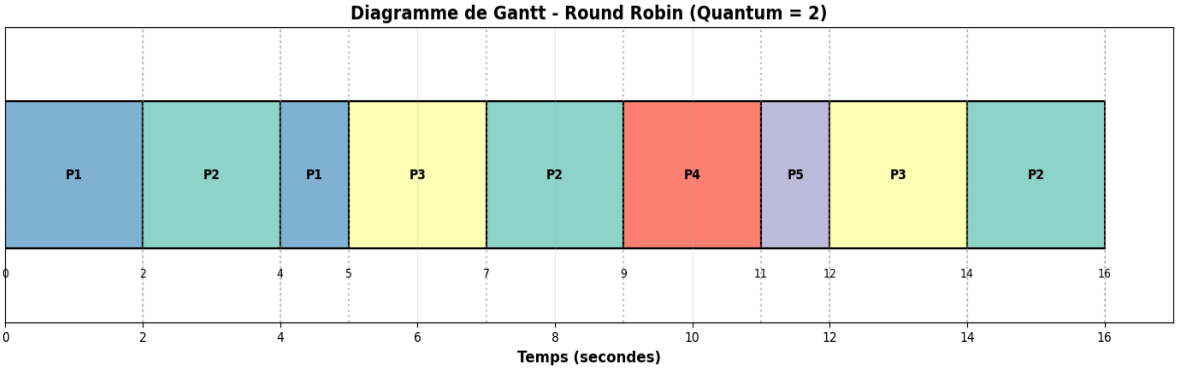
En appliquant notre algorithme correspondant à la méthode PAPS, nous obtenons le diagramme de Gantt ci-dessous :



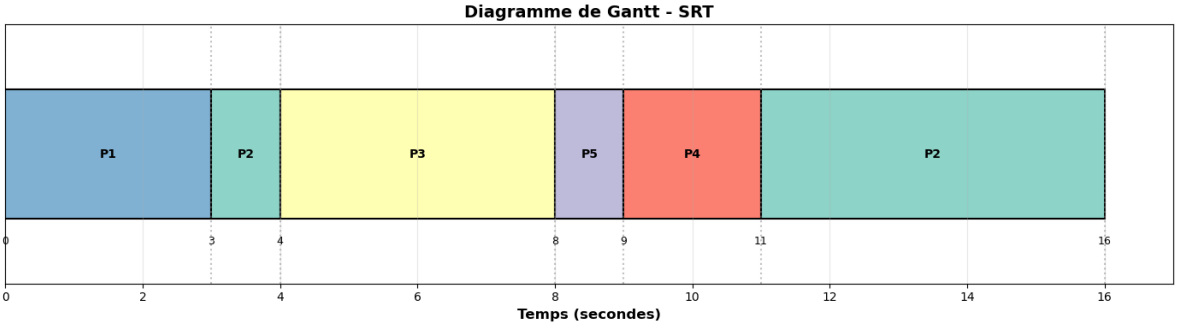
En appliquant notre algorithme correspondant à la méthode SJF, nous obtenons le diagramme de Gantt ci-dessous :



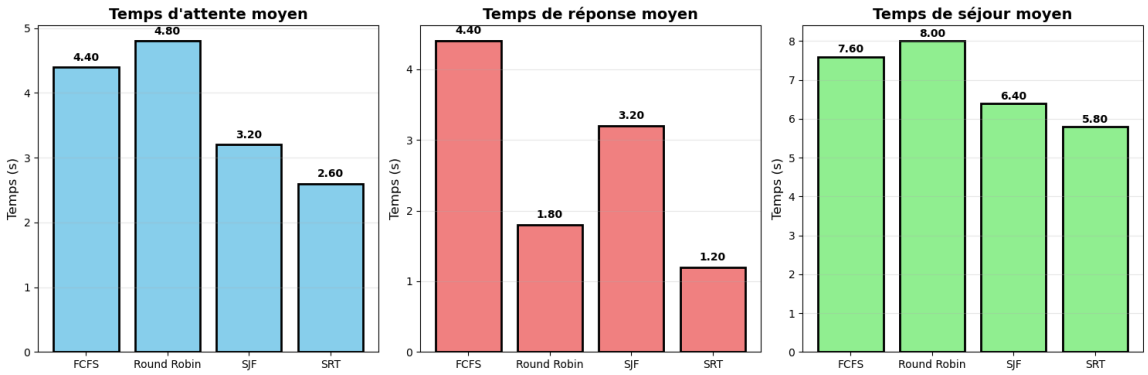
En appliquant notre algorithme correspondant à la méthode RR avec un quantum de 2 unités de temps, nous obtenons le diagramme de Gantt ci-dessous :



En appliquant notre algorithme correspondant à la méthode SRT, nous obtenons le diagramme de Gantt ci-dessous :



Nous comparons maintenant les méthodes d'ordonnancement selon les critères : temps de séjour moyen, temps d'attente moyen, temps de réponse moyen. Nous obtenons le diagramme de Gantt ci-dessous :



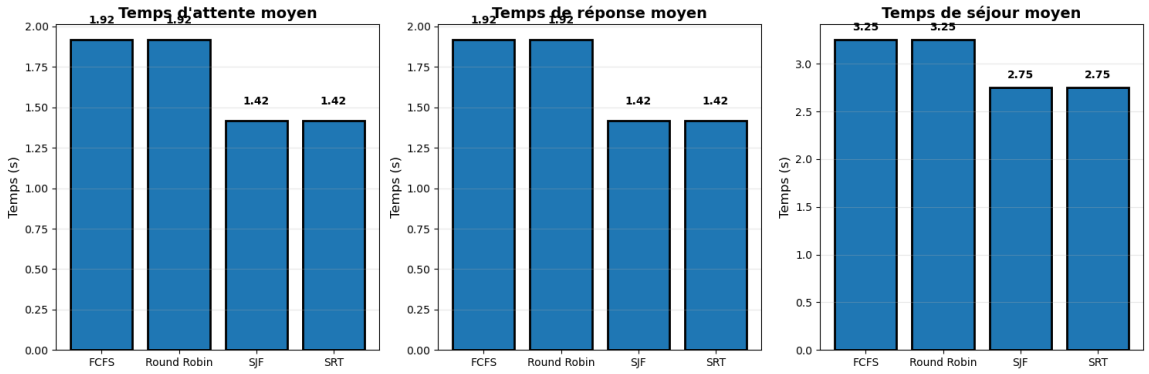
Donc nous pouvons constater que pour un semblable de type de tâche, l'ordonnanceur minimise le temps d'attente moyen et le temps de réponse moyen.

(b) Jeu de données 2

Dans le tableau ci-dessous, nous avons de nombreux processus avec un temps d'exécution moins élevé.

Processus	Temps d'exécution	Temps d'arrivage
A	1	0
B	1	0.5
C	2	1
D	1	1.5
E	2	2
F	1	2.5

Nous obtenons les diagrammes ci-dessous correspondants au temps de séjour moyen, au temps de réponse moyen et au temps d'attente moyen :



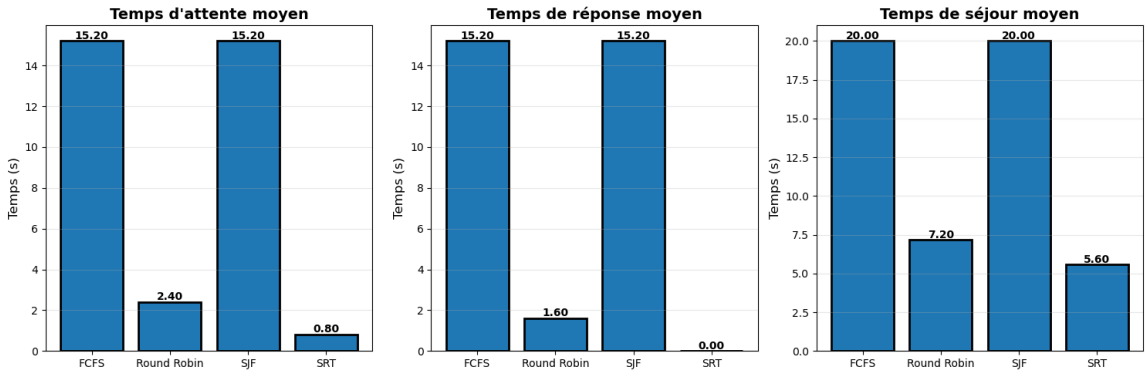
Selon ce type de jeu de données, nous remarquons que l'ordonnanceur favorise un temps d'attente moyen et un temps de réponse moyen tous deux relativement courts uniquement pour les méthodes SJF et SRT.

(c) Jeu de données 3

Dans le tableau ci-dessous, nous avons le processus arrivant premier qui possède un très grand nombre de temps d'exécution.

Processus	Temps d'exécution	Temps d'arrivage
A	20	0
B	1	1
C	1	2
D	1	3
E	1	4

Pour visualisation au niveau des temps de réponse moyen, de séjour moyen, d'attente moyen, nous présentons :



Nous remarquons immédiatement la faiblesse de la méthode d'ordonnancement FCFS qui monopolise le processeur et défavorise les processus ayant des courts temps d'exécution vu qu'ils sont prêts à l'exécution plus tard. Cependant, le SRT préemptif favorise les plus petits processus.

Les différents jeux de données présentés nous ont permis d'illustrer l'impact réel des politiques d'ordonnancement sur les performances d'un système. Nous avons pu observer que certaines méthodes, comme SJF ou SRT, sont particulièrement efficaces pour des charges composées de processus courts, tandis que d'autres, comme FCFS ou Round Robin, se comportent différemment selon les arrivées, la longueur des tâches ou la présence de phénomènes de convoi.
Il est important de rappeler que ces résultats ne couvrent pas l'ensemble des scénarios possibles. En pratique, l'efficacité d'une politique d'ordonnancement dépend fortement du contexte d'exécution, de la nature des processus, de leur fréquence d'arrivée et des ressources matérielles disponibles. Une étude plus large intégrant davantage de cas extrêmes ou réalistes permettrait d'avoir une vision encore plus complète

7. Bibliographie et webographie

« INF3600 Systèmes d'exploitation » de *Hanifa Boucheneb* et *Juan Manuel Torres-Moreno*

« COURS D'ALGORITHMIQUE » de *Professeur KONAN HYACINTHE*

« Programmer en C++ l'algorithme d'ordonnancement FCFS » : https://labex.io/fr/tutorials/cpp-c-program-for-fcfs-scheduling-algorithm-96161

« Exercices corrigés ordonnancement des processus » : https://waytolearnx.com/2024/10/exercice-corrige-ordonnancement-des-processus-partie-2.html

« Installer Jupyter Notebook » : https://waytolearnx.com/2024/10/exercice-corrige-ordonnancement-des-processus-partie-2.html