Automne 2024

Algorithmes et structures de données

GLO-2100/IFT-2008

Thierry EUDE, Ph.D
Kim Rioux-Paradis, M.Sc

Travail à faire individuellement
À rendre avant la date indiquée
sur le portail du cours
(voir modalités de remise à la fin de l'énoncé)

Ce travail doit être de votre propre création.
Tout étudiant qui commet une infraction au Règlement disciplinaire à l'intention des étudiants de l'Université Laval dans le cadre du présent cours, notamment en matière de plagiat, est passible des sanctions qui sont prévues dans ce règlement. Il est très important pour tout étudiant de prendre connaissance des articles 23 à 46 du Règlement disciplinaire. Celui-ci peut être consulté à l'adresse suivante:
http://ulaval.ca/reglement-disciplinaire
Tout étudiant est tenu de respecter les règles relatives au plagiat.
De plus, les étudiants ou étudiantes ayant collaboré au plagiat seront soumis aux sanctions normalement prévues à cet effet par les règlements de l'Université.

Tous les travaux sont soumis à un logiciel de détection de plagiat.

Travail Pratique #1
Ordonnancement de processus

UNIVERSITÉ LAVAL
Faculté des sciences et de génie
Département d'informatique
et de génie logiciel

Mise en contexte

Concept de processus :

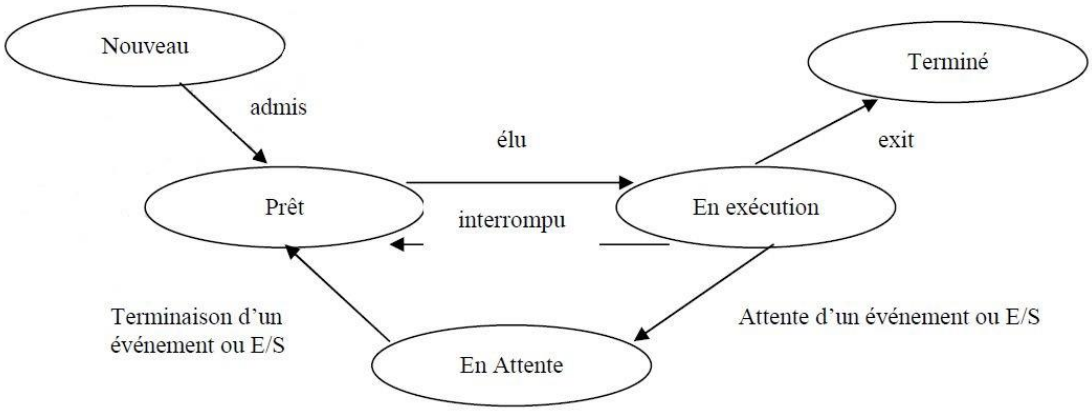
On peut trouver plusieurs appellations possibles des activités que peut avoir un processeur. Un système de traitement par lots exécute des travaux, tandis qu'un système en temps partagé possède des programmes utilisateurs ou des tâches.

Définition :

On peut définir un processus (process) comme un programme en exécution. Autrement dit, un programme par lui-même n'est pas un processus. Un programme est une entité passive, comme le contenu d'un fichier stocké sur disque, tandis qu'un processus est une entité active, avec un compteur d'instructions spécifiant l'instruction suivante à exécuter et un ensemble de ressources associées.
Évidemment, il est possible d'avoir plusieurs processus différents associés à un même programme. C'est le cas, par exemple, de plusieurs utilisateurs qui exécutent chacun une copie du programme de messagerie. Il est également courant qu'un processus génère à son tour plusieurs processus lors de son exécution.

Etats d'un processus :

Durant son séjour dans un SE, un processus peut changer d'état à plusieurs reprises. Ces états peuvent être résumés dans le schéma suivant :



Voici une brève explication de chacun de ces états :

- Nouveau : Le processus vient d'être créé.
- Prêt : le processus est placé dans la file d'attente des processus prêts, en attente d'affectation du processeur.
- En exécution : le processus a été affecté à un processeur libre. Ses instructions sont en exécution.
- En attente : Le processus attend qu'un événement se produise, comme l'achèvement d'une opération d'E/S ou la réception d'un signal.
- Terminé : le processus a terminé son exécution.

Ordonnancement de processus :

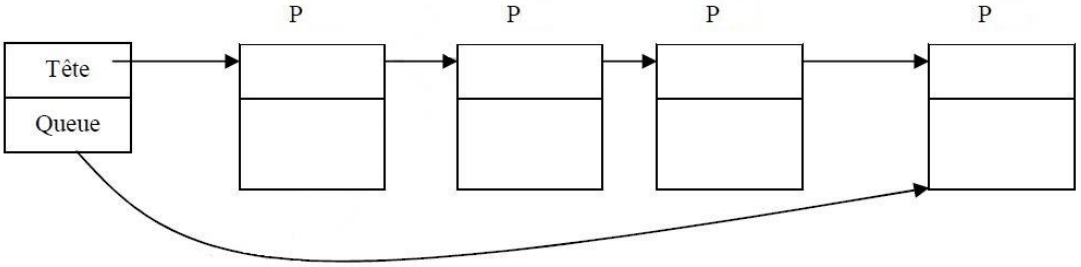
- **Files d'attentes d'ordonnancement:**

Pour gérer les processus durant leur séjour, le SE maintient plusieurs files d'attente. On peut citer entre autres :

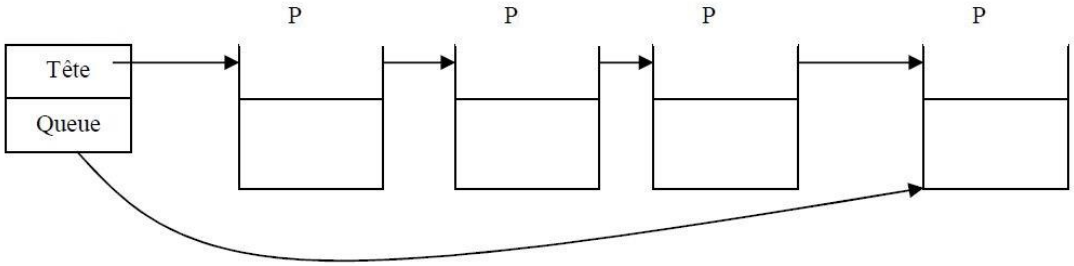
- File d'attente des processus prêts : Cette file contient tous les processus en attente du processeur.
- File d'attente de périphérique : Pour réguler les demandes d'allocation des différents périphériques, on peut imaginer une file d'attente pour chaque périphérique. Quand un processus demande une opération d'E/S, il est mis dans la file d'attente concernée.

Concrètement une file d'attente est représentée par une liste chaînée de PCB (Process Control Bloc – Bloc de contrôle *du processus*), comme le montre le schéma suivant.

File d'attente des processus prêts :



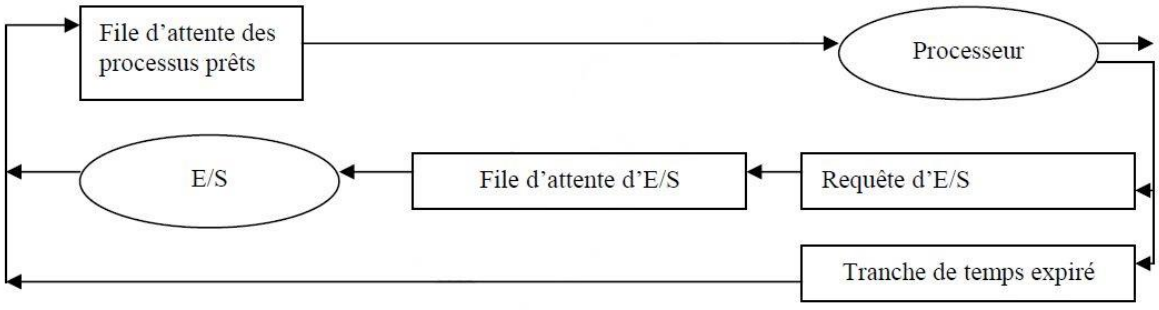
File d'attente d'un périphérique :



Un nouveau processus est initialement placé dans la file d'attente des processus prêts. Il attend dans cette file jusqu'à ce qu'il soit sélectionné pour son exécution et qu'on lui accorde le processeur. Une fois qu'on a alloué le processeur au processus et que celui-ci est en cours d'exécution, il pourrait se produire l'un des événements suivants :

- Le processus pourrait émettre une requête d'E/S et ensuite placé dans une file d'attente d'E/S.
- Le processus pourrait être enlevé du processeur ; on dit aussi que le processeur a été réquisitionné. Dans ce cas, le processus est remis dans la file d'attente des processus prêts.
- Le processus pourrait se terminer.

Les différents états de transition du processus entre les files d'attente sont résumés par la figure suivante.



- **L'ordonnanceur:**

L'ordonnanceur est un programme du SE qui s'occupe de choisir, selon une politique d'ordonnancement donnée, un processus parmi les processus prêts pour lui affecter le processeur.

- **Les critères d'ordonnancement:**

Les divers algorithmes d'ordonnancement du processeur possèdent des propriétés différentes et peuvent favoriser une classe de processus plutôt qu'une autre. Pour choisir quel algorithme utiliser dans une situation particulière, nous devons tenir compte des propriétés des divers algorithmes.
Plusieurs critères ont été proposés pour comparer et évaluer les performances des algorithmes d'ordonnancement du processeur. Les critères les plus souvent utilisés sont :

- Utilisation du processeur : Un bon algorithme d'ordonnancement sera celui qui maintiendra le processeur aussi occupé que possible.
- Capacité de traitement : C'est la quantité de processus terminés par unité de temps.
- Temps de restitution : C'est le temps s'écoulant entre la soumission du travail et sa terminaison.
- Temps d'attente : C'est le temps passé à attendre dans la file d'attente des processus prêts.
- Temps de réponse : C'est le temps passé dans la file d'attente des processus prêts avant la première exécution.

Il s'agit dans ce travail de construire un simulateur afin de comparer plusieurs algorithmes d'ordonnancement.

Les algorithmes d'ordonnancement:

L'algorithme du Premier Arrivé Premier Servi (FCFS) :

L'algorithme d'ordonnancement du processeur le plus simple est l'algorithme du Premier Arrivé Premier Servi (First Come First Served : FCFS). Avec cet algorithme, on alloue le processeur au premier processus qui le demande. L'implémentation de la politique FCFS est facilement gérée avec une file d'attente FIFO. Quand un processus entre dans la file d'attente des processus prêts, son PCB est enchaînée à la queue de la file d'attente. Quand le processeur devient libre, il est alloué au processeur en tête de la file d'attente.
Exemple : Trois processus P1, P2 et P3 arrivent dans cet ordre au système. Leurs durées d'exécution sont respectivement : 24, 3, 3 unités de temps. Pour représenter l'historique d'occupation du processeur, on utilise le diagramme suivant, appelé diagramme de Gantt :

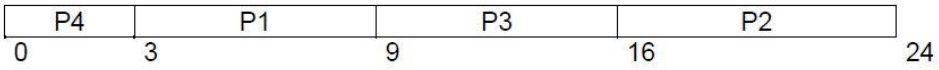
Ce diagramme montre que le processus P1 occupe le processeur de l'instant 0 jusqu'à l'instant 24. A l'instant 24, le processeur devient occupé par le processus P2, puis à l'instant 27 il sera suivi du processus P3.
Le temps d'attente est égal à 0 pour le processus P, 24 pour le processus P2 et 27 pour le processus P3. Le temps d'attente moyen est égal à : (0+24+27)/3, soit 17 unités de temps.

L'algorithme du Plus Court d'abord (SJF) :

Cet algorithme (en anglais Shortest Job First : SJF) affecte le processeur au processus possédant le temps d'exécution le plus court. Si plusieurs processus ont la même durée, une politique FIFO sera alors utilisée pour les départager.
Exemple : On soumet au système quatre processus P1, P2, P3 et P4 dont les durées d'exécution sont données par le tableau suivant :

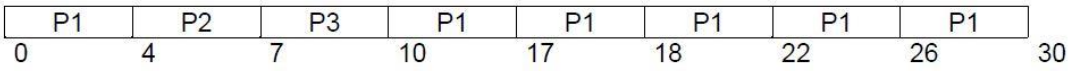
Processus	Durée d'exécution
P1	6
P2	8
P3	7
P4	3

L'algorithme du travail le plus court donnera alors le résultat suivant :



L'algorithme de Round Robin (Tourniquet) :

L'algorithme d'ordonnancement du tourniquet, appelé aussi Round Robin, a été conçu pour des systèmes à temps partagé. Il alloue le processeur aux processus à tour de rôle, pendant une tranche de temps appelée quantum.
Exemple : On dispose de 3 processus P1, P2 et P3 ayant comme durée d'exécution, respectivement 24, 3 et 3 ms. En utilisant un algorithme Round Robin, avec un quantum de 4 ms, on obtient le diagramme de Gantt suivant :

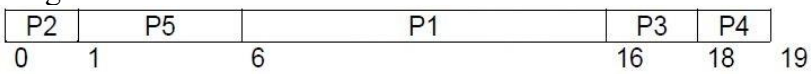


Ordonnancement avec priorité

Cet algorithme associe à chaque processus une priorité, et le processeur sera affecté au processus de plus haute priorité. Cette priorité varie selon les systèmes et peut aller de 0 à 127.
Les priorités peuvent être définies en fonction de plusieurs paramètres : le type de processus, les limites de temps, les limites mémoires, …etc.
Exemple : On dispose de 5 processus ayant des priorités différentes, comme le montre ce tableau :

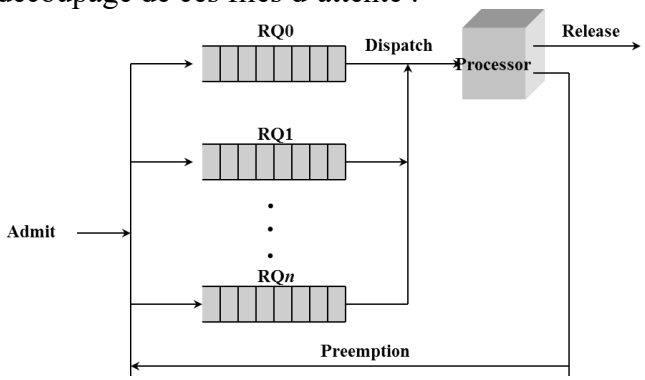
Processus	Durée d'exécution	Priorité
P1	10	2
P2	1	4
P3	2	2
P4	1	1
P5	5	3

Diagramme de Gantt :



Ordonnancement avec files d'attente multiniveaux :

Une autre classe d'algorithmes d'ordonnancement a été développée pour des situations où on peut facilement classer les processus dans des groupes différents. Par exemple, il serait intéressant de faire une distinction entre les processus de premier plan (interactifs) et les processus d'arrière-plan (traitement par lot). En effet, ces deux types de processus possèdent des besoins différents en ce qui concerne le temps de réponse et ils pourraient donc devoir être ordonnancés différemment. De plus les processus de premier plan peuvent être prioritaires par rapport aux processus d'arrière-plan.
Ainsi, un algorithme d'ordonnancement avec des files d'attente multiniveaux découpe la file d'attente des processus prêts en plusieurs files d'attentes séparées. La figure suivante donne un exemple de découpage de ces files d'attente :



Les processus sont en permanence assignés à une file d'attente en se basant généralement sur certaines propriétés du processus, comme : le type de processus, la taille de la mémoire, la priorité, … etc. Chaque file d'attente possède son propre algorithme d'ordonnancement. Par exemple, la file d'attente des processus systèmes est ordonnancée selon algorithme de plus haute priorité, celles des processus interactifs est gérée selon l'algorithme Round Robin et les files des processus batch et utilisateurs sont gérées selon l'algorithme FCFS.
D'autre part, il doit y avoir un ordonnancement entre les files d'attente elles-mêmes. Observons par exemple, l'ensemble des files d'attente multiniveaux suivants :
1. Processus systèmes
2. Processus interactifs
3. Processus batch
4. Processus utilisateurs

Chaque file d'attente est absolument prioritaire par rapport aux files d'attente de niveau inférieur. Par exemple, aucun processus de la file d'attente des processus batch ne pourra s'exécuter à moins que les files d'attente des processus système et interactifs ne soient toutes vides. De plus, si un processus interactif arrive au système, alors qu'un processus batch est en train de s'exécuter, celui-ci doit être interrompu.

Processus	Type	Durée	Priorité
P1	systemes	5	2
P2	batch	10	1
P3	interactifs	7	1
P4	utilisateurs	9	1
P5	systemes	4	4
P6	interactifs	6	1
P7	utilisateurs	5	1
P8	batch	15	1

Travail à réaliser

Il s'agit de réaliser un simulateur afin de pouvoir comparer les algorithmes d'ordonnancement décrits plus haut. Le développement en langage C++ sera composé de 3 modules :

- Un module Processus (classe)
- Un module Ordonnanceur (regroupant l'ensemble des implémentations des algorithmes)
- Un programme de simulation

Dans tous les algorithmes d'ordonnancement, on a recourt à des files. Vous pouvez utiliser ou vous inspirer des implémentations présentées en cours en laboratoire. Cependant, non seulement on doit avoir la possibilité de rechercher un élément dans la file, mais aussi de pouvoir l'y extraire (cf. algorithme FJS ou par priorité).

Module Processus

Ce module permet de modéliser les processus qui seront manipulés par des fonctions implémentant les algorithmes d'ordonnancement.
Un processus sera modélisé par différents attributs tels que :

```
std::string m_pid; //identifiant pour l'affichage du résultat
int m_arrivee; // (instant d'arrivée du processus dans le système)
int m_duree; //(duree d'exécution initiale du processus)
int m_restant; //temps restant à exécuter
int m_attente; // (temps d'attente cumulé)
int m_fin;//temps à la préhemption (RR)
int m_priorite; // priorité du processus
TypeProcessus m_type; // type du processus pour multiniveaux
```

Prévoyez des accesseurs et mutateurs en conséquence ainsi que la surcharge d'opérateurs pour faciliter la recherche de relations d'ordre (ordonnancement par priorité) ou l'affichage des résultats.
Pour améliorer la lisibilité du code, pensez à utiliser une liste d'énumération pour les différents types de processus (ordonnancement multiniveau).

Module Ordonnanceur

Ce module regroupe l'implémentation des différents algorithmes d'ordonnancement sous forme de fonctions.

- **Fonctions à développer**

Les fonctions devront avoir OBLIGATOIREMENT les prototypes suivant :

File< Processus > TP::fcfs (File< Processus > f_entree, int& temps)

Paramètres:

[in]	**f_entree**	Une file de processus contenant la simulation
[in,out]	**temps**	le temps qui s'écoule

Renvoie:

une file par valeur contenant le résultat de l'exécution de la simulation, les processus ont été mis à jour selon leur exécution

File< Processus > TP::fjs (File< Processus > f_entree, int& temps)

le processus le plus court est retiré de la file pour être exécuté au complet.
La file "pret" est mise à jour avant la sélection d'un nouveau processus prêt pour exécution si de nouveaux processus sont arrivés entre temps.

Paramètres:

[in]	**f_entree**	Une file de processus contenant la simulation
[in,out]	**temps**	le temps qui s'écoule

Renvoie:

une file par valeur contenant le résultat de l'exécution de la simulation, les processus ont été mis à jour selon leur exécution

File< Processus > TP::round_robin (File< Processus > f_entree, int quantum, int& temps)

le premier processus à l'état prêt est exécuté pendant un quantum de temps s'il n'est pas exécuté au complet, il est remis (enfiler) dans la liste d'attente avant de défiler le suivant.
La file "pret" est mise à jour avant la sélection d'un nouveau processus prêt pour exécution si de nouveaux processus sont arrivés entre temps.

Paramètres:

[in]	**f_entree**	Une file de processus contenant la simulation
[in]	**quantum**	la durée du quantum d'exécution
[in,out]	**temps**	le temps qui s'écoule

Renvoie:

une file par valeur contenant le résultat de l'exécution de la simulation, les processus ont été mis à jour selon leur exécution

File< Processus > TP::priorite (File< Processus > f_entree, int& temps)

le processus ayant la priorité la plus élevée est retiré de la file pour être exécuté au complet, seuls les processus arrivés sont enfilés dans la file "pret".
la file "pret" est mise à jour avant la sélection d'un nouveau processus prêt pour exécution si de nouveaux processus sont arrivés entre temps.

Paramètres:

[in]	**f_entree**	Une file de processus contenant la simulation
[in,out]	**temps**	le temps qui s'écoule

Renvoie:

une file par valeur contenant le résultat de l'exécution de la simulation, les processus ont été mis à jour selon leur exécution

File< Processus > **multiniveaux** (**File**< Processus > f_entree, int quantum, int &temps)

Les processus prêts sont ordonnancés selon leur type. Dans l'ordre : Les processus système sont ordonnancés selon leur priorité, les processus interactifs sont ordonnancés selon Round Robin, les processus batch ainsi que les processus utilisateurs sont ordonnancés selon FCFS. Tous les processus sont supposés être arrivés à temps=0 (on ne considère pas les processus arrivés entre temps)

Paramètres:

[in]	**f_entree**	Une file de processus contenant la simulation
[in]	**quantum**	la durée du quantum d'exécution pour les processus interractifs
[in,out]	**temps**	le temps qui s'écoule

Renvoie:

une file par valeur contenant le résultat de l'exécution de la simulation, les processus ont été mis à jour selon leur exécution

Programme de simulation

Ce programme consiste à charger successivement des scénarios de simulation mémorisés dans des fichiers texte et de présenter le résultat de l'application des algorithmes d'ordonnancement sur ces scénarios (voir l'exemple d'exécution à la fin de cet énoncé). Les fichiers contenant les scénarios doivent être prévus pour mettre en évidence les avantage et inconvénients des algorithmes évalués. Des exemples de fichiers vous sont fournis. Vous devez respecter leur format.

Erreurs de compilation

Si un travail comporte ne serait-ce qu'une erreur de compilation, il sera fortement pénalisé, et peut même se voir attribuer la note zéro systématiquement.

Erreurs à l'exécution

Testez intensivement vos programmes. Rappelez-vous qu'un programme qui ne plante pas n'est pas nécessairement sans bogue, mais qu'un programme qui plante même une seule fois comporte nécessairement un bogue. D'ailleurs, si un programme ne plante nulle part sauf sur l'ordinateur de votre ami, c'est qu'**il comporte un bogue**. Il risque donc de planter un jour ou l'autre sur votre ordinateur et, encore pire, sur celui des correcteurs.
Si vous ne testez pas suffisamment votre travail, il risque de provoquer des erreurs à l'exécution lors de la correction. La moindre erreur d'exécution rend la correction extrêmement difficile et vous en serez donc fortement pénalisés.

Bien livrable :

Le premier travail pour le cours GLO-2100/IFT-2008 est **à faire INDIVIDUELLEMENT.**
Tous vos développements doivent être faits dans le dépôt github privé qui vous est créé après avoir accepté « l'assignment »
https://classroom.github.com/a/-zNoQD9G
RAPPEL :
Vous **DEVEZ ABSOLUMENT CHOISIR VOTRE COURRIEL dans la liste** avant d'accepter l'assignment sinon, on ne pourra pas vous identifier lors de la correction et vous donner votre résultat.

On devra y retrouver minimalement les fichiers source : *Processus.h, Processus.cpp, Ordonnanceur.h, Ordonnanceur.cpp* et *simulateur.cpp*, les fichiers nécessaires à leur compilation ainsi que les fichiers texte de scénarios que vous avez utilisé pour faire votre simulation.
Il est fortement recommandé d'utiliser la théorie du contrat pour fiabiliser votre code. L'outil pour l'implémenter est disponible dans notes de cours/Documents divers
Votre travail sera évalué dans la machine virtuelle « officielle » du cours. Il sera compilé avec Cmake. Votre remise doit donc aussi inclure le fichier CMakeLists.txt permettant de compiler votre code dans la machine virtuelle officielle du cours.
Il est donc important de vérifier la portabilité de votre code avec l'environnement de correction ceci avant la remise.

Critères d'évaluation

1) Respect des biens livrables
2) portabilité de votre code source
3) Structure, organisation du code, lisibilité
4) Exactitude du code (fonctionnalité)

Particularités du barème

- *Si des pénalités sont appliquées, elles le seront sur l'ensemble des points.*
- *Si un travail comporte ne serait-ce qu'une erreur de compilation, il sera fortement pénalisé, et peut même se voir attribuer la note zéro systématiquement.*
- *Il est très important que votre travail respecte strictement les consignes indiquées dans l'énoncé, en particulier les prototypes des fonctions, sous peine de fortes pénalités*

Plagiat

Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

Bon travail

Exemple d’exécution du simulateur

```
p1 arrivee : 0 Duree : 24 temps d'attente : 0 Priorite : 1 Type : 1
p2 arrivee : 0 Duree : 3 temps d'attente : 0 Priorite : 1 Type : 1
p3 arrivee : 0 Duree : 3 temps d'attente : 0 Priorite : 1 Type : 1
p4 arrivee : 30 Duree : 2 temps d'attente : 0 Priorite : 1 Type : 1
fin chargement

simulation de FCFS Resultat:
p1 arrivee : 0 Duree : 24 temps d'attente : 0 Priorite : 1 Type : 1
p2 arrivee : 0 Duree : 3 temps d'attente : 24 Priorite : 1 Type : 1
p3 arrivee : 0 Duree : 3 temps d'attente : 27 Priorite : 1 Type : 1
p4 arrivee : 30 Duree : 2 temps d'attente : 0 Priorite : 1 Type : 1
Temps d'attente moyen : 12.75
simulation de FJS Resultat:
p2 arrivee : 0 Duree : 3 temps d'attente : 0 Priorite : 1 Type : 1
p3 arrivee : 0 Duree : 3 temps d'attente : 3 Priorite : 1 Type : 1
p1 arrivee : 0 Duree : 24 temps d'attente : 6 Priorite : 1 Type : 1
p4 arrivee : 30 Duree : 2 temps d'attente : 0 Priorite : 1 Type : 1
Temps d'attente moyen : 2.25
simulation de Round Robin Resultat:
p2 arrivee : 0 Duree : 3 temps d'attente : 4 Priorite : 1 Type : 1
p3 arrivee : 0 Duree : 3 temps d'attente : 7 Priorite : 1 Type : 1
p1 arrivee : 0 Duree : 24 temps d'attente : 6 Priorite : 1 Type : 1
p4 arrivee : 30 Duree : 2 temps d'attente : 0 Priorite : 1 Type : 1
Temps d'attente moyen : 4.25

p1 arrivee : 0 Duree : 10 temps d'attente : 0 Priorite : 2 Type : 1
p2 arrivee : 0 Duree : 1 temps d'attente : 0 Priorite : 4 Type : 1
p3 arrivee : 0 Duree : 2 temps d'attente : 0 Priorite : 2 Type : 1
p4 arrivee : 0 Duree : 1 temps d'attente : 0 Priorite : 1 Type : 1
p5 arrivee : 0 Duree : 5 temps d'attente : 0 Priorite : 3 Type : 1
fin chargement
simulation avec priorite Resultat:
p2 arrivee : 0 Duree : 1 temps d'attente : 0 Priorite : 4 Type : 1
p5 arrivee : 0 Duree : 5 temps d'attente : 1 Priorite : 3 Type : 1
p1 arrivee : 0 Duree : 10 temps d'attente : 6 Priorite : 2 Type : 1
p3 arrivee : 0 Duree : 2 temps d'attente : 16 Priorite : 2 Type : 1
p4 arrivee : 0 Duree : 1 temps d'attente : 18 Priorite : 1 Type : 1
Temps d'attente moyen : 8.2

p1 arrivee : 0 Duree : 5 temps d'attente : 0 Priorite : 2 Type : 1
p2 arrivee : 0 Duree : 10 temps d'attente : 0 Priorite : 1 Type : 3
p3 arrivee : 0 Duree : 7 temps d'attente : 0 Priorite : 1 Type : 2
p4 arrivee : 0 Duree : 9 temps d'attente : 0 Priorite : 1 Type : 4
p5 arrivee : 0 Duree : 4 temps d'attente : 0 Priorite : 4 Type : 1
p6 arrivee : 0 Duree : 6 temps d'attente : 0 Priorite : 1 Type : 2
p7 arrivee : 0 Duree : 5 temps d'attente : 0 Priorite : 1 Type : 4
p8 arrivee : 0 Duree : 15 temps d'attente : 0 Priorite : 1 Type : 3
fin chargement
simulation multiniveaux Resultat:
p5 arrivee : 0 Duree : 4 temps d'attente : 0 Priorite : 4 Type : 1
p1 arrivee : 0 Duree : 5 temps d'attente : 4 Priorite : 2 Type : 1
p3 arrivee : 0 Duree : 7 temps d'attente : 13 Priorite : 1 Type : 2
p6 arrivee : 0 Duree : 6 temps d'attente : 16 Priorite : 1 Type : 2
p2 arrivee : 0 Duree : 10 temps d'attente : 22 Priorite : 1 Type : 3
p8 arrivee : 0 Duree : 15 temps d'attente : 32 Priorite : 1 Type : 3
p4 arrivee : 0 Duree : 9 temps d'attente : 47 Priorite : 1 Type : 4
p7 arrivee : 0 Duree : 5 temps d'attente : 56 Priorite : 1 Type : 4
Temps d'attente moyen : 23.75

Fin du programme
```