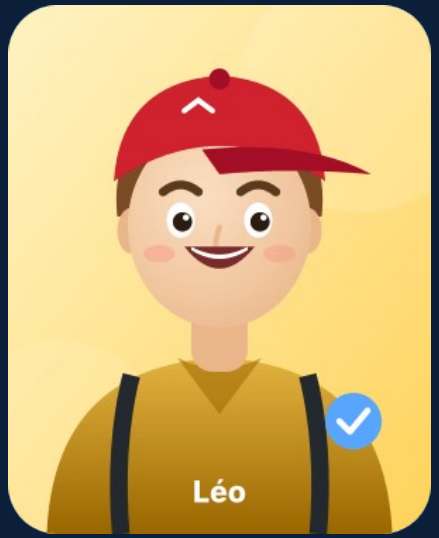
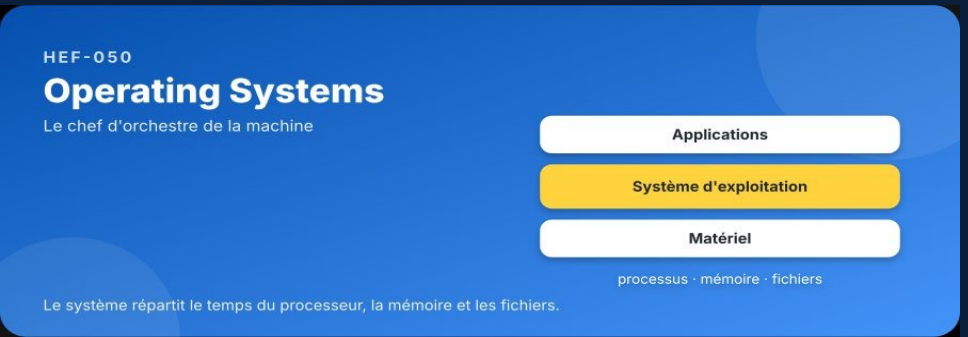
HEF-050 · MISSION 04 / 10

L'ordonnancement

Quand plusieurs programmes veulent le processeur, qui passe en premier ?





Badge : Chef de planning

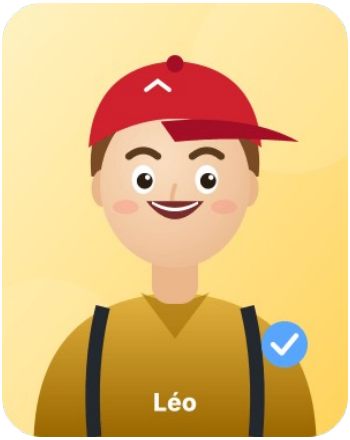
70 min · intermédiaire

L'histoire

L'histoire

Un serveur de HashBox exécute à la fois des tâches très courtes (afficher une page) et une tâche très longue (générer le rapport mensuel). Parfois, les pages s'affichent avec un grand retard.

> *« Le processeur a décidé de finir le rapport avant, dit Sam. Une règle d'ordonnancement différente donnerait de meilleurs résultats pour les utilisateurs. »*



Tu apprends à comparer des politiques d'ordonnancement avec des métriques précises.

Ta mission

À la fin, je serai capable de…

1. définir ordonnancement, quantum, préemption
2. simuler FCFS, SJF et Round Robin
3. calculer temps d'attente et temps de rotation
4. expliquer le compromis entre équité, débit et réactivité
5. expliquer le problème de la famine

Défi d'échauffement

Défi d'échauffement

À la poste, il y a un seul guichet. Trois clients arrivent en même temps : A (30 min), B (2 min), C (3 min). Dans quel ordre les servir pour que **la moyenne des temps d'attente** soit la plus faible ? Calcule l'attente moyenne pour l'ordre A, B, C puis pour B, C, A.

Le concept

Le rôle de l'ordonnanceur

Il y a plus de processus prêts que de cœurs. L'**ordonnanceur** (*scheduler*) décide, à chaque instant, **quel processus reçoit le CPU** et pour combien de temps.

Les métriques

Métrique	Définition
Temps d'attente	temps passé à attendre le CPU dans la file des prêts
Temps de rotation (turnaround)	du moment d'arrivée à la fin = attente + exécution
Temps de réponse	de l'arrivée à la première exécution
Débit	nombre de tâches terminées par unité de temps

Le concept

Les principales politiques

Politique	Principe	Avantage	Défaut
FCFS (premier arrivé, premier servi)	file simple	simple, équitable en ordre	effet « convoi » : une longue tâche retarde tout
SJF (plus court d'abord)	on exécute la tâche la plus courte	attente moyenne minimale	famine des longues, durée à deviner
Round Robin	chacun son tour pendant un **quantum**	réactif, équitable	trop de changements si le quantum est trop petit
Priorités	on exécute le plus prioritaire	favorise l'urgent	famine des basses priorités

Le concept

La préemption

Un ordonnanceur est **préemptif** quand il peut **interrompre** un processus avant qu'il ait fini (fin du quantum, arrivée d'un processus plus prioritaire). Les systèmes modernes le sont tous.

Le changement de contexte

Passer d'un processus à un autre coûte : sauvegarder les registres, restaurer ceux de l'autre, vider des caches. C'est pourquoi un quantum trop petit gaspille du temps.

La famine

Une tâche est en **famine** si elle n'obtient jamais le CPU. Remède : **vieillissement** (*aging*) : la priorité augmente avec l'attente.

Le concept

Simuler FCFS

```
def fcfs(taches):
    # taches : liste de (nom, duree) toutes arrivées à t = 0
    temps, attentes = 0, {}
    for nom, duree in taches:
        attentes[nom] = temps
        temps += duree
    return attentes

attentes = fcfs([("A", 30), ("B", 2), ("C", 3)])
print(attentes, sum(attentes.values()) / len(attentes))   # attentes 0, 30, 32 : m
   oyenne ≈ 20,7
```

🧠 ***Astuce*** *: SJF minimise l'attente moyenne, mais Round Robin donne le meilleur sentiment de réactivité.*

Exemple pas à pas

Problème : comparer FCFS, SJF et Round Robin (quantum 2) sur les tâches A (6), B (2), C (4), toutes arrivées à t = 0.

```
def fcfs(taches):
    t, fin = 0, {}
    for nom, d in taches:
        t += d
        fin[nom] = t
    return fin

def sjf(taches):
    return fcfs(sorted(taches, key=lambda x: x[1]))

def round_robin(taches, quantum):
    restant = dict(taches)
    file = [n for n, _ in taches]
    t, fin = 0, {}
    while file:
```

Exemple pas à pas (suite)

```
        nom = file.pop(0)
        run = min(quantum, restant[nom])
        t += run
        restant[nom] -= run
        if restant[nom] == 0:
            fin[nom] = t
        else:
            file.append(nom)
    return fin

taches = [("A", 6), ("B", 2), ("C", 4)]
duree = dict(taches)
for nom, f in (("FCFS", lambda: fcfs(taches)), ("SJF", lambda: sjf(taches)), ("RR
   q=2", lambda: round_robin(taches, 2))):
    fin = f()
    rotation = sum(fin.values()) / len(fin)
    attente = sum(fin[n] - duree[n] for n in fin) / len(fin)
    print(f"{nom:7s} fin={fin}  rotation moy={rotation:.2f}  attente moy={attente:
   .2f}")
```

Exemple pas à pas (suite)

Résultats attendus :

Politique	Ordre de fin	Attente moyenne
FCFS	A(6), B(8), C(12)	4,67
SJF	B(2), C(6), A(12)	2,67
Round Robin (q = 2)	B(4), C(10), A(12)	4,67

Lecture : SJF minimise l'attente moyenne (2,67 contre 4,67). Round Robin a ici la même attente moyenne que FCFS, mais il offre une meilleure **réponse** aux tâches courtes : B démarre dès t = 2 (contre t = 6 en FCFS) et C dès t = 4 (contre t = 8), sans qu'on ait besoin de connaître les durées à l'avance.

À toi, avec aide

Tâches : A (durée 10), B (durée 1), C (durée 1), toutes arrivées à t = 0.

1. Calcule à la main l'attente moyenne en FCFS dans l'ordre A, B, C.
2. Calcule l'attente moyenne en SJF.
3. Vérifie avec les fonctions de l'exemple.
4. Quel est l'effet de convoi observé ?

À toi, avec aide · réponse

Indices progressifs

Indice 1

FCFS : A attend 0, B attend 10, C attend 11.

Indice 2

SJF : B attend 0, C attend 1, A attend 2.

Indice 3

L'effet de convoi vient de la longue tâche placée en tête.

À toi, avec aide · réponse

Réponse

FCFS : (0 + 10 + 11) / 3 = 7. SJF : (0 + 1 + 2) / 3 = 1.

```
def fcfs_attentes(taches):
    t, att = 0, {}
    for n, d in taches:
        att[n] = t
        t += d
    return att

for nom, ordre in (("FCFS", [("A", 10), ("B", 1), ("C", 1)]), ("SJF", [("B", 1), (
   "C", 1), ("A", 10)])):
    a = fcfs_attentes(ordre)
    print(nom, a, sum(a.values()) / 3)
```

L'effet de convoi : A, très long, retient les deux petites tâches qui attendent 10 et 11 unités pour rien.

À toi, en solo

Round Robin avec arrivées. Les tâches n'arrivent pas toutes à t = 0 : `(nom, arrivée, durée)`. Écris `round_robin_arrivees(taches, quantum)` qui renvoie, pour chaque tâche, son temps de fin, son temps d'attente et son temps de réponse. Teste avec `[("A", 0, 5), ("B", 1, 3), ("C", 2, 1)]` et quantum 2.
Étudie l'effet du quantum (1, 2, 4, 10) sur le nombre de changements de contexte et sur l'attente moyenne.

À toi, en solo · réponse

Pistes de correction

```
def round_robin_arrivees(taches, quantum):
    taches = sorted(taches, key=lambda x: x[1])
    restant = {n: d for n, _, d in taches}
    arrivee = {n: a for n, a, _ in taches}
    duree = dict(restant)
    premiere = {}
    fin, file = {}, []
    t, i, changements, dernier = 0, 0, 0, None
    while len(fin) < len(taches):
        while i < len(taches) and taches[i][1] <= t:
            file.append(taches[i][0]); i += 1
        if not file:
            t = taches[i][1]; continue
        nom = file.pop(0)
        if dernier is not None and dernier != nom:
            changements += 1
        dernier = nom
        premiere.setdefault(nom, t)
        run = min(quantum, restant[nom])
```

Pistes de correction (suite)

```
        t += run
        restant[nom] -= run
        while i < len(taches) and taches[i][1] <= t:
            file.append(taches[i][0]); i += 1
        if restant[nom] == 0:
            fin[nom] = t
        else:
            file.append(nom)
    attente = {n: fin[n] - arrivee[n] - duree[n] for n in fin}
    reponse = {n: premiere[n] - arrivee[n] for n in fin}
    return fin, attente, reponse, changements

taches = [("A", 0, 5), ("B", 1, 3), ("C", 2, 1)]
for q in (1, 2, 4, 10):
    fin, att, rep, ch = round_robin_arrivees(taches, q)
    print(q, "attente moy", round(sum(att.values()) / 3, 2), "réponse moy", round(
   sum(rep.values()) / 3, 2), "changements", ch)
```

À toi, en solo · réponse

Pistes de correction (suite)

Quantum petit : réponse rapide mais beaucoup de changements de contexte. Quantum très grand : on retombe sur FCFS.

Pièges classiques

Piège	Pourquoi c'est un problème	Comment l'éviter
Quantum trop petit	Trop de changements de contexte	Choisir un quantum de quelques millisecondes
Quantum trop grand	Round Robin devient FCFS	Trouver l'équilibre
SJF sans garde-fou	Famine des longues tâches	Vieillissement des priorités
Ignorer les arrivées	Simulation irréaliste	Modéliser les instants d'arrivée

Mini-boss

Comparateur d'ordonnanceurs. Génère un jeu de 12 tâches aléatoires (graine fixée) avec arrivées et durées. Implémente FCFS, SJF (non préemptif), Round Robin et un ordonnanceur **à priorités avec vieillissement**.
Pour chacun, calcule : attente moyenne, temps de réponse moyen, attente maximale (équité) et nombre de changements de contexte. Présente un tableau et écris une recommandation pour un serveur web (réactivité) et pour un traitement par lots (débit).

Mini-boss · réponse

Pistes de correction

Un bon travail : une fonction de simulation commune qui prend une « règle de choix » ; un tableau de 4 lignes × 4 métriques ; l'observation que SJF a la meilleure attente moyenne mais une attente maximale élevée pour les longues tâches ; que Round Robin a le meilleur temps de réponse ; que le vieillissement réduit l'attente maximale. Recommandation : Round Robin ou priorités avec vieillissement pour le web ; SJF/FCFS pour un traitement par lots où le débit prime.

Quiz éclair

Quiz éclair

1. L'ordonnanceur décide :

A. quel processus reçoit le CPU
B. quel fichier ouvrir
C. quelle adresse IP utiliser

2. L'effet de convoi est dû à :

A. un quantum trop petit
B. une longue tâche en tête de file
C. un manque de RAM

3. Round Robin donne :

A. le CPU au plus court

Quiz éclair

Quiz éclair (suite)

B. le CPU au plus prioritaire
C. à chaque processus le CPU à tour de rôle pour un quantum

4. 🔁 *Révision mission 03* : Une condition de course est :

A. un résultat qui dépend de l'ordre d'exécution
B. un bug de syntaxe
C. un type de verrou

5. 🔁 *Révision mission 02* : Un processus est :

A. un fichier exécutable
B. un programme en cours d'exécution
C. un type de mémoire

Quiz éclair · réponse

Réponses

1. **A**.
2. **B**.
3. **C**.
4. **A**.
5. **B**.

Je réfléchis

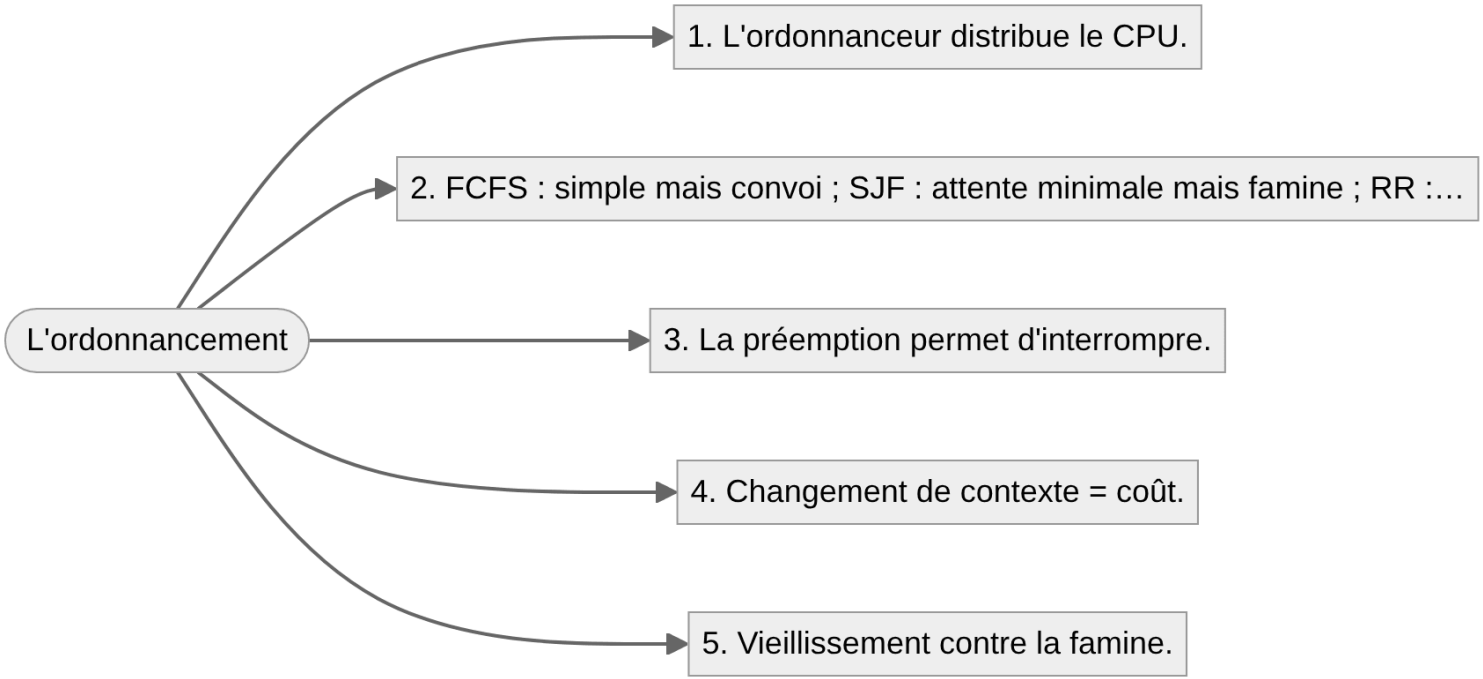
1. Dans ta journée, comment organises-tu tes tâches : plus court d'abord, tour de rôle, priorités ?
2. Quel risque de « famine » existe dans ta vie professionnelle ou personnelle ?

À retenir

1. L'ordonnanceur distribue le CPU.
2. FCFS : simple mais convoi ; SJF : attente minimale mais famine ; RR : réactif.
3. La préemption permet d'interrompre.
4. Changement de contexte = coût.
5. Vieillissement contre la famine.

À retenir

Le schéma de la mission



MISSION ACCOMPLIE

+100 XP

Badge débloqué : Chef de planning

Prochaine mission : La mémoire virtuelle

