Ordonnancement

SYE

8 - Ordonnancement

Résumé du document

La documentation présente les politiques d'ordonnancement, comme FCFS, SJF, et RR, pour gérer l'exécution des processus selon divers critères. Elle aborde la préemption pour prioriser les processus urgents et l'ordonnancement par files d'attente multiples, où les processus sont regroupés et traités par priorité. Des mécanismes de rétroaction ajustent dynamiquement les priorités, évitant ainsi la famine. Ces techniques visent à optimiser l'utilisation du CPU tout en garantissant des performances équilibrées.

Table des matières

1. Ordonnacement et mesures

Les différents algorithmes d'ordonnancement ou appelé politiques d'ordonnancement sont utilisés pour gérer les processus en attente d'exécution. Ces algorithmes permettent de déterminer l'ordre d'exécution des processus en fonction de différents critères.

- Durée d'exécution
  - Peut-être estimée ou mesurée
- Délai d'attente
  - Temps écoulé dans l'état **READY**

1.1. Complément du cours

*Un processus effectue une alternance de cycles CPU (aussi appelé **burst**) et de cycles I/O (interactions avec le matériel). Comme l'ordonnanceur n'a pas d'influence directe sur les processus dans l'état waiting, on s'intéressera particulièrement aux moments où les processus disposent du processeur, c-à-d du moment où ceux-ci se retrouvent dans l'état running et à ceux où les processus seront dans l'état READY. C'est pourquoi, le critère le plus intéressant pour la mesure de performance sera le délai d'attente.*

2. Préemption de processus

Un processus peut être interrompu par l'ordonnanceur pour donner la main à un autre processus. On parle de préemption. Pour être préempté, un processus doit être dans l'état **READY**.

On appelle point de préemption le moment où un processus est interrompu par l'ordonnanceur. Ces points peuvent suvenir à différents moments tel que **à la fin d'une routine de service** ou encore **en fin de traitement d'un appel système**.

*A chaque point de préemption, l'ordonnanceur est activé et peut **"décider"** de préempter éventuellement le processus en cours d'exécution (running). On parle également d'une approche opportuniste.*

2.1. Role de l'ordonnanceur

L'ordonnanceur a également pour rôle d'attribuer les processus (ou threads) aux différents processeurs (CPU), en tenant compte de facteurs tels que leur charge ou la nécessité qu'un processus s'exécute sur un processeur spécifique (comme un GPU ou d'autres coprocesseurs). Les ordonnanceurs modernes sont capables, dans certaines situations, de transférer un ou plusieurs processus d'un CPU à un autre.

2.2. Ordonnancement préemptif vs non préemptif

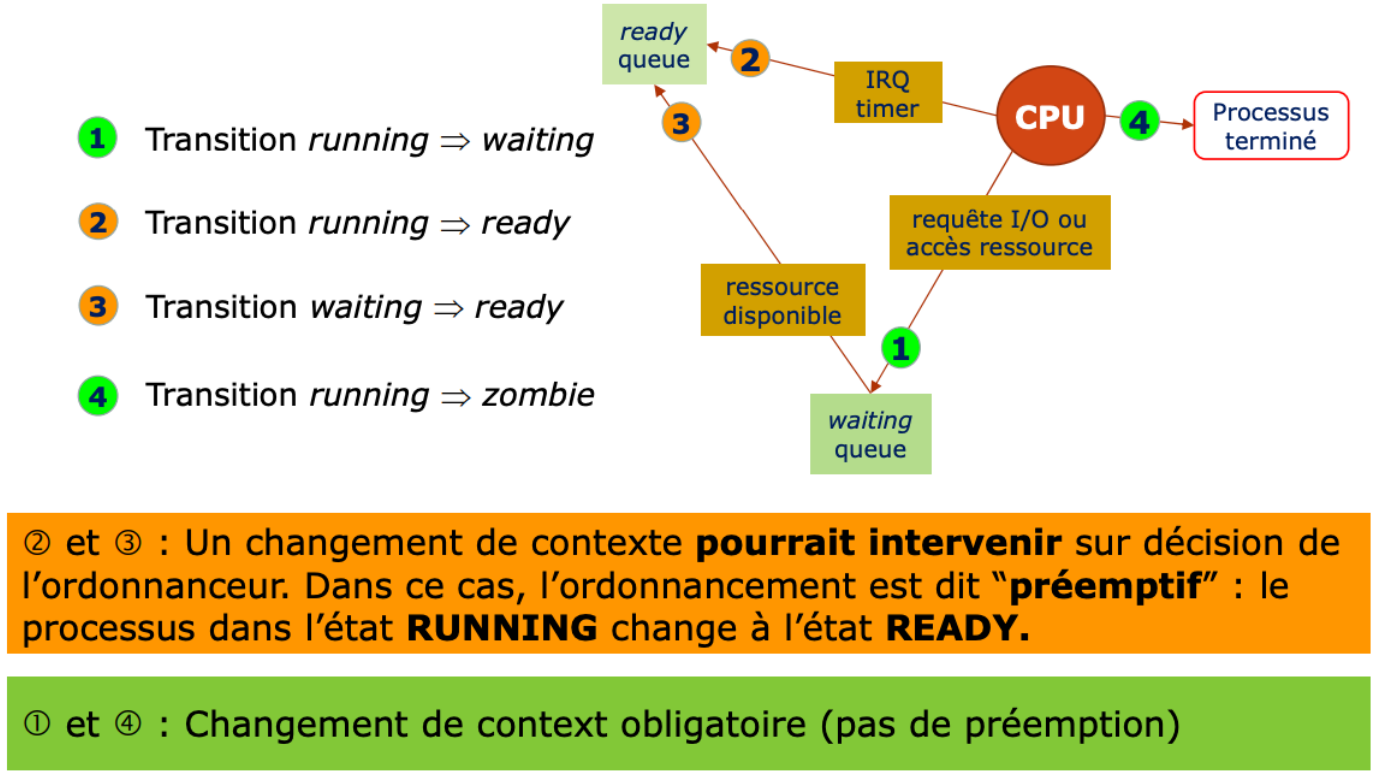


Figure 1: Image tirée des slides de cours, chapitre 8 - Scheduling

3. Ordonnancement FCFS, SJF, RR

3.1. Ordonnancement FCFS

Le premier arrivé est le premier servi. Les processus sont exécutés dans l'ordre d'arrivée. **FCFS** signifie **First Come First Served**.

Il s'agit d'un ordonnancement non préemptif et possède les caractéristiques suivantes:
- Gestion des processus FIFO
- Pas de préemption
- Simple à mettre en oeuvre
- N'est **pas** optimisé

Dans le cas ou un très long processus serait en état **RUNNING**, les processus suivants devront attendre longtemps peut-importe leur durée d'exécution ou criticité.

3.2. Ordonnancement SJF

Le plus court d'abord. Les processus sont exécutés en fonction de leur durée d'exécution. **SJF** signifie **Shortest Job First**.

Il s'agit d'un ordonnancement pouvant être préemptif ou non préemptif et possède les caractéristiques suivantes:
- Gestion des processus par durée d'exécution
- Difficle à implémenter
  - Nécessite une estimation de la durée d'exécution
- Optimisé pour minimiser le temps d'attente

Dans le cas de la version premptive, on parle plus de *shortest **remaining** time first* car si un job est préempté, on traitera désomrais le temps restant.

3.2.1. Estimation de la durée d'exécution

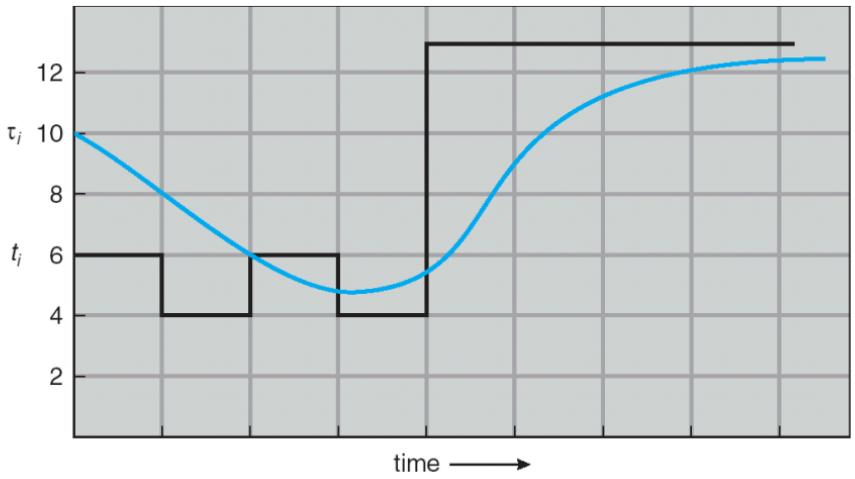
Cet ordonnancement nécessite une estimation de la durée d'exécution des processus. Pour cela on peut utliser la moyenne exponentielle.

$$t_n = \text{durée effective mesurée du burst n}$$
$$T_n + 1 = \text{durée estimée du burst n+1}$$

$$T_n + 1 = \alpha t_n + (1 - \alpha) T_n, \text{ où } 0 < \alpha < 1$$

Plus α est proche de 1, plus la moyenne est sensible aux variations dans ce cas on appelera ça une correction rapide.

- **$\alpha = 0.5$**



Durée effective:	6	4	6	4	13	13	13	13	…
Estimation:	10	8	6	6	5	9	11	12	…

3.2.2. Risque de famine

Le risque de famine est un problème qui peut survenir avec l'ordonnancement SJF. En effet, si des processus courts arrivent en permanence, les processus longs ne seront jamais exécutés. C'est pourquoi cette politique comme d'autres sera combinée avec des techniques d'ordonnancement plus sophistiquées, basées notamment sur les priorités dynamiques, permettant de traiter ce problème. Ces techniques sont abordées dans ce chapitre.

3.3. Ordonnancement RR

Les processus sont exécutés en fonction de leur durée d'exécution mais avec un quantum de temps. Cela signifie que chaque processus possède un temps fixe définit et sera préempté dans le cas où il n'a pas terminé son exécution à la fin de ce temps. **RR** signifie **Round Robin**.

Il s'agit d'un ordonnancement préemptif et possède les caractéristiques suivantes:

- Gestion des processus par durée d'exécution
- Utilisation d'un quantum de temps
  - Nécessite un timer
- Basé sur FCFS

3.3.1. Version canonique

Dans la version classique de l'algorithme Round Robin (RR), le quantum de temps est constant et ne varie pas. Si ce quantum est trop court, cela entraînera un nombre élevé de changements de contexte, ce qui risque de saturer le système, car le processeur passera tout son temps à exécuter du code noyau. À l'inverse, un quantum trop long rendra la politique RR similaire à une politique FCFS, réduisant ainsi son efficacité. La valeur optimale du quantum est déterminée de manière empirique, de façon à ce qu'environ 80 % des burst se terminent dans ce délai, tandis que 20 % entraînent une préemption. En général, les valeurs courantes pour le quantum se situent entre 10 et 200 ms.

4. Ordonnancement par priorité

Les processus dans l'état **READY** sont exécutés en fonction de leur priorité.

Il s'agit d'un ordonnancement pouvant être préemptif ou non préemptif et possède les caractéristiques suivantes:

- Gestion des processus par priorité
- Priorité statique ou dynamique
- À priorité égale, on utilisera une stratégie de type **FIFO**

4.1. Complément du cours

*Différentes variantes existes: la priorité peut être attribué à la création du processus et rester constante tout au long de son existence. Cette approche peut conduire à une **famine** des processus à basse priorité. C'est pourquoi une augmentation graduelle de la priorité d'un processus inactif depuis un certain temps peut grandement améliorer les temps de réponses. Lorsque le processus a "réussi" à s'exécuter, il retrouvera sa priorité initiale.*

5. Ordonnancement par files d'attentes multiples

Les processus sont regroupés en files d'attentes en fonction de leur priorité. Les files d'attentes sont ensuite ordonnancées en fonction de leur priorité.

5.1. Principe Général

- **Basé sur l'ordonnancement par priorité** : Les processus sont placés dans des files d'attente en fonction de catégories spécifiques.
- **Chaque file a une politique d'ordonnancement propre** :
  - File haute priorité : Round Robin (RR).
  - File basse priorité : First-Come, First-Served (FCFS).
- Les catégories de priorités possibles :
  - **Very High Priority**
  - **High Priority**
  - **Medium Priority**
  - **Low Priority**
  - **Very Low Priority**

5.1.1. Fonctionnement

1. **Classement des processus** : Les processus sont assignés à une file d'attente en fonction de leur catégorie.
2. **Priorité des files** : L'ordonnancement s'effectue en choisissant d'abord la file ayant la priorité la plus élevée.
3. **Application des politiques internes** : Une fois la file sélectionnée, l'ordonnanceur applique la politique spécifique à cette file.

5.1.2. Limites

- La priorité est liée à la **file**, et non au **processus**.
  - Un processus à très basse priorité peut se retrouver dans une file à très haute priorité.
- **Problème de catégorisation** :
  - Il est difficile de classer les processus (par exemple : interactif, batch, temps réel).
  - Le noyau ne dispose pas toujours des informations nécessaires pour effectuer ce classement.
- Solution : Une politique d'ordonnancement est attribuée à une file selon la **priorité de la file**, et non en fonction d'une catégorie de processus.

5.2. Ordonnancement par Files d'Attente Multiples avec Rétroaction

5.2.1. Principe

- **Priorité dynamique** :
  - Les processus peuvent **migrer entre les files** en fonction de leur état ou du temps écoulé depuis leur dernière exécution.
  - Un processus inactif depuis trop longtemps peut passer à une file de priorité supérieure.
  - Une fois exécuté, le processus retourne progressivement vers une file de priorité inférieure.

5.2.2. Exemple

- File 1 : RR avec un quantum de 8 ms.
- File 2 : RR avec un quantum de 16 ms.
- File 3 : FCFS.

5.2.3. Avantage

- **Évite la famine** :
  - Les processus à faible priorité finissent par être exécutés grâce à l'augmentation progressive de leur priorité.

5.2.4. Fonctionnement Étape par Étape

1. Les processus non exécutés depuis un certain temps **montent en priorité** en migrant vers des files supérieures.
2. Une fois exécutés, ils redescendent graduellement vers leur file d'origine.