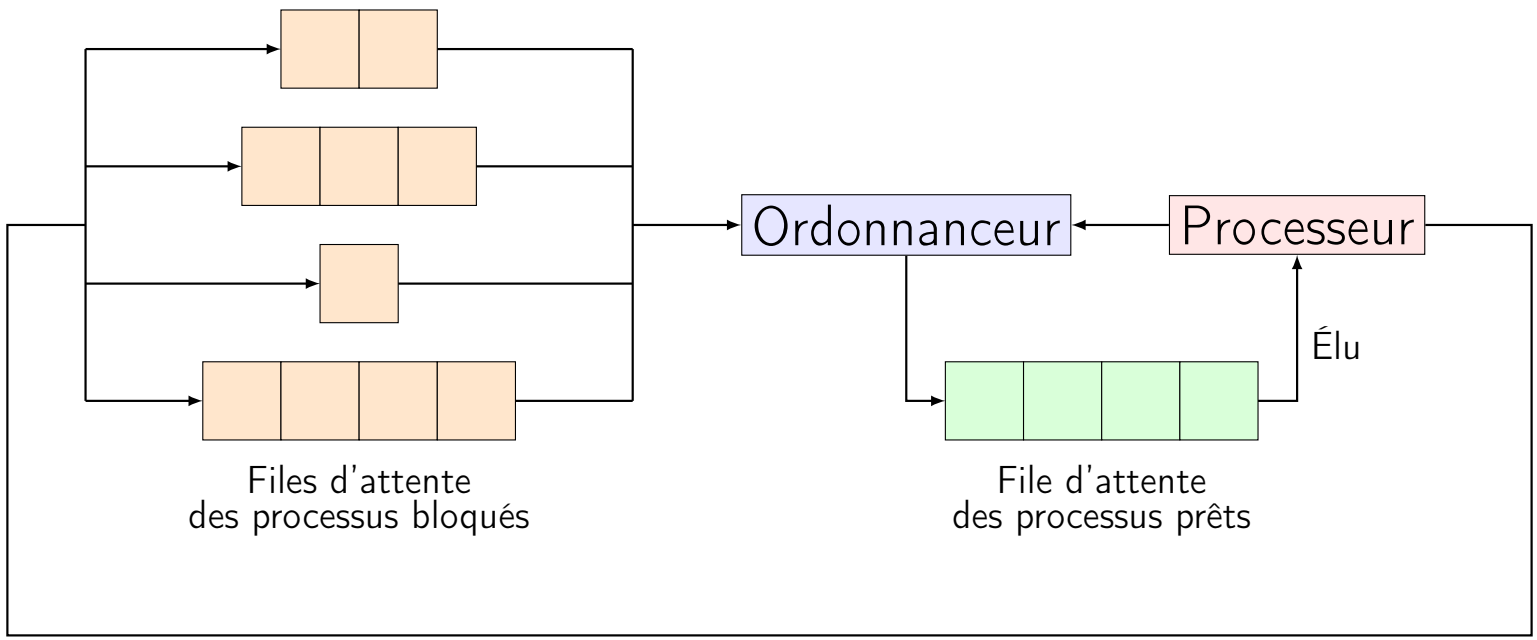
Programmation système

Ordonnancement

loig.jezequel@univ-nantes.fr

Ordonnancement, schéma général

L'ordonnancement consiste à choisir le processus/la tâche à exécuter à chaque instant et déterminer le temps durant lequel le processeur lui sera alloué.



L'ordonnanceur alloue le processeur aux différents processus selon un algorithme d'ordonnancement donné.

Objectifs de l'ordonnancement

Approche naïve

On pourrait juste exécuter les processus quand ils arrivent (premier arrivé premier servi)

Objectifs de l'ordonnancement

- Minimiser le nombre de changements de contexte
- Maximiser le nombre de processus exécutés par unité de temps
- Minimiser le temps d'attente de chaque processus
- Maximiser le temps d'utilisation des processeurs
- Favoriser les processus prioritaire

Types d'algorithmes d'ordonnancement

- Monoprocesseur/multiprocesseur
  - ordonnancer les processus sur un ou plusieurs processeurs ne se fait pas de la même façon
- En ligne/Hors ligne
  - les processus peuvent être ordonnancés une fois pour toute ou bien au fur et à mesure de leur création
- Préemptif/Non préemptif
  - on peut parfois interrompre un processus en cours à la demande de l'ordonnanceur
- Oisif/Non oisif
  - un processeur peut parfois être laissé inactif alors que des processus sont en attente d'être exécutés
- Centralisé/Réparti

Méthodes classiques d'ordonnancement

Non préemptif

- Selon l'ordre d'arrivée
  (First Come First Served, FCFS)
- Selon la durée de calcul
  (Shortest Job First, SJF)

Préemptif

- Selon la durée de calcul restante
  (Shortest Remaining Time, SRT)
- Temps partagé avec politique du tourniquet
  (Round-Robin, RR)
- Selon la priorité des tâches

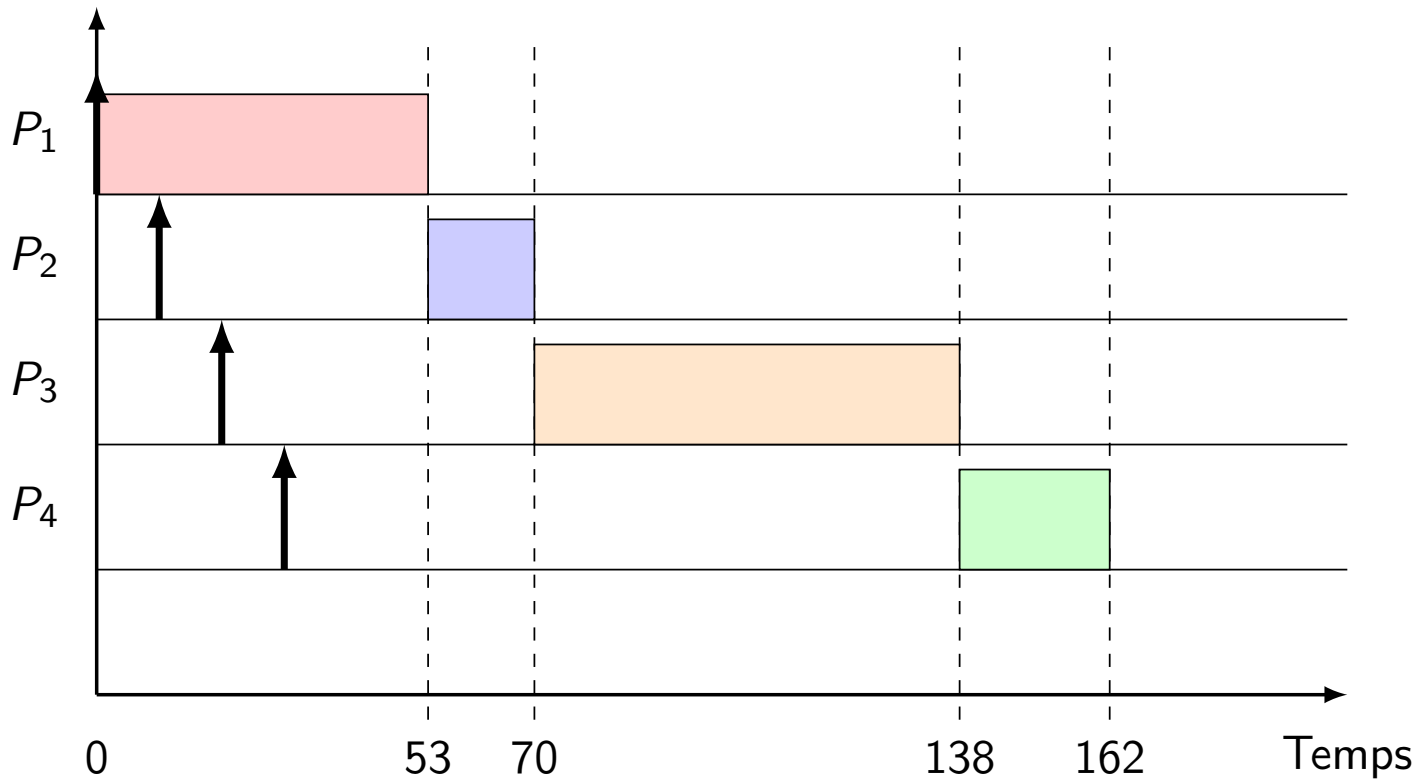
Ordonnancement First Come First Served

Principe

Les tâches sont ordonnancées selon leur ordre d'arrivée

Exemple

P_1 (arrive à $t = 0$, durée 53), P_2 (à $t = 10$, durée 17), P_3 (à $t = 20$, durée 68), P_4 (à $t = 30$, durée 24)



Ordonnancement First Come First Served, suite

Avantages

- Faible complexité d'implantation (simplement basé sur une file)

Inconvénients

- Pas de prise en compte de l'importance relative des tâches
- Temps d'attente du processeur par les tâches généralement important

Ordonnancement Shortest Job First

Principe

La tâche dont le temps d'exécution est le plus court est ordonnancée en priorité

Exemple

P_1 (arrive à $t = 0$, durée 53), P_2 (à $t = 0$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24)

Représentez l'histogramme de l'exécution des tâches par le processeur

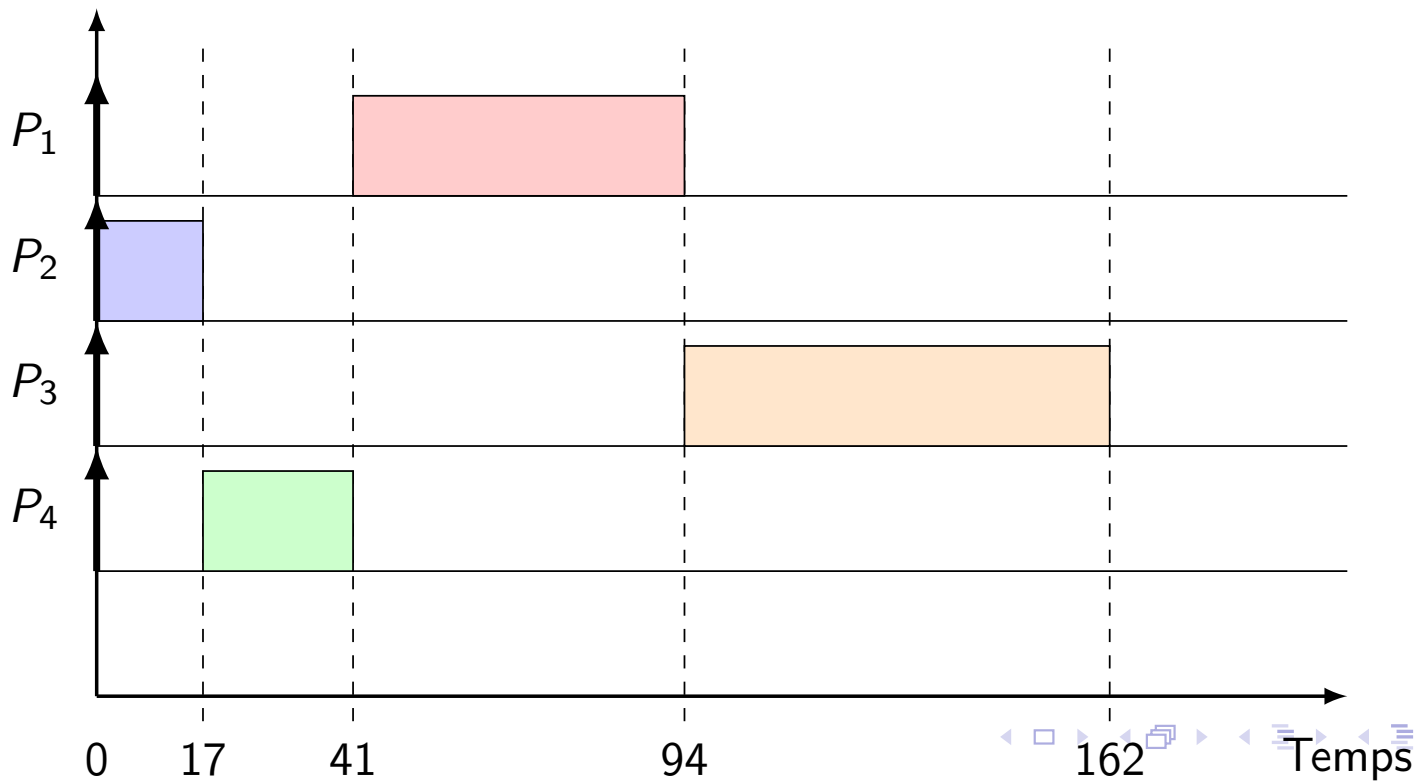
Ordonnancement Shortest Job First

Principe

La tâche dont le temps d'exécution est le plus court est ordonnancée en priorité

Exemple

P_1 (arrive à $t = 0$, durée 53), P_2 (à $t = 0$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24)



Ordonnancement Shortest Job First, suite

Avantages

- Réduit le temps d'attente des processus (par rapport à FCFS)

Inconvénients

- Pas de prise en compte de l'importance relative des tâches
- Optimal seulement si tous les processus sont disponibles simultanément

Ordonnancement Shortest Remaining Time

Principe

La tâche dont le temps d'exécution restant est le plus court parmi celles qui restent à exécuter est ordonnancée en premier

Exemple

P_1 (arrive à $t = 50$, durée 53), P_2 (à $t = 20$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24)

Représentez l'histogramme de l'exécution des tâches par le processeur

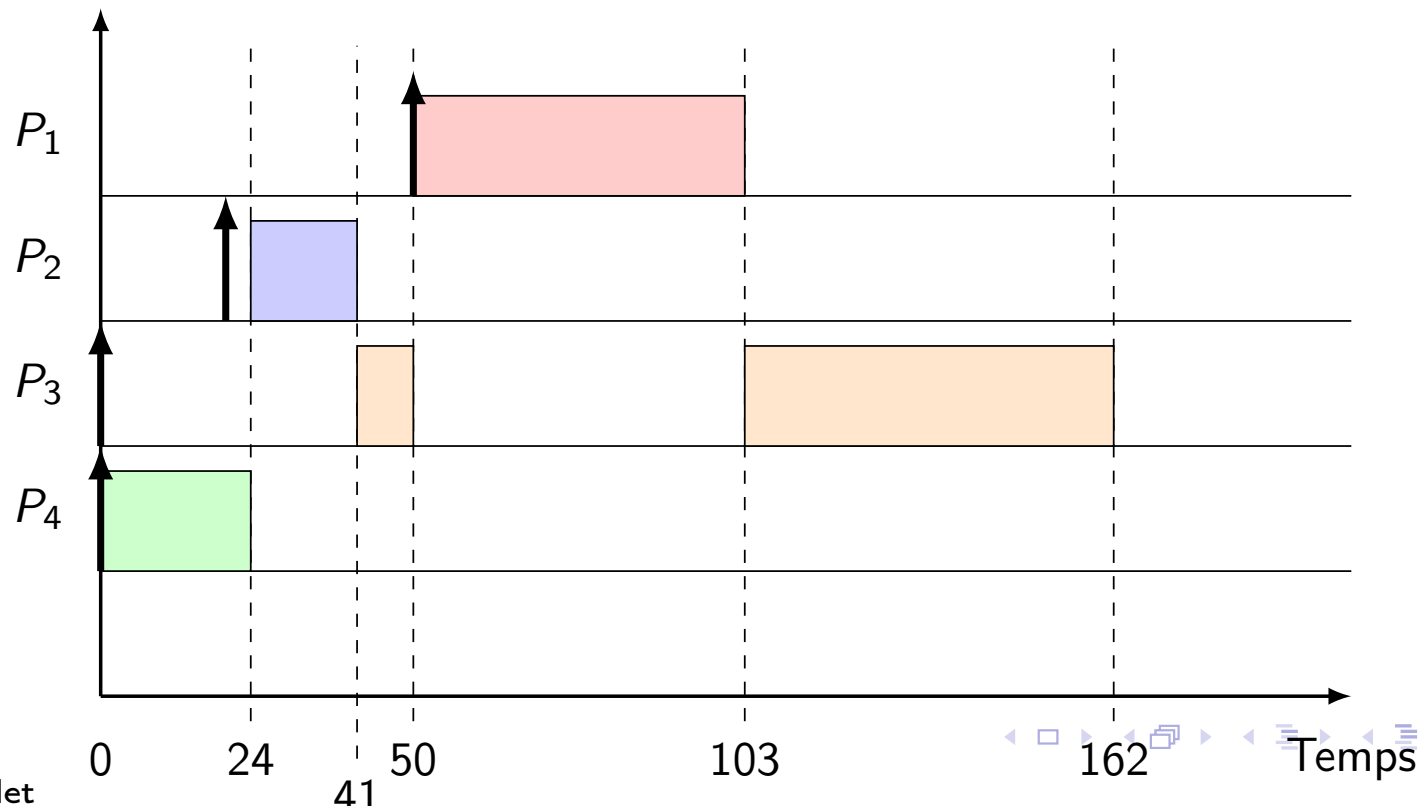
Ordonnancement Shortest Remaining Time

Principe

La tâche dont le temps d'exécution restant est le plus court parmi celles qui restent à exécuter est ordonnancée en premier

Exemple

P_1 (arrive à $t = 50$, durée 53), P_2 (à $t = 20$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24)



Ordonnancement Shortest Remaining Time, suite

Avantages

- Minimise le temps d'attente moyen des processus courts

Inconvénients

- Pas de prise en compte de l'importance relative des tâches
- Non équité de service : les processus longs sont pénalisés
- Possibilité de famine pour les processus les plus longs

Ordonnancement Round Robin

Principe

Le processeur est alloué par tranches de temps de taille fixe (quantum), chacun son tour.

Exemple

P_1 (arrive à $t = 0$, durée 53), P_2 (à $t = 0$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24), le quantum vaut 20

Représentez l'histogramme de l'exécution des tâches par le processeur

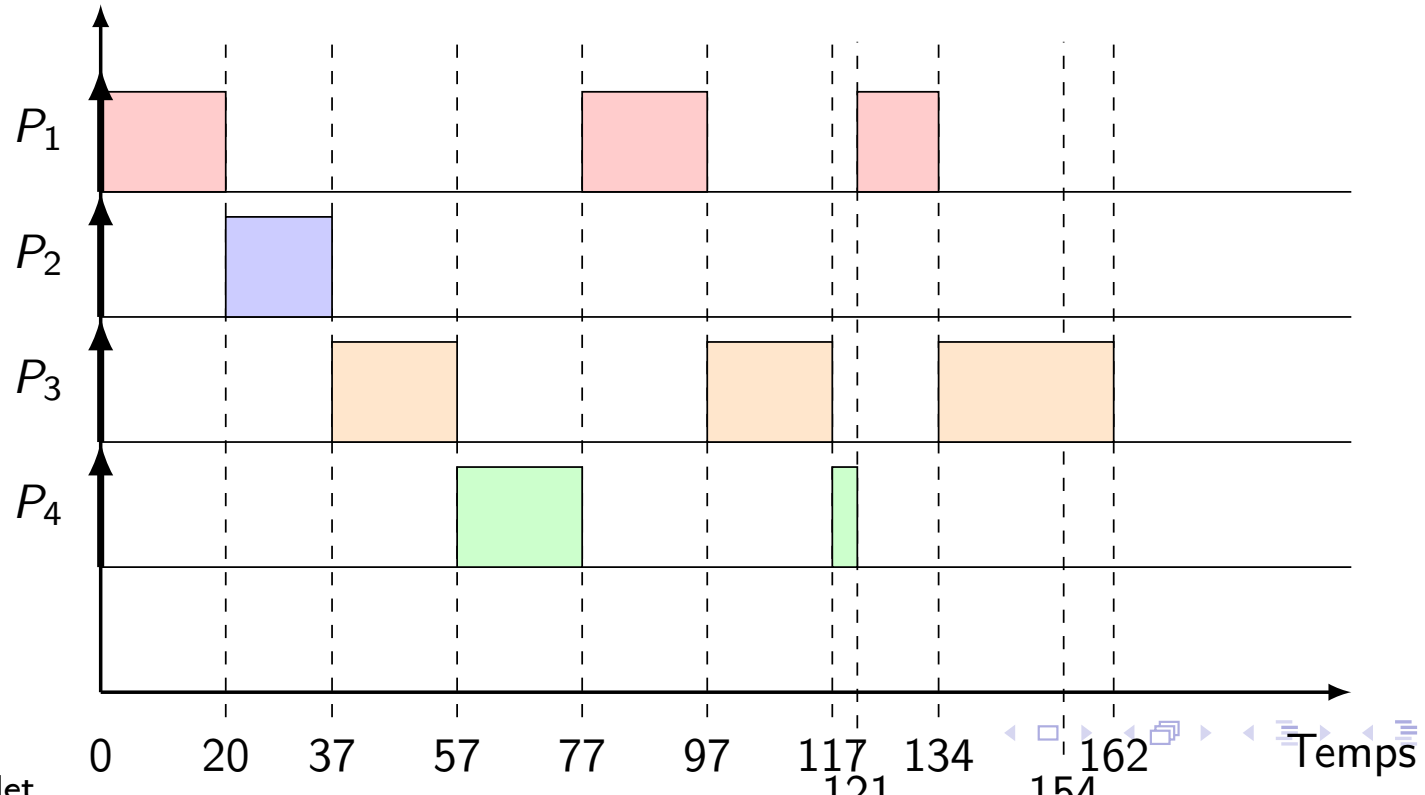
Ordonnancement Round Robin

Principe

Le processeur est alloué par tranches de temps de taille fixe (quantum), chacun son tour.

Exemple

P_1 (arrive à $t = 0$, durée 53), P_2 (à $t = 0$, durée 17), P_3 (à $t = 0$, durée 68), P_4 (à $t = 0$, durée 24), le quantum vaut 20



Ordonnancement Round Robin, suite

Avantages

- Équité de l'attribution du processeur entre toutes les tâches
- Avec n tâches, chacune obtient le processeur au bout de $(n-1) \times q$ unités de temps au maximum

Inconvénients

- Pas de prise en compte de l'importance relative des tâches
- Difficulté du choix de la tranche de temps
  - trop grande, Round Robin équivalent à FCFS
  - trop petite, beaucoup de changements de contexte

Ordonnancement à priorités statiques

Principe

Le processeur est alloué selon des priorités affectées aux tâches pour toute la durée de vie de l'application

Exemple

P_1 (à $t = 0$, $d = 53$, priorité 0), P_2 (à $t = 0$, $d = 17$, priorité 1),
P_3 (à $t = 0$, $d = 68$, priorité 3), P_4 (à $t = 0$, $d = 24$, priorité 2)

Représentez l'histogramme de l'exécution des tâches par le processeur

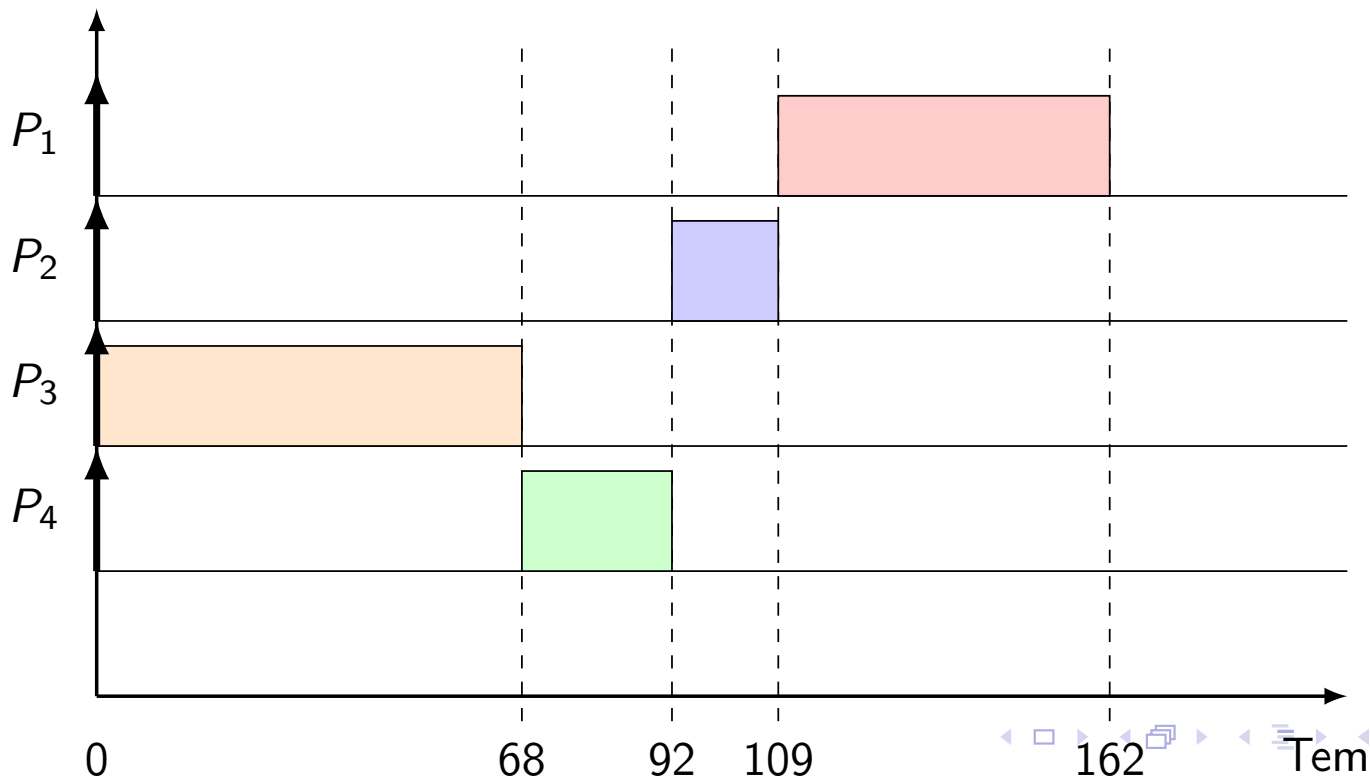
Ordonnancement à priorités statiques

Principe

Le processeur est alloué selon des priorités affectées aux tâches pour toute la durée de vie de l'application

Exemple

P_1 (à $t = 0$, $d = 53$, priorité 0), P_2 (à $t = 0$, $d = 17$, priorité 1), P_3 (à $t = 0$, $d = 68$, priorité 3), P_4 (à $t = 0$, $d = 24$, priorité 2)



Ordonnancement à priorités statiques, suite

Avantages

- Prise en compte de l'importance relative des tâches

Inconvénients

- Risque de famine pour les tâches de moindre priorité