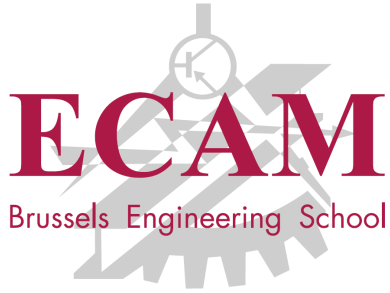


I403A Systèmes d'exploitation

Séance 3

Ordonnancement de processus

Sébastien Combéfis

2018–2019

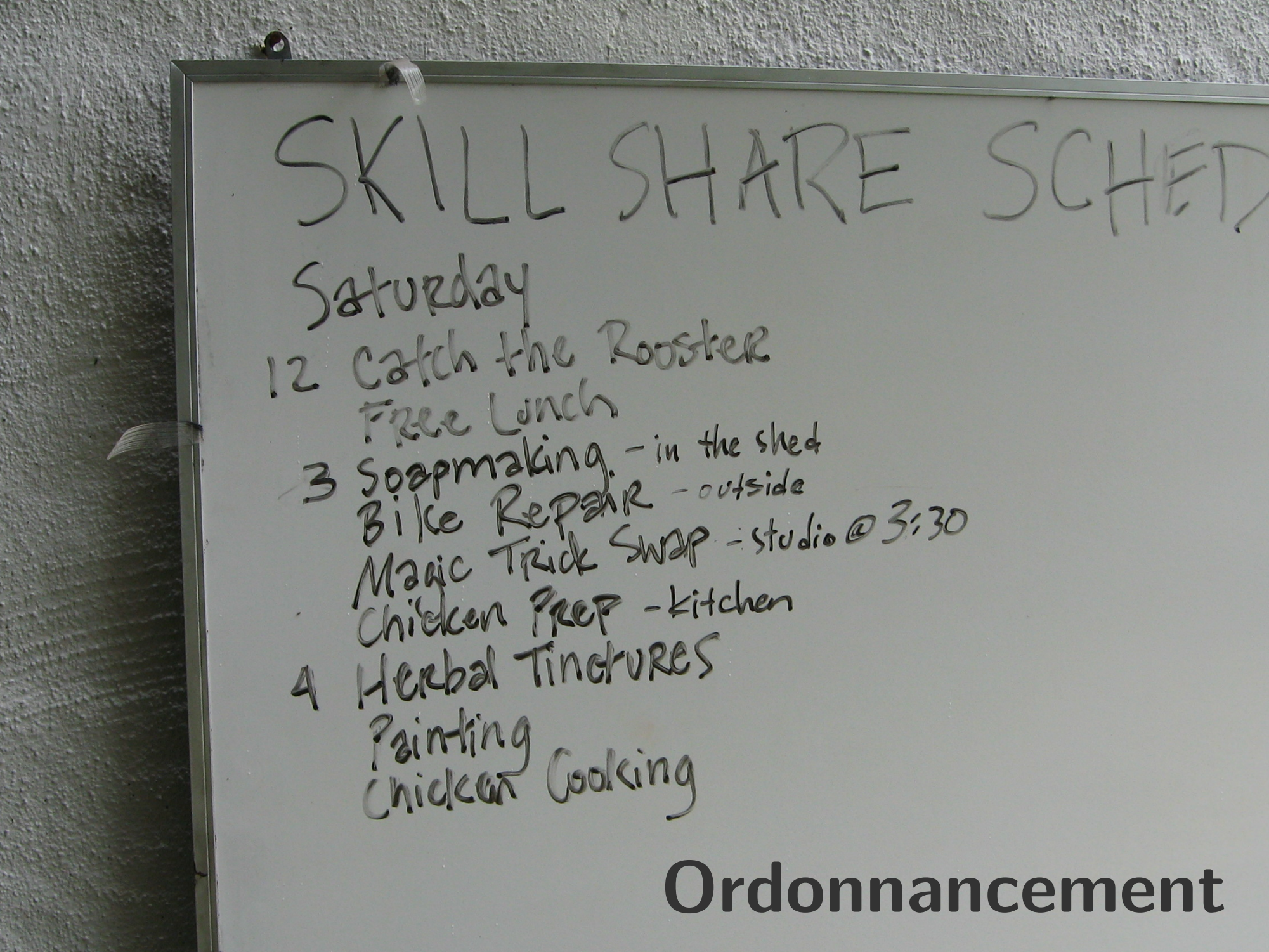
Rappels

- Définition des notions de processus et de thread
  - Création, exécution et terminaison par appels systèmes
  - Structure en mémoire conservées par l'OS
- Modélisation de la gestion des tâches par l'OS
  - Vie et différents états des processus
  - Différents modèles de multi-threading

Objectifs

- Décrire et comprendre le rôle de l'ordonnanceur de tâches
  - Principes, salve CPU et dispatcher
  - Algorithmes et critères d'ordonnancement
- Exemples d'ordonnanceurs courants

  Les cas de Solaris, Windows et Linux



Ordonnancement

Multiprogrammation

- La multiprogrammation permet d'exploiter au mieux le CPU

  Le CPU est partagé entre plusieurs processus

- Un processus est exécuté jusqu'à ce qu'il doive attendre

  Par exemple pour une opération d'E/S

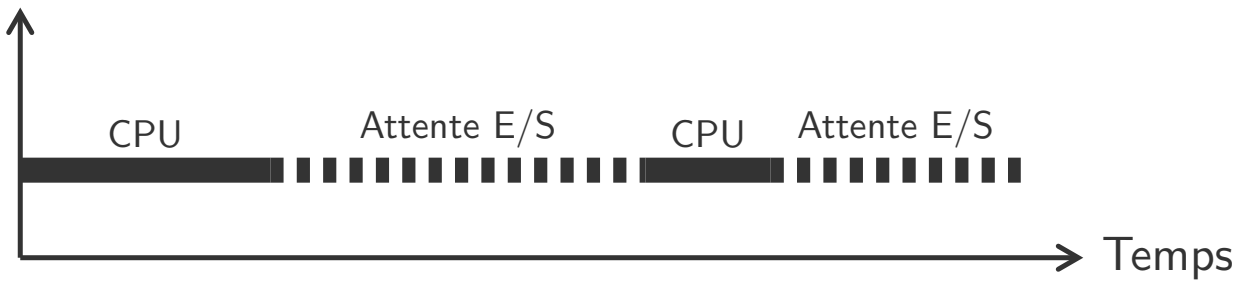
- Plusieurs processus sont maintenus en mémoire

  L'ordonnanceur alterne ces processus sur le CPU

Cycle de salves CPU–E/S

- La vie d'un processus est un cycle entre CPU et attente d'E/S

  Alternance entre salves CPU et salves E/S



- Beaucoup de salves CPU courtes, et peu de longues

  Sur base de nombreuses mesures effectuées

Ordonnanceur CPU

- Choix d'un processus dans la ready queue pour le CPU

  On prend un processus qui est prêt à directement démarrer

- Choix effectué par l'ordonnanceur court terme

  Parmi tous les processus qui sont déjà en mémoire

- L'ordonnanceur prend une décision lorsqu'un processus...

  1. ...passe de *Running* à *Waiting* (requête E/S, appel `wait`...)
  2. ...passe de *Running* à *Ready* (interruption...)
  3. ...passe de *Waiting* à *Ready* (réponse E/S...)
  4. ...se termine

Préemption

- Ordonnanceur non-préemptif (seulement 1 et 4)
  - Aussi appelé coopératif
  - Aucun choix en terme d'ordonnancement, il faut un nouveau
  - Un processus garde le CPU jusqu'à ce qu'il le relâche
- Ordonnanceur préemptif
  - Nécessite un timer hardware
  - Un processus peut être retiré à tout moment du CPU

Problèmes de la préemption

- Apparition de conditions de course (*race condition*)
  - Lorsque plusieurs processus partagent des données
  - Mise à jour partielle avant préemption provoque inconsistence
- Attention à porter au design du kernel
  - Kernel maintient des données pour chaque processus
  - Attention à la préemption pendant une mise à jour des données

Dispatcher

- Le dispatcher donne le contrôle du CPU à un processus
  1. Changement de contexte
  2. Basculer en mode utilisateur
  3. Aller à la bonne adresse en mémoire pour relancer le processus
- Cause un temps de latence lors de chaque dispatch

  Le dispatcher doit être le plus rapide possible

Critères d'ordonnancement (1)

- Utilisation du CPU
  - Pourcentage de temps où le CPU est occupé
  - De 40% pour charge légère à 90% pour charge lourde
- Débit de processus
  - Nombre de processus terminés par unité de temps
  - De $1/h$ pour longs processus à $10/s$ pour transactions courtes

Critères d'ordonnancement (2)

- Temps de rotation
  - Temps total écoulé pour l'exécution d'un processus
  - Chargement en mémoire, ready queue, exécution CPU, E/S
- Temps d'attente

  Somme des temps d'attente en ready queue
- Temps de réponse
  - Temps entre la soumission du processus et sa première réponse
  - La sortie commence à arriver, pendant que la suite est calculée

Optimisation des critères d'ordonnancement

- Plusieurs choix possibles pour l'optimisation de ces critères
  - Optimisation de la valeur moyenne des critères
  - Optimisation de la valeur minimale ou maximale
- Cas particulier selon système, par exemple pour interactifs

  Minimiser variance du temps de réponse, plutôt que moyenne

Critère	**Optimisation**
Utilisation CPU	Maximiser
Débit	Maximiser
Temps de rotation	Minimiser
Temps d'attente	Minimiser
Temps de réponse	Minimiser

Algorithmes d'ordonnancement

First-Come First-Served (FCFS) (1)

- Processus exécutés dans l'ordre d'arrivée (FIFO)

$\Rightarrow$ Temps d'attente P_1 : 0, P_2 : 24, P_3 : 27, temps d'attente moyen : **17**

$\Rightarrow$ Temps d'attente P_1 : 6, P_2 : 0, P_3 : 3, temps d'attente moyen : **3**

First-Come First-Served (FCFS) (2)

- Ordonnancement non-préemptif
  - Un processus long peut empêcher des petits de se terminer
  - Pas adapté à un système en temps partagé
- Temps d'attente moyen dépend du choix d'ordonnancement

  Des salves de longueurs différentes sont pénalisantes
- Induit un effet convoi
  - Un processus attaché-CPU, des petits processus attaché-E/S
  - Les petits processus se trouvent coincés derrière des gros

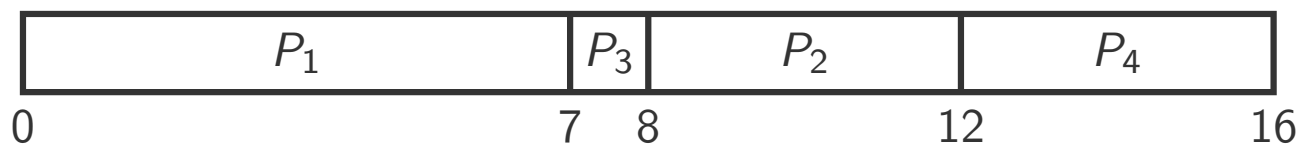
Shortest-Job-First (SJF) (1)

- Durée de la prochaine salve CPU la plus courte

  Prouvé optimal par rapport à la durée d'attente moyenne

- Utilisation de FCFS en cas d'égalité

Processus	**Instant d'arrivée**	**Durée de salve**
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



$\Rightarrow$ Temps d'attente $P_1 : 0$, $P_2 : 8 - 2$, $P_3 : 7 - 4$, $P_4 : 12 - 5$
Temps d'attente moyen : **4**

Shortest-Job-First (SJF) (2)

- Ordonnancement préemptif (Shortest-Remaining-Time-First)

  Quand un nouveau processus arrive, éventuel changement

Processus	**Instant d'arrivée**	**Durée de salve**
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

$\Rightarrow$ Temps d'attente $P_1 : 11 - 2$, $P_2 : 5 - 4$, $P_3 : 0$, $P_4 : 7 - 5$
Temps d'attente moyen : **3**

Détermination de la durée des salves (1)

- Possible pour ordonnancement long terme de job en batch

  Utilisation du temps limite spécifié par l'utilisateur

- Moyenne exponentielle des durées des salves précédentes
  - t_n durée réelle de la n^e salve
  - τ_{n+1} valeur prédite de la durée de la prochaine salve
  - Pour α tel que $0 \leq \alpha \leq 1$, on définit

$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

Détermination de la durée des salves (2)

- τ_n est une mesure de l'historique

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + ... + (1-\alpha)^j \alpha t_{n-j} + ... + (1-\alpha)^{n+1} \tau_0$$

- α définit dans quelle mesure l'historique est pris en compte
  - $\alpha = 0$: aucune influence historique récent ($\tau_{n+1} = \tau_n$)
  - $\alpha = 1$: aucune influence historique ($\tau_{n+1} = t_n$)
- Durée salve CPU similaire aux durées des précédentes salves

Priorité (1)

- Chaque processus se voit attribuer un numéro de priorité
  - Processus de plus haute priorité choisi en premier
  - Utilisation de FCFS lors d'égalités
- SJF est un cas particulier de priorité
  - La priorité est égale à 1 / durée salve CPU
  - Plus basse priorité pour les salves de durées les plus longues

Priorité (2)

- Peut être non-préemptif ou préemptif

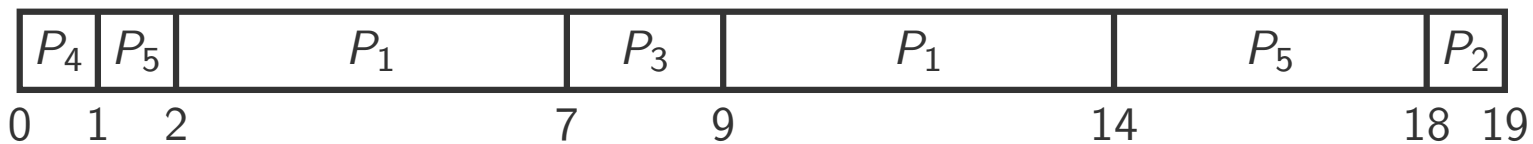
Processus	Instant d'arrivée	Durée de salve	Priorité
P_1	2	10	3
P_2	0	1	1
P_3	7	2	4
P_4	0	1	5
P_5	0	5	2

$\Rightarrow$ Temps d'attente $P_1 : 6 - 2$, $P_2 : 18$, $P_3 : 16 - 7$, $P_4 : 0$, $P_5 : 1$
Temps d'attente moyen : **6.4**

Priorité (3)

- Peut être non-préemptif ou préemptif

Processus	Instant d'arrivée	Durée de salve	Priorité
P_1	2	10	3
P_2	0	1	1
P_3	7	2	4
P_4	0	1	5
P_5	0	5	2



$\Rightarrow$ Temps d'attente $P_1 : 9 - 7$, $P_2 : 18$, $P_3 : 0$, $P_4 : 0$, $P_5 : 1 + (14 - 2)$
Temps d'attente moyen : **6.6**

Choix des priorités

- Définition des priorités de manière interne ou externe
  - En **interne**, priorité basée sur quantités mesurables

    Limite de temps, besoin en mémoire, nombre fichiers ouverts...
  - En **externe**, priorité basée sur critères en dehors de l'OS

    Importance du processus, paiements...
- Processus de faibles priorités ne passent jamais

  Vieillissement : augmenter la priorité avec le temps

Round-Robin (RR) (1)

- Petite unité de temps CPU (*time quantum* ou *time slice*)

  FCFS avec ajout de préemption, unités de 10 à 100 ms

- La ready queue est une file circulaire

  Si la queue contient n processus, et que le time quantum est q chaque processus reçoit $1/n$ temps par morceaux de q temps

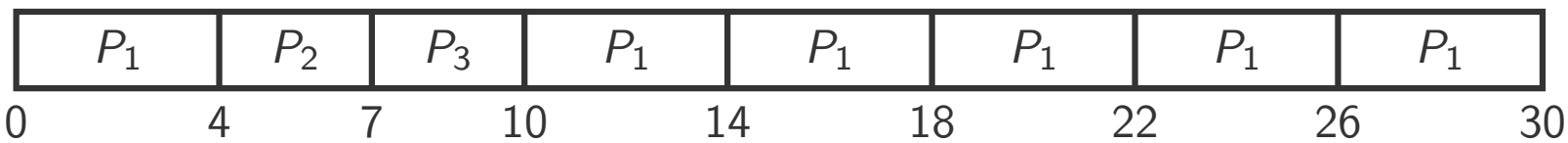
- Time quantum élevé $=$ FCFS

  Mais q doit être plus grand que temps de basculement ($\sim 10\mu s$)

Round-Robin (RR) (2)

- Exemple avec un time quantum de 4

Processus	**Durée de salve**
P_1	24
P_2	3
P_3	3



$\Rightarrow$ Temps d'attente $P_1 : 10 - 4$, $P_2 : 4$, $P_3 : 07$
Temps d'attente moyen : **5.6**

File multi-niveaux (1)

- Files multi-niveaux si les processus sont classés en catégories

  Processus avant-plan (interactif) / arrière-plan (batch)

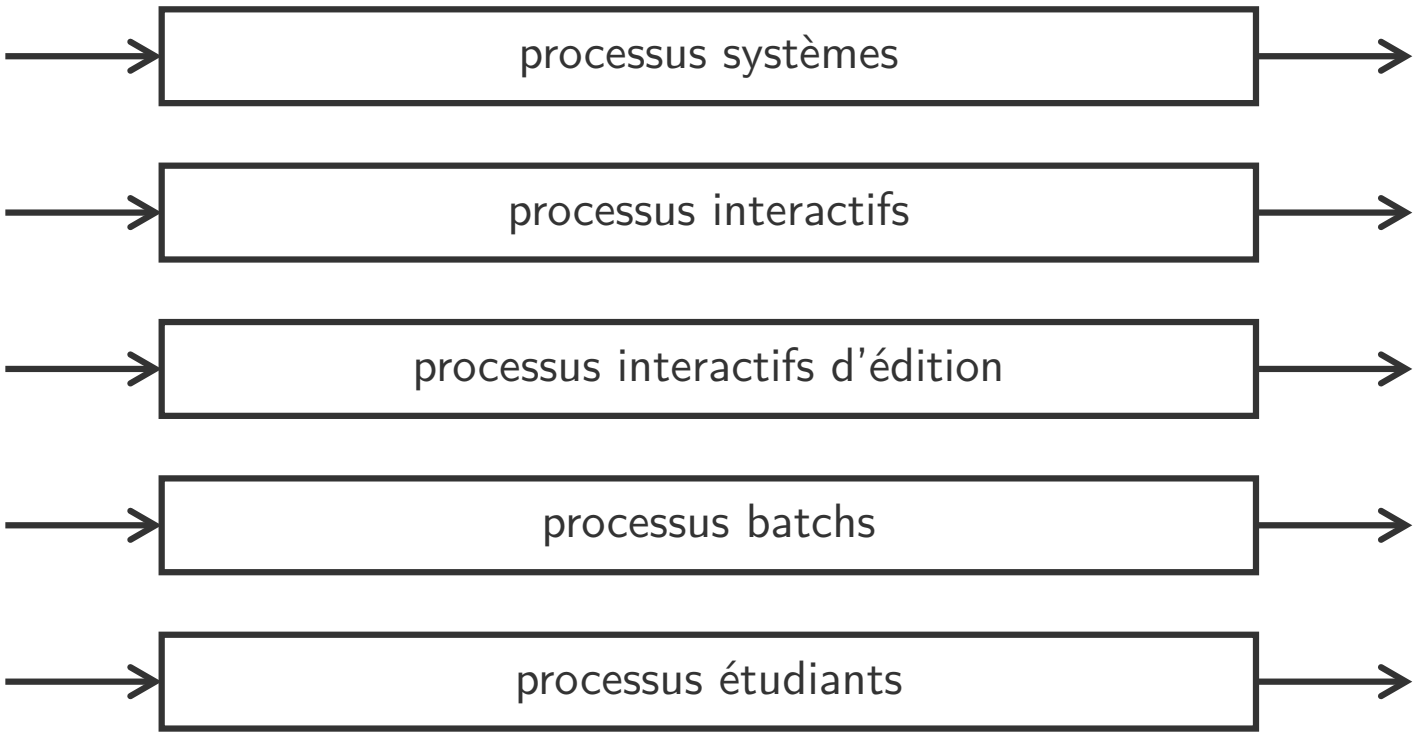
- Chaque file possède son propre algorithme d'ordonnancement

  Typiquement RR pour l'avant-plan et FCFS pour l'arrière-plan

- Algorithme d'ordonnancement entre les files
  - Priorité absolue fixée entre les files
  - Tranche de temps (*e.g. 80%/20% pour avant/arrière-plan*)

File multi-niveaux (2)

Priorité haute



Priorité basse

File multi-niveaux avec rétroaction (1)

- Changement de file selon durée salves CPU
  - Vers **priorité basse** si trop d'utilisation du CPU

    Priorité aux processus interactif et attaché-E/S
  - Vers **priorité haute** si trop long temps d'attente

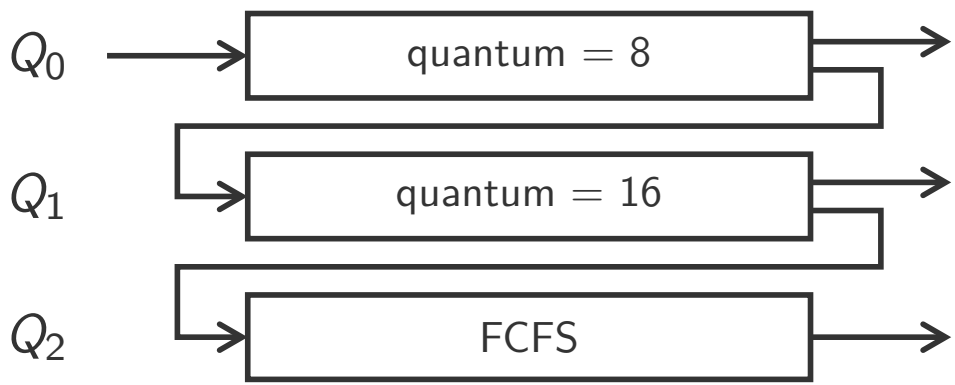
    Genre de vieillissement pour empêcher dégénérescence
- Plusieurs paramètres possibles
  - Le nombre total de files
  - L'algorithme d'ordonnancement de chaque file
  - Règle qui {promeut/rétrograde/file initiale} d'un processus

File multi-niveaux avec rétroaction (2)

- Exemple avec trois files

  RR avec $q = 8$ *et* $q = 16$ *pour* Q_0 *et* Q_1 *et FCFS pour* Q_2

- Choix de trois règles pour les processus
  - Nouveau processus rentre en Q_0
  - Transit des processus $Q_0 \rightarrow Q_1 \rightarrow Q_2$
  - Processus qui rentre en Q_i préempte processus de Q_{i-1}



Ordonnancement des threads

- Différence entre threads utilisateurs et threads noyaux
  - Threads noyaux ordonnancés par l'OS
  - Threads utilisateurs ordonnancés par la librairie de threads
- Pour les modèles plusieurs-vers-un et plusieurs-vers-plusieurs
  - Process-Contention Scope (PCS)

    Compétition pour CPU dans les processus
  - System-Contention Scope (SCS)

    Compétition entre tous les threads du système

Exemples d'OS courants

Solaris (1)

- Ordonnancement par priorités avec six classes différentes
  - Temps partagé (TS) (classe par défaut)
  - Interactif (IA)
  - Temps réels (RT)
  - Système (SYS)
  - Juste parts (FSS)
  - Priorité fixée (FP)

Solaris (2)

- Classe Temps partagé (TS) par défaut
  - File multi-niveaux avec rétroaction
  - Relation inverse : priorité $\downarrow$ et time slice $\uparrow$
  - Haute priorité aux tâches interactives, basse aux attaché-CPU
- Classe Interactif (IA) comme TS mais avec plus haute priorité
- Plus haute priorité pour les Temps-réel (RT)

  Exécution directe d'un processus présent dans cette classe

Solaris (3)

- Classe Système (SYS) pour les threads kernel
  - Démon en charge de l'ordonnancement, pagination
  - Exclusivement pour l'utilisation du kernel
- Priorité fixe (FP) et juste parts (FSS) depuis Solaris 9
  - Threads dans FP comme TS mais sans priorité dynamique
  - FSS se base sur des parts de CPU à partager

Solaris (4)

- Calcul d'une priorité globale basée sur priorités dans les classes

  Ordonnancement de type RR en cas d'égalité

- Thread sélectionné s'exécute jusqu'à l'un des trois conditions

  1. Le thread bloque
  2. Il a consommé toute sa time slice
  3. Il a été préempté par un thread de priorité plus haute

- Présence de 10 threads de gestion des interruptions

  Ordonnancement : interruption $>$ *RT* $>$ *SYS* $>$ *FSS, FX, TS, IA*

Windows (1)

- Ordonnancement basé sur les priorités et préemptif
  - Le thread avec la plus haute priorité sera celui exécuté
  - Exécution jusque terminaison, fin quantum, appel bloquant
  - Préemption par thread RT plus haute priorité
- Découpe en 32 niveaux de priorité coupés en deux classes
  - Variable (1 à 15) et temps réel (16 à 31)
  - Un thread au niveau 0 gère la mémoire
- Une file par niveau de priorité

  Priorité relative pour les thread au sein de leur classe

Windows (2)

- Priorité dans la classe variable peut changer
  - Priorité **diminuée** lorsque temps d'un thread écoulé
  - Priorité **augmentée** lorsque attente E/S finie
- Boost ou ralentissement dynamique d'un processus

  La valeur reste supérieure à la priorité de base initiale
- Trois fois plus de temps pour le processus d'avant-plan

  Augmentation de l'interactivité, avec préemption time-sharing

Linux (1)

- Variante de l'ordonnanceur de Unix (avant kernel 2.5)
  - Pas de support des multi-processeurs
  - Mauvaises performances avec beaucoup de processus
- Ordonnanceur $O(1)$ apparu dans le noyau 2.5
  - Exécution en temps constant (par rapport nombre de tâches)
  - Paramètre d'affinité des processeurs et load balancing (SMP)
  - Mauvais temps de réponse pour tâches interactives

Linux (2)

- Completely Fair Scheduler (CFS) dans noyau 2.6.23
  - Classes d'ordonnancement avec une priorité pour chacune
  - Une classe temps réel et une par défaut avec CFS
- Affectation d'une proportion de temps CPU par tâche
  - Valeur du **nice** par tâche (de -20 à +19)
  - Petite valeur indique priorité relative plus grande
  - Valeur 0 par défaut, l'augmenter signifie qu'on est gentil

Linux (3)

- Utilisation d'un temps d'exécution virtuel
  - Facteur de désintégration basé sur la priorité de la tâche
    - Pour *nice* $= 0$, temps d'exécution virtuel $=$ physique
    - Priorité plus basse (haute), temps virtuel plus grand (petit)
  - Choix de la tâche avec le plus petit `vruntime`
- Utilisation d'un Red-Black Tree pour faire le choix

  Complexité en $\mathcal{O}(\log n)$*, mais cache du RB-leftmost*

Crédits

- https://www.flickr.com/photos/forresto/488853222
- https://www.flickr.com/photos/arenagroove/1390730781
- https://www.flickr.com/photos/charliedees/2757271354