Chapitre III : La gestion du processeur
Mécanismes d'ordonnancement

Eric.Leclercq@u-bourgogne.fr

Département IEM
http://ufrsciencestech.u-bourgogne.fr
http://ludique.u-bourgogne.fr/~leclercq

Janvier 2017

Plan

Introduction et problématique

- De nombreux processus sont gérés par le SE
- L'efficacité théorique maximale est atteinte si, à un instant t le nombre de processeurs est identique à celui des processus à exécuter (c'est irréaliste).
- Dans la plupart des cas la machine possède 1 seul processeur (soit quelques cœurs) par conséquent : un grand nombre de processus se partagent le processeur (une ressource limitée).
- Il faut définir :
  - un mécanisme qui permet au SE de garder le contrôle des exécutions
  - une politique d'accès au processeur (ordonnancement ou *scheduling*) : l'utilisation du processeur est divisée en tranches (slices) affectées au processus sous la forme de quantum.

Problématique

Le SE ne se contente pas de réaliser une abstraction du matériel mais propose une véritable machine virtuelle aux application (cf ch1). Par conséquent :

- il est nécessaire d'empêcher les applications d'accéder aux ressources sans passer par le SE
- certaines instructions doivent avoir des comportement différents selon celui qui les utilisent (SE ou applications utilisateur)

Tout comme le SE, le processeur possède plusieurs modes d'exécution : superviseur / noyau et utilisateur

Modes d'exécution du processeur

- en mode **superviseur** (correspondant au mode noyau du SE) : toutes les instructions sont autorisées et seul le SE peut les exécuter
- en mode **utilisateur** : certaines instruction sont interdites ou limitées, les applications n'ont accès qu'a ce mode

Exemple :

mode noyau pour la capture des informations sur le réseau, mode utilisateur pour limiter les plages d'adresses mémoire accessibles par les processus

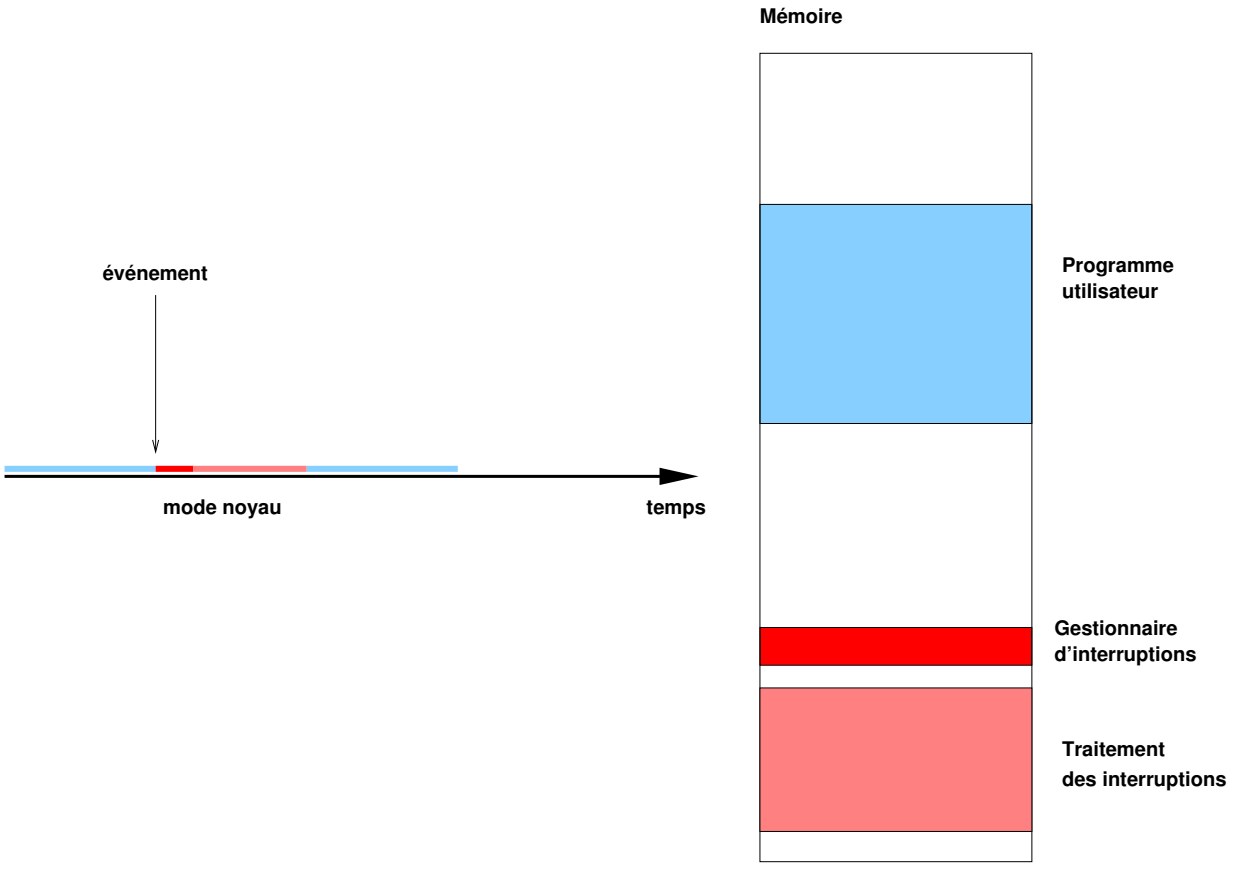
Il existe un autre mode : le mode hyperviseur utilisé pour implanter la virtualisation de certain SE

Notions d'interruption

En cas d'erreur ou d'événement déclenché par le matériel (transfert, timer, etc.) l'exécution du processus est suspendue et le pointeur d'instruction est positionné sur un programme particulier : le gestionnaire d'interruptions

- le gestionnaire d'interruption (*interrupt handler* se trouve à une position mémoire spécifique
- en fonction de l'événement ou de l'erreur le gestionnaire d'interruption lance l'exécution du code pour traiter l'événement
- la programmation du gestionnaire d'interruption (son paramétrage) n'est possible qu'en mode superviseur

Notions d'interruption



Le noyau d'un système d'exploitation comporte un ensemble de routines pour gérer les interruptions (chargées au démarrage du système)

Appel système

Lorsqu'un processus s'exécute (en mode utilisateur) et qu'il souhaite accéder à une ressource (disque par exemple) ou invoquer une fonction du SE, il procède à un **appel système** (*system call*) :

- appel système
- mise en file pour l'accès à la ressource
- passage en mode noyau pour la réalisation de l'appel
- notification par interruption matérielle

États d'un processus

Lorsqu'un processus s'exécute, il change d'état :

- **en exécution :** les instructions sont en cours d'exécution dans le CPU ;
- **en attente :** les processus attend un événement (notification suite à la demande d'une ressource ou d'une opération d'E/S) ;
- **prêt :** le processus attend d'être affecté au processeur

Un seul processus peut être affecté à un processeur (cœur) à un instant donné. Cependant plusieurs processus peuvent être prêt et mis en attente.

Cas du noyau Linux

Le noyau 2.4 de Linux ne garanti pas un traitement rapide des interruptions matérielles c'est-à-dire le lancement de la tâche associée (plusieurs centaines de ms).

Le noyau 2.6 ajouter la notion de préemption du noyau lui même :

- en 2.4 aucun ré-ordonnancement des tâches n'était possible tant que le noyau n'avait pas terminé le traitement associé à un appel système
- donc impossible de prédire quand une tâche de haute priorité pouvait s'exécuter
- le noyau 2.6 introduit la notion de point de préemption (*hooks*) qui permettent au noyau de s'interrompre pour exécuter une tâche

Notion d'ordonnanceur

Définition : (Ordonnanceur)

Le mécanisme d'ordonnancement définit les critères selon lesquels les processus ont accès au processeur

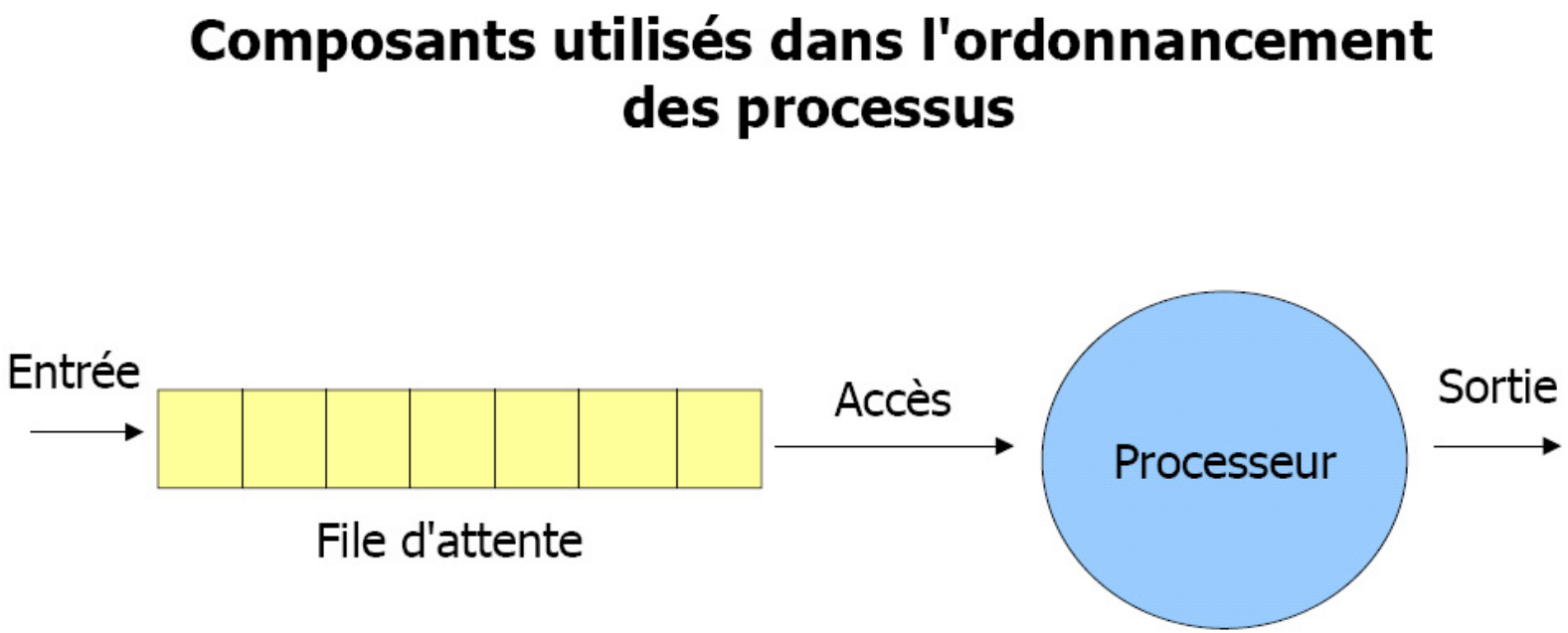
Les demandes d'accès au processeurs sont gérées dans des files d'attente.

Les différentes méthodes de gestion des files vont donner lieu à des stratégies d'ordonnancement différentes et des comportement du SE différents.

On souhaite :

- un temps de réponse rapide pour les applications interactives
- un débit élevé pour les travaux en arrière plan
- éviter la famine

Composants de l'ordonnanceur



Objectif : réaliser un implémentation qui minimise le temps d'exécution de la stratégie.

Principes d'ordonnancement

On distingue 3 grands principes de gestion des accès au processeur selon :

- l'ordre d'arrivée : premier arrivé premier servi ;
- le degré d'urgence : le premier servi est celui dont le besoin d'accès rapide à la ressource est le plus grand ;
- l'importance : le premier servi est celui dont l'accès à la ressource est le plus important.

Suivant le principe retenu le SE aura un comportement différent : il ne sera pas destiné a tous les types d'utilisations (par ex. adapté aux traitements temps réel).

Pour s'adapter à des cas particuliers, plusieurs principes sont souvent combinés

Notion de pénalisation

Définition : (Pénalisation)

Lorsqu'un processus ne peut pas accéder directement à une ressource qu'il convoite on dit qu'il est pénalisé

La pénalisation que subit un processus peut être représentée par son temps d'attente

Définition : (Temps d'attente)

Le temps d'attente est le nombre d'unité de temps durant lesquelles le processus est présent dans la file d'attente (sans être exécuté)

Notion de pénalisation

La mesure de la pénalité peut être affinée :

- en relativisant le temps d'attente par rapport à la durée du processus
- on obtient ainsi un taux de retard T défini par le rapport :

$$T = \frac{d}{a + d}$$

  - d est durée du processus
  - a durée d'attente (cumulée)
  - $a + d$ le temps total du processus passé dans le système
- dans le cas idéal $T = 1$

Paramètres essentiels

Deux paramètres peuvent être pris en compte pour élaborer une stratégie d'ordonnancement :

- Stratégie d'accès
- Utilisation de processeur

Traitement jusqu'à terminaison

Principe : la politique de traitement du processus jusqu'à terminaison

- accorde le processeur à un processus.
- ne l'interrompt jamais quelque soit sa durée ;

Stratégie FCFS (*First Come First Serve*) c-à-d premier arrivé premier servi

- elle correspond à une gestion FIFO (First In, First Out) de la file d'attente
- c'est la stratégie plus simple
- le traitement des processus est séquentiel
- les processus courts sont pénalisés et le temps d'attente moyen est souvent important
- problème avec les processus *IO-bound* et *CPU-bound* provoque un effet *convoy*

Traitement jusqu'à terminaison

Stratégie SFJ (*Shorstest Job First*) c-à-d moindre durée :

- on garde le principe d'occupation du processeur jusqu'à terminaison
- la file d'attente est ordonnée non plus de façon chronologique mais en fonction du temps d'exécution nécessaire (on fait passer en tête les travaux courts)
- cependant la durée totale est *a-priori* inconnue, on utiliser une décomposition en rafales (*burst*) CPU et IO (nombre d'instructions)
- prouvé comme étant optimal vis à vis de temps d'attente moyen

Ordonnancement basé sur une priorité

- Une priorité est associé à chaque processus
- Le CPU est alloué à celui qui a la priorité la plus haute
- En cas de priorité égale on utilise FCFS
- Le SFJ est un cas simple de stratégie utilisant une priorité (inverse de τ)
- D'une manière générales les algorithmes utilisant un priorité peuvent provoquer une attente infinie ou une famine
- Un solution est de corriger la priorité avec l'age du processus

Traitement avec préemption

La préemption concerne la gestion du processeur.

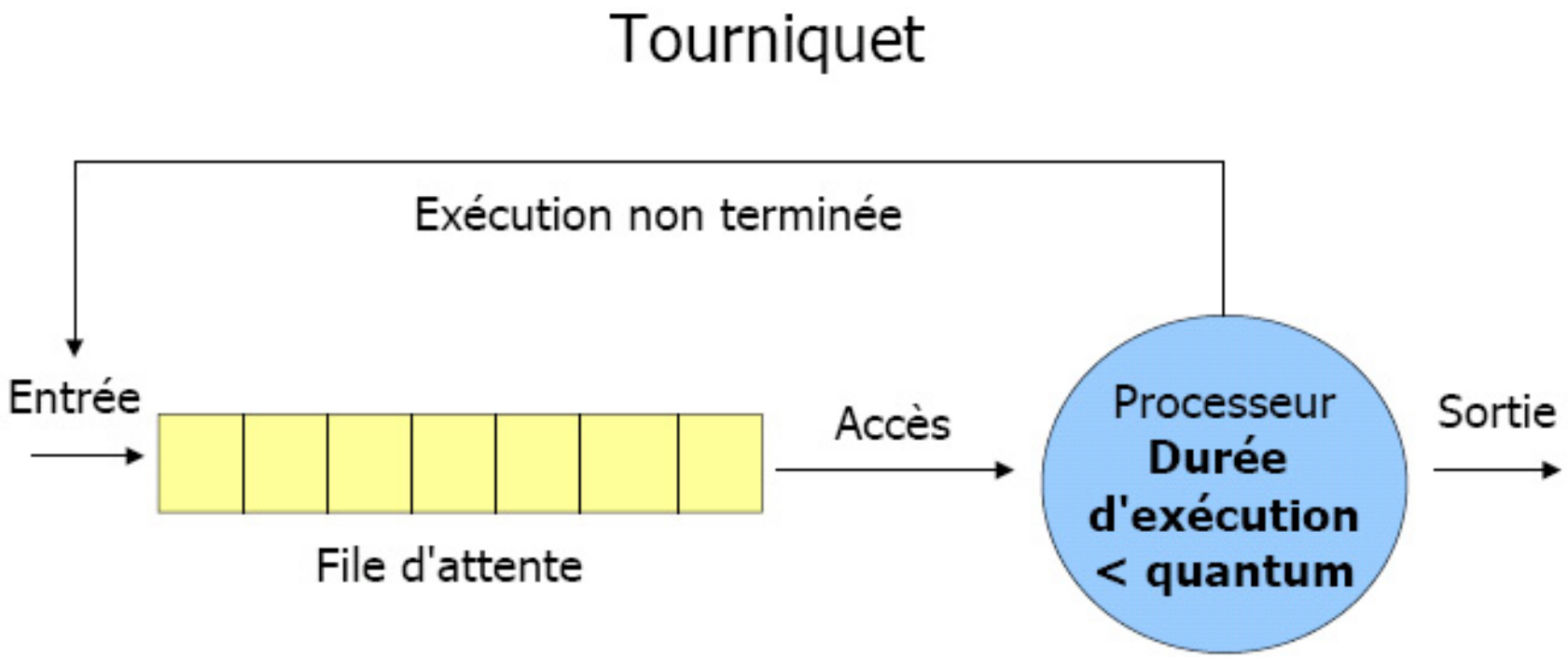
- Elle consiste à décider en fonction de certains critères, de remettre le processus en file d'attente avant la fin de son exécution
- Les processus font plusieurs passages dans la file d'attente
- Le temps d'attente d'un processus est sa durée d'attente cumulée
- Ce mécanisme est surtout utilisé dans la stratégie du tourniquet
- La stratégie SJF peut être adaptée pour un mode préemptif : si un processus plus court que le processus actif arrive dans la file, le processus actif est préempté

Traitement avec préemption

Stratégie du tourniquet RR (*Round-Robin*)

- objectif : vider les processus qui s'attardent trop dans le processeur
- l'ordonnanceur réalise la commutation de contexte
- un processus est remis en file d'attente dès que sa durée d'occupation du processeur dépasse une durée prédéfinie : quantum de temps
- la gestion de la file d'attente est faite selon le principe FIFO :
  - les travaux assez courts sont vite servis
  - les travaux longs sortiront du système au bout d'un temps fini

Traitement avec préemption



Traitement avec préemption

Stratégie du tourniquet RR (*Round-Robin*)

- l'efficacité du système dépend de la valeur du quantum :
  - trop petit, la machine perd du temps à changer de contexte (swapping)
  - trop long, les petits travaux ne sont pas exécutés
- on peut vouloir éviter que les processus long ne s'attardent pas trop dans le système : tourniquet exclusif
- on ajoute une file d'entrée au tourniquet et on détermine deux classes de processus :
  - les processus acceptés, qui sont intégrés au tourniquet
  - les nouveaux processus (arrivants)

SFJ scheduling

- Il des difficile de détermine la durée des phases de rafale CPU cependant on peut effectuer :
- prédiction de la prochaine valeur du CPU burst basée sur une moyenne exponentielle des précédentes mesures :

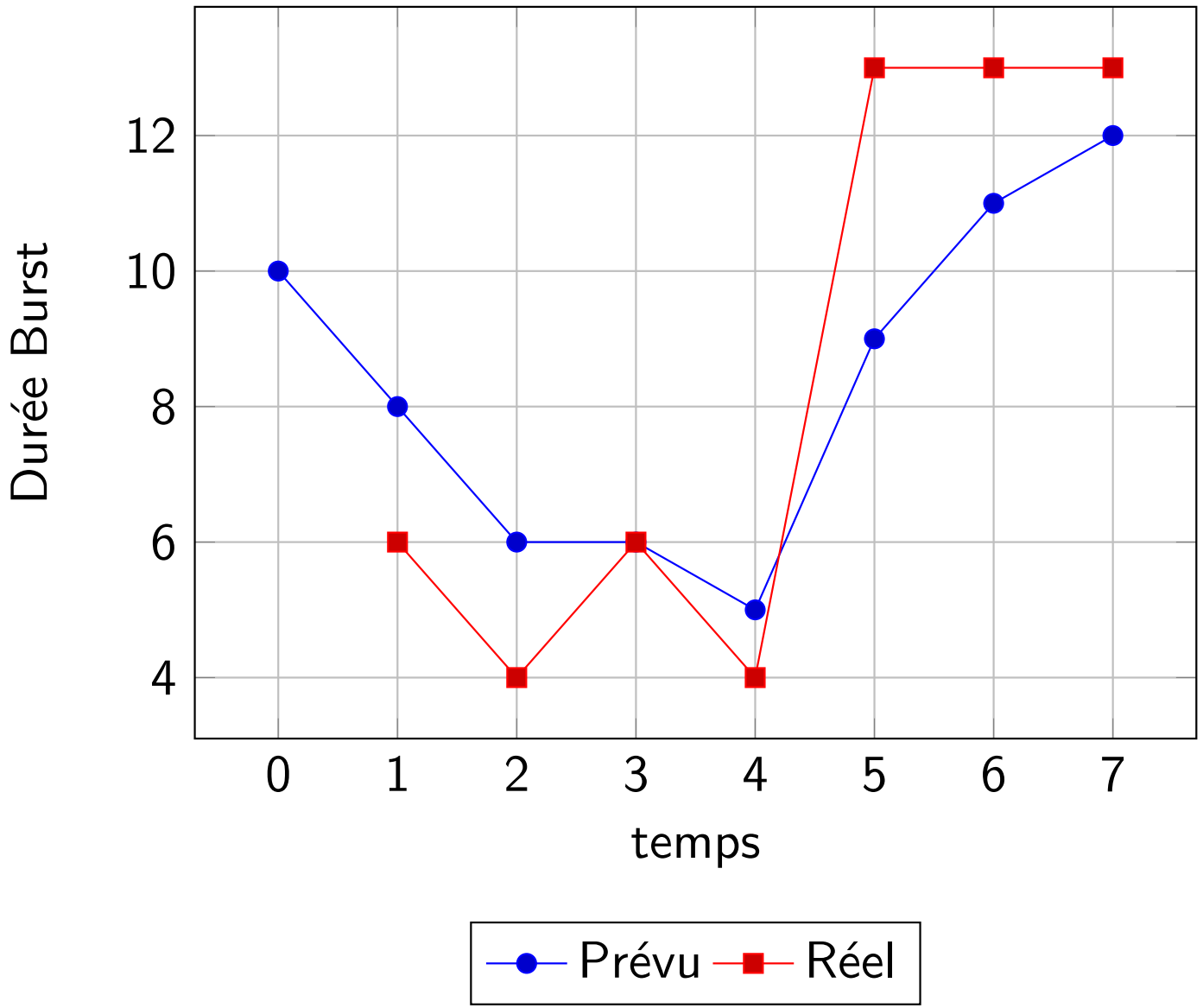
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$$

où

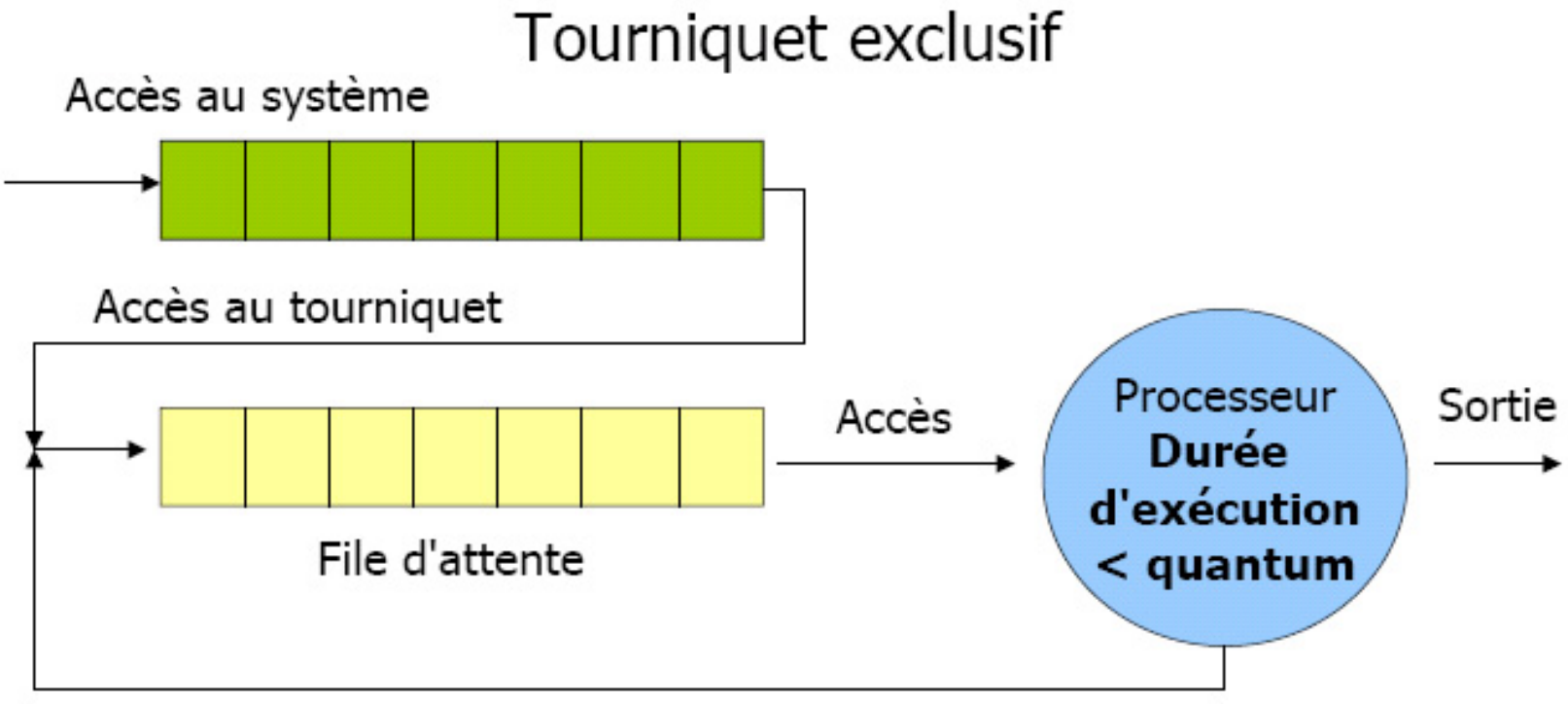
  - t_n : longueur du nieme CPU burst (mesuré)
  - $0 \leq \alpha \leq 1$ facteur de prise en compte du temps précédent

- si $\alpha = 0$ $\tau_{n+1} = \tau_n$
- si $\alpha = 1$ $\tau_{n+1} = t_n$
- on utilise τ pour interclasser les processus dans la file

SFJ scheduling



Traitement avec préemption



Traitement avec préemption (*priority scheduling*)

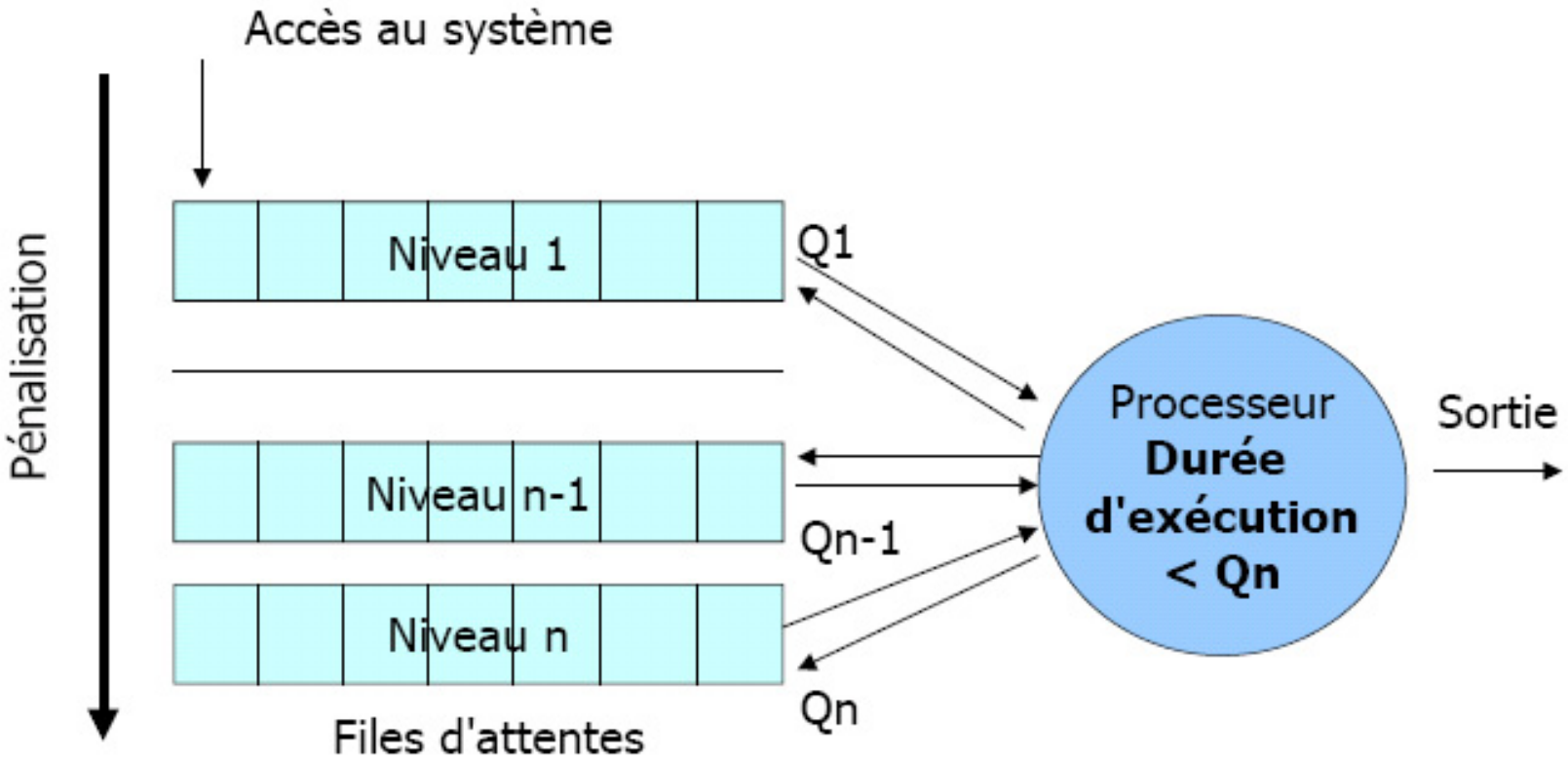
- On détermine une priorité $p = F(t)$
- Pour un processus P_i la priorité p_i augmente en fonction du temps passé dans le système
- La priorité d'un nouveau processus est toujours plus faible que la plus basse des priorités des processus acceptés
- Un nouveau processus dont la priorité à atteint celle d'un processus accepté est a son tour intégré au système
- Si la priorité des nouveaux processus augmente plus rapidement que celui des acceptés il peut y avoir un phénomène de rattrapage

Traitement avec préemption

Tourniquet à plusieurs files

- Les files d'attente permettent d'introduire une hiérarchie entre les demandeurs
  - fonction d'une priorité associée aux processus
  - fonction du temps déjà passé dans le système
  - ou bien d'autres critères

Traitement avec préemption et réquisition



Tourniquet avec files d'attente multiples

Traitement avec préemption et réquisition

Les règles de fonctionnement sont les suivantes :

- le niveau de priorité maximal (niveau 1) = file d'attente des processus arrivants
- à chaque file d'attente est associé un quantum de temps spécifique
- la file la moins prioritaire à un quantum de temps important ($Q_{n+1} \geq Q_n$)
- Quand un processus atteint la fin de son quantum de temps sans être terminé, il descend d'un étage
- les processus d'un niveau de priorité ne peuvent accéder au processeur que si les files de priorité supérieures sont vides
- l'apparition d'un nouveau processus dans une file de niveau inférieur à celui de la file d'origine d'un processus en cours provoque le vidage de ce processus (réquisition du CPU)

Critères à optimiser

- Utilisation du ou des CPU
- Débit (*Throughput*) : nombre de processus terminés par unité de temps
- Rotation (*Turnaround time*)
- Temps d'attente des processus (*Waiting Time*)
- Response Time (*Response Time*)

Ordonnacement pour les RT : EDT

Earliest Deadline First algorithme préemptif à priorité dynamique utilisé dans les systèmes temps réel.

- il attribue une priorité à chaque processus en fonction de l'échéance ;
- plus l'échéance d'une tâche est proche, plus sa priorité est grande.

Cet algorithme est difficile à implanter.

Ordonnancement sous UNIX

On distingue généralement trois classes de processus :

- les processus interactifs : interagissent constamment avec l'utilisateur (délai moyen de 50 à 150ms) comme par exemple les shell, les éditeurs et tout l'environnement graphique du bureau ;
- les processus batch : ne nécessitent pas d'interactions utilisateur, ils s'exécutent en tâche de fond (background) comme par exemple les SGBD, les calculs scientifiques ;
- les processus temps-réel : nécessitent des politiques d'ordonnancement spécifiques car ils ne doivent jamais se retrouver bloqués et doivent garantir un temps de réponse maximum (contrôle de robots, capteurs etc.)

Idées clés

Les ordonnanceur UNIX reposent sur les notions suivantes :

- les processus obtiennent un rang par rapport à leur priorité ;
- la priorité des processus est dynamique ;
- l'ordonnanceur garde un trace de ce que font les processus et ajuste périodiquement leur priorité ;
- la détermination du quantum est critique (par rapport au temps nécessaire pour le changement de contexte) ;
- un quantum long ne dégradera pas forcement les performances de processus interactifs car les priorités déclenchent une préemption .

Aperçu sur UNIX*BSD

- Les systèmes BSD 4.3 utilisent un RR
- Les files matérialisent des niveau de priorité différents (*multi-level-feddback round robin queues)*
- l'ordonnanceur *scheduler* parcourt les listes de haut en bas pour trouver un processus éligible
- il existe des listes internes pour le noyau avec la possibilité de doubler les processus endormis
- pour les liste utilisateur la règle générale de préemption est appliquée
- un processus qui utilise le CPU voit sa priorité augmenter
- un processus qui libère le CPU pour une E/S ne modifie pas sa priorité
- un processus qui épuise tout son quantum de tamps est préempté
- plus la priorité augmente moins le processus est prioritaire

Le cas Linux

- l'ordonnanceur du noyau 2.6 Linux utilise un algorithme heuristique pour décider de la classe du processus (batch, interactif, real-time) ;
- l'ordonnanceur cherche à favoriser les processus interactifs ;
- l'algorithme dans les anciennes versions : à chaque changement de contexte l'ordonnanceur liste les files des processus runnable, calcule leur priorités et sélectionne le meilleur processus à exécuter. Mais cet algorithme simple à une complexité fonction du nombre de processus dans la file.
- Completely Fair Scheduler (ordonnanceur complétement équitable) : gère l'allocation du processeur en maximisant l'utilisation globale du CPU tout en optimisant l'interactivité (auteur Ingo Molnár).