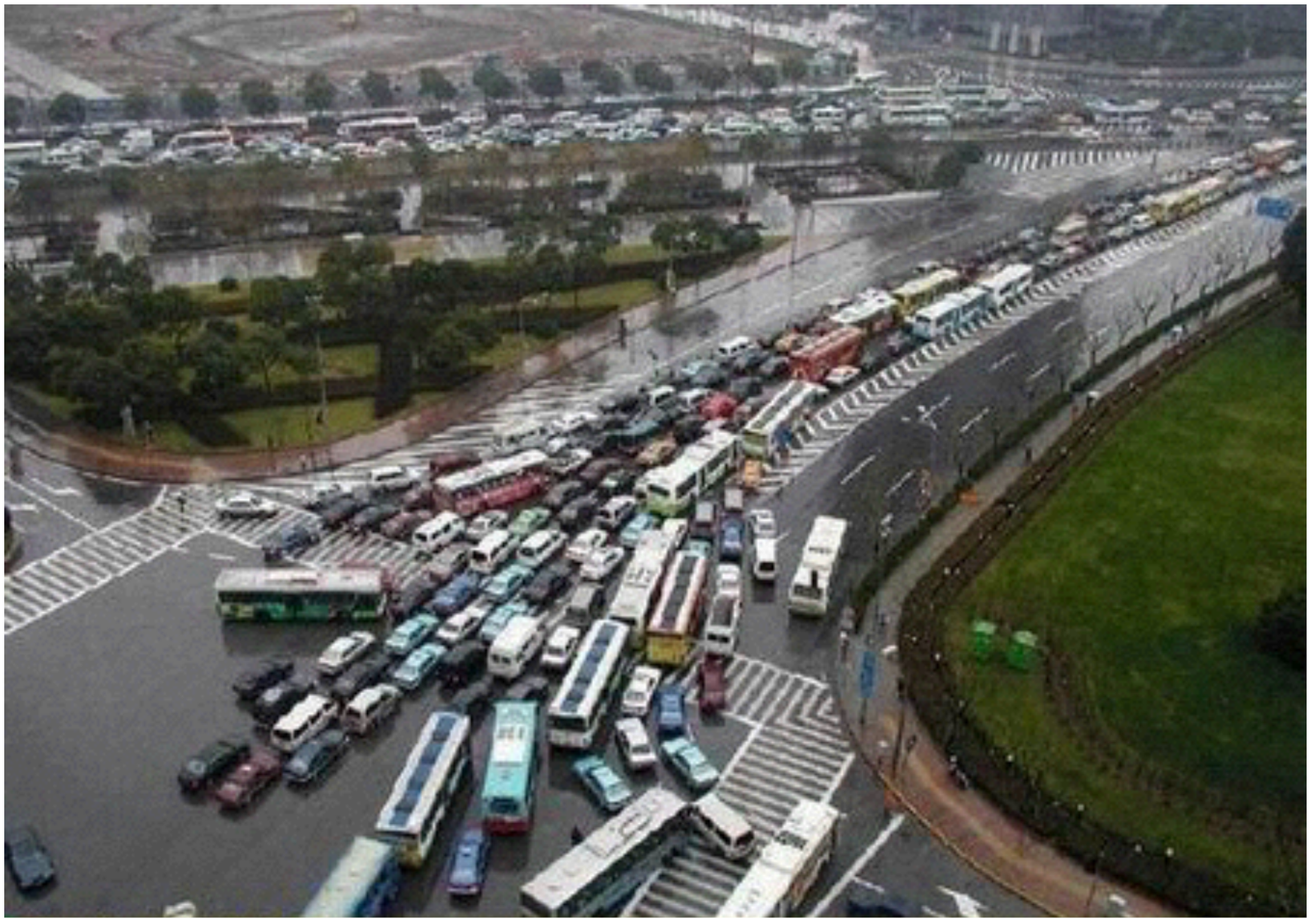
La gestion des processus



GIF-1001 Ordinateurs: Structure et Applications, Hiver 2016

Jean-François Lalonde

Image: fugly.com

Analogies

- Un illusionniste :
  - Fait disparaître certaines limites du matériels
  - Donne l'illusion que la machine a une mémoire infinie et une infinité de processeurs

- Un gouvernement :
  - Protège les utilisateurs les uns des autres
  - Partage des ressources de façon efficace et équitable

Crédit: P. Giguère

Programmes, processus et threads

- Un **programme** est un ensemble d'instructions et de variables dont le but est d'accomplir une tâche précise. Un programme est habituellement créé par un programmeur doté du compilateur adéquat.

- Un **processus** est composé d'un programme et de l'ensemble des ressources reliées à l'exécution du programme. Ces ressources incluent de la mémoire, des I/Os, des fichiers ouverts par le programme, du temps de CPU et autres. Un processus n'est pas un programme! Un processus est créé par:
  - une requête de l'usager (ex: exécution de programme),
  - le système d'exploitation
  - ou un autre processus (un processus parent crée un processus enfant ("child")).

- Un « **thread** » est une partie d'un processus qui peut être exécutée indépendamment des autres éléments du processus (en parallèle). Un thread a ses propres registres (incluant le compteur de programme) et sa propre pile, mais il partage le reste de ses ressources avec les autres constituant du processus. Les threads sont créés habituellement au début d'un programme. Ils peuvent servir à répondre à des événements dans des programmes (exemples d'événement: clic de souris, touche de clavier enfoncée, message reçu par port série, etc.).

Exécution multi-tâches

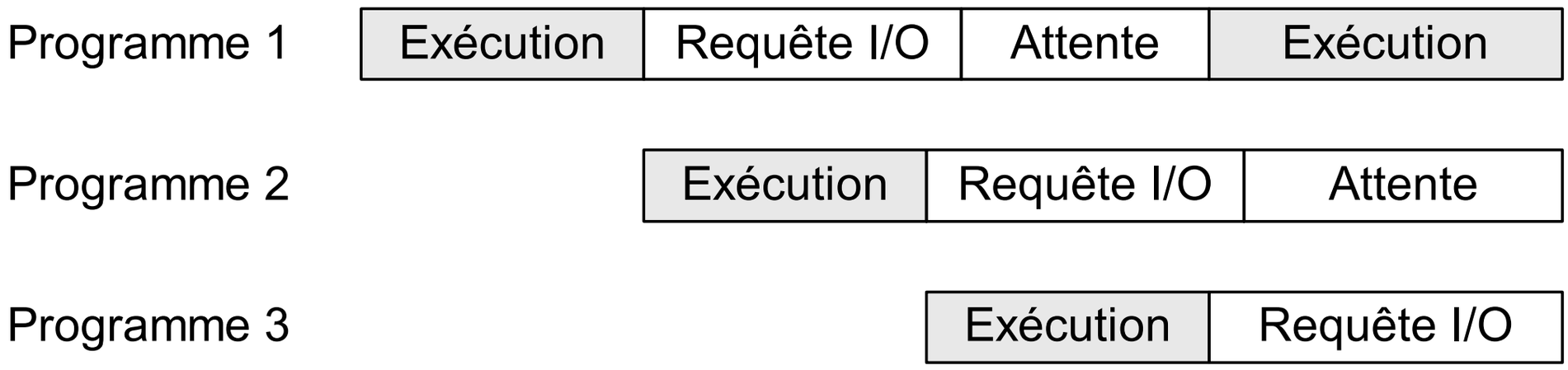
- Exécuter un seul programme à la fois est un gaspillage de temps et de ressources, car les programmes attendent souvent après un périphérique. Par exemple, un programme peut attendre qu'un fichier sur le disque dur soit lu. Pendant ce temps, le microprocesseur tourne à vide.

- Certains programmes ont une vitesse d'exécution limitée par la puissance du microprocesseur (CPU Bound), mais la plupart des programmes sont limités par de nombreux accès aux périphériques (I/O Bound).

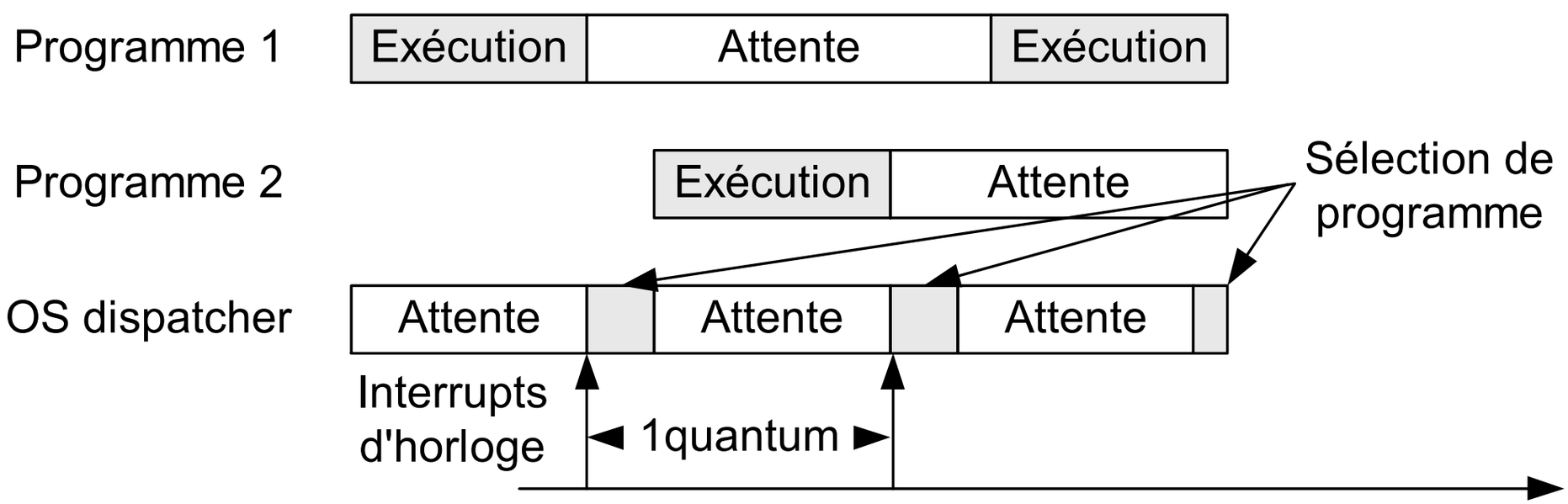
- Un système d'exploitation multi-tâches est un SE où plusieurs programmes roulent « simultanément ». Il faut rappeler que le microprocesseur n'exécute qu'un programme à fois. L'acétate suivante montre deux façons d'exécuter plusieurs programmes « simultanément ».

Exécution multi-tâches

- Partage du CPU lors d'attente après les I/Os



- Partage du CPU dans le temps



Rappel: Interruptions et système d'exploitation

- Les interruptions permettent l'exécution de plusieurs processus
- Comment?
  - Une horloge génère des interruptions périodiquement
  - À chaque interruption, on change le processus à exécuter

Exécution multi-tâches

- Le SE change le programme exécuté:
    - lorsque ce programme attends après un périphérique. Lorsqu'un programme utilise les services d'I/O, un nouveau programme est exécuté.
    - lorsque l'horloge du système émet une interruption.
- Le SE interrompt le programme qui roule avec une interruption répétitive (basée sur l'horloge du système). Lorsque cette interruption survient, le système d'exploitation utilise un peu de temps de CPU afin de déterminer quel est le prochain programme à exécuter. Chaque programme est ainsi exécuté en petits morceaux (time-slicing) de temps appelés quantum. L'opération consistant à choisir le prochain programme exécuté se nomme ordonnancement (dispatching).
- L'ordonnanceur (dispatcher) choisit le prochain programme à effectuer en fonction de plusieurs critères (priorité, temps inactif, bloqué par un accès aux périphériques…).
- Un SE qui limite le temps d'exécution d'un programme afin de faire de la supervision est dit préemptif (avec réquisition). Lorsqu'il n'y pas de réquisition, les programmes s'exécutent un peu plus vite car le système d'exploitation ne revient pas continuellement. Toutefois, il ne faut pas que le programme plante…

Process Control Block (PCB)

- Chaque processus:
  - a un bloc de contrôle qui le décrit;
  - a un IDentifieur unique;
  - peut avoir des enfants ou un père;
  - a un état;
  - a ses registres, sa mémoire et sa pile;
  - a une priorité.
- Les processus peuvent partager de la mémoire, des processus, des fichiers, des I/Os et autres.

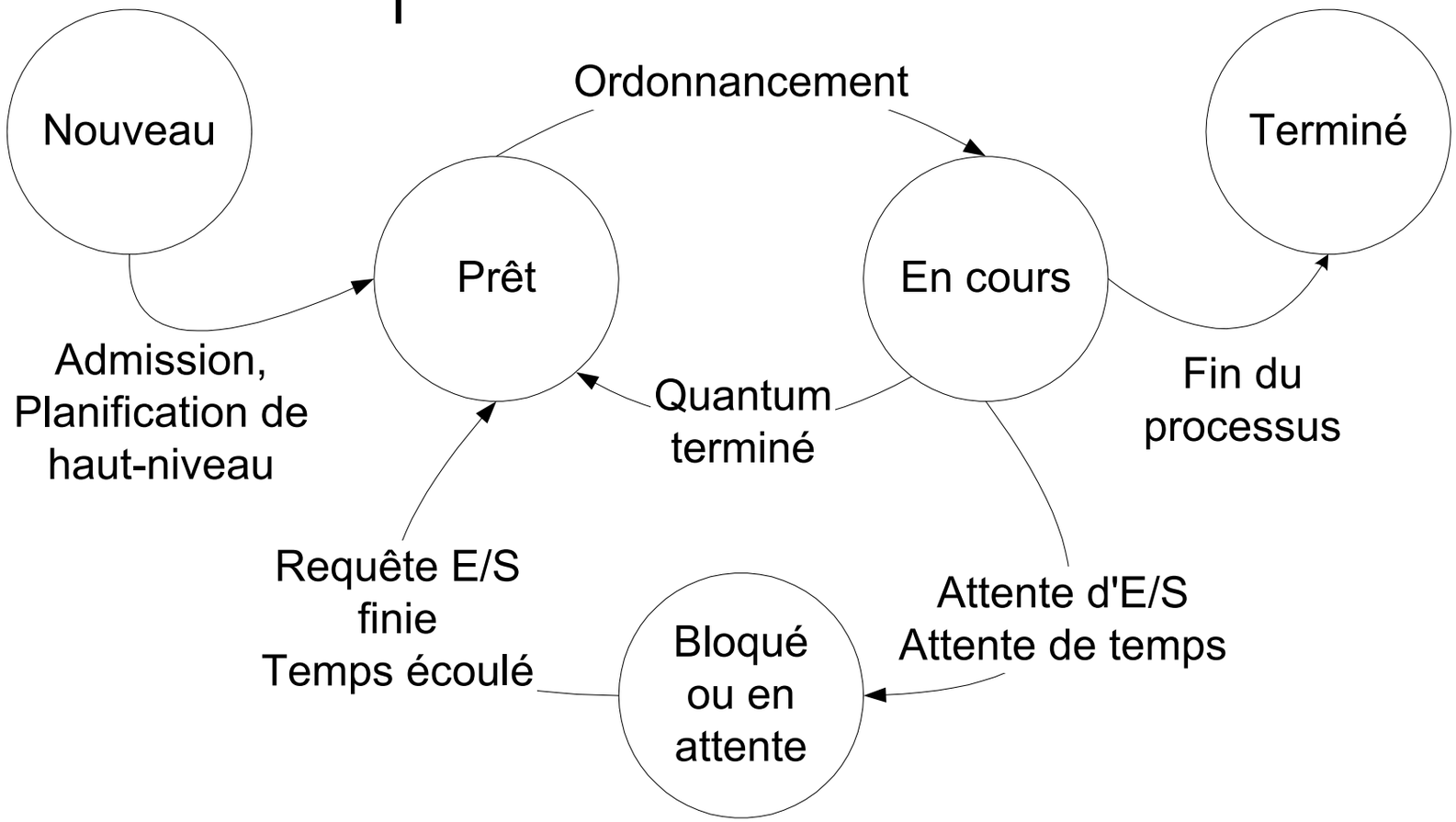
Process ID (PID)
Pointer to parent process
Pointers to child processes
Process state
Program Counter
Registers
Memory pointers
Priority information
Accounting information
Pointers to shared ressources

PCB Typique

Planification de haut niveau

- Il y a deux niveaux de planification:
  - L'**admission** de haut niveau ("high level scheduler") est responsable de l'admission des processus.
    - Décide quel programme sera mis en mémoire.
    - Sert à décider quand et si un processus sera admis en fonction de la mémoire et des E/Ss disponible. Il essaie d'optimiser l'utilisation des E/Ss et de la mémoire.
    - Dans un environnement interactif, il a un rôle moins important: les processus sont admis rapidement afin de satisfaire les demandes de l'usager sans retard. Le high level scheduler devient important dans un environnement où les programmes sont exécutés en groupes et lorsque les ressources du système sont limitées.
  - L'**ordonnancement** des processus ("dispatcher"), qui est responsable de l'ordre dans lequel les processus admis seront exécutés

États des processus



- En cours d'exécution
  - processus OK, processeur OK
- Prêt (suspendu provisoirement pendant qu'un autre processus s'exécute)
  - processus OK, processeur occupé
- Bloqué ou en attente (attendant un événement d'un périphérique ou suspendu pour un certain temps)
  - processus non OK, même si processeur OK

États des processus

- Lorsqu'un processus est admis, il est mis dans l'état Prêt/Ready par défaut. Il est prêt à être exécuté.
- Quand le dispatcher détermine que le processus doit être exécuté, il le met en mode En Cours/Running et il l'exécute. Il n'y habituellement qu'un seul processus qui roule à la fois par microprocesseur.
- Lorsque le dispatcher est appelé de nouveau, il peut décider de cesser d'exécuter le processus en cours pour en exécuter un autre. Le processus en cours est remis dans l'état Prêt/Ready.
- Un processus qui roule peut faire des requêtes à des E/Ss et attendre qu'elles soient complétées. Il devient bloqué. D'autres processus peuvent rouler pendant que l'accès aux E/Ss se fait.
- Lorsque l'accès au périphérique est complété pour un processus donné, ce processus est remis à l'état Prêt/Ready.
- Lorsqu'un processus se termine, il est détruit, terminé ou « tué »
- Il existe d'autres états non affichés pour les processus: suspendu (par l'usager ou par manque de ressources), en reprise (après avoir été suspendu), swap (voir plus loin).

Ordonnancement (Dispatching)

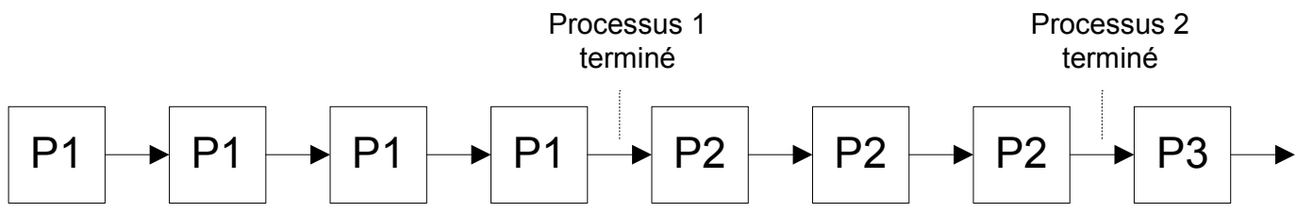
- L'ordonnancement des processus consiste simplement à décider quel processus sera exécuté dans le quantum suivant.
- Il existe plusieurs algorithmes de dispatching présentés plus loin.
- Tous les algorithmes de dispatching devraient avoir les objectifs suivants:

Assurer l'équité	Tous les processus sont traités également
Maximiser l'exécution	Terminer le plus de processus possible
Temps d'exécution min.	Temps d'exécution le plus court possible
Utilisation max du CPU	Le CPU doit être utilisé au maximum
Utilisation max ressources	Les ressources doivent être utilisées au max.
Détérioration graduelle	Un système surchargé doit ralentir, pas planter
Temps d'attente min.	Petit délai entre l'admission et l'exécution
Temps de réponse correct	Tâches longues, longues et tâches courtes, courtes
Prévenir la famine (starvation)	L'exécution d'un processus ne doit pas être reportée indéfiniment.

Analogie de la vie de tous les jours™

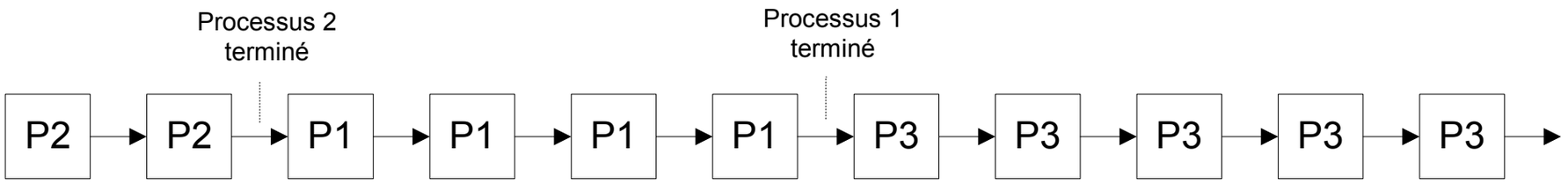
NETFLIX

Algorithmes d'ordonnancement



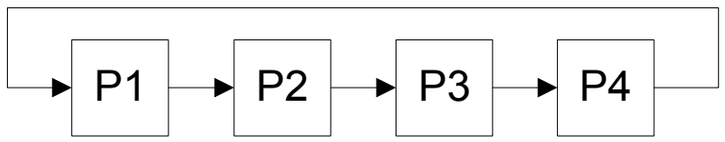
- **Premier arrivé, premier servi** (exemple de mauvais algorithme)
  - Le premier processus admis est exécuté jusqu'à sa fin. Puis, on exécute le suivant.
  - Cet algorithme n'est jamais utilisé. Il équivaudrait à remplacer le noyau du système d'exploitation par une file d'exécution des processus et détruirait l'illusion d'une exécution simultanée de plusieurs processus.

Algorithmes d’ordonnancement



- **Le plus court d’abord**: On exécute le processus le plus court d’abord.
  - Avantages: Maximise l’exécution et le temps d’exécution
  - Désavantages: Famine possible, inéquitable

Algorithmes d’ordonnancement



- **Le tourniquet (round-robin)**: On exécute les processus à tour de rôle.
  - Avantages: Tous les processus ont du temps de CPU, très équitable
  - Désavantages: Ne maximise pas l’exécution ni le temps d’exécution

Exercice pour se réchauffer

- Les processus suivants sont admis à l'ordonnanceur (dans l'ordre):
  - P1, durée: 5
  - P2, durée: 3
- Ecrivez quel processus est exécuté par le micro-processeur à chaque quantum de temps si l'algorithme utilisé par l'ordonnanceur est:
  - premier arrivé, premier servi
  - plus court d'abord
  - tourniquet

Exercice pour se réchauffer

- Les processus suivants sont admis à l'ordonnanceur (dans l'ordre):
  - P1, durée: 5
  - P2, durée: 3
- Ecrivez quel processus est exécuté par le micro-processeur à chaque quantum de temps si l'algorithme utilisé par l'ordonnanceur est:
  - premier arrivé, premier servi: P1—P1—P1—P1—P1—P2—P2—P2
  - plus court d'abord: P2—P2—P2—P1—P1—P1—P1—P1
  - tourniquet: P1—P2—P1—P2—P1—P2—P1—P1

Exercice pour se réchauffer (bis)

- Les processus suivants sont admis à l'ordonnanceur (dans l'ordre):
  - P1, durée: 5, temps d'arrivée: 0
  - P2, durée: 4, temps d'arrivée: 0
  - P3, durée: 2, temps d'arrivée: 2
  - P4, durée: 3, temps d'arrivée: 5
- Écrivez quel processus est exécuté par le micro-processeur à chaque quantum de temps si l'algorithme utilisé par l'ordonnanceur est:
  - premier arrivé, premier servi
  - plus court d'abord
  - tourniquet

Exercice pour se réchauffer (bis)

- Les processus suivants sont admis à l'ordonnanceur (dans l'ordre):
  - P1, durée: 5, temps d'arrivée: 0
  - P2, durée: 4, temps d'arrivée: 0
  - P3, durée: 1, temps d'arrivée: 2
  - P4, durée: 3, temps d'arrivée: 5
- Ecrivez quel processus est exécuté par le micro-processeur à chaque quantum de temps si l'algorithme utilisé par l'ordonnanceur est:
  - premier arrivé, premier servi
    - P1—P1—P1—P1—P1—P2—P2—P2—P2—P3—P4—P4—P4
  - plus court d'abord
    - P2—P2—P3—P2—P2—P4—P4—P4—P1—P1—P1—P1—P1
  - tourniquet
    - P1—P2—P3—P1—P2—P4—P1—P2—P4—P1—P2—P4—P1

Algorithmes d'ordonnancement

- **Priorité**: On exécute les processus selon leur priorité.
  - Avantages: Temps de réponse correct, temps d'attente minimum
  - Désavantages: Ne maximise pas l'exécution ni le temps d'exécution, famine possible, le programmeur doit déclarer des priorités

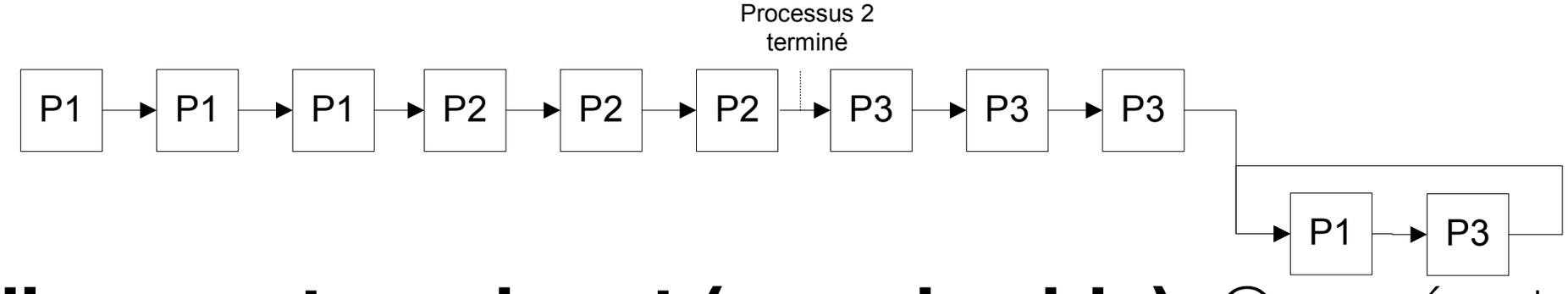
Algorithmes d'ordonnancement

- **Priorité Variable**: On exécute les processus selon leur priorité. La priorité d'un processus change dynamiquement en fonction d'évènements (fin d'attente, le processus a faim, admission, …).

- Avantages: Temps de réponse correct, temps d'attente minimum

- Désavantages: Même que l'algorithme de priorité, mais avec un impact beaucoup moindre.

Algorithmes d'ordonnancement



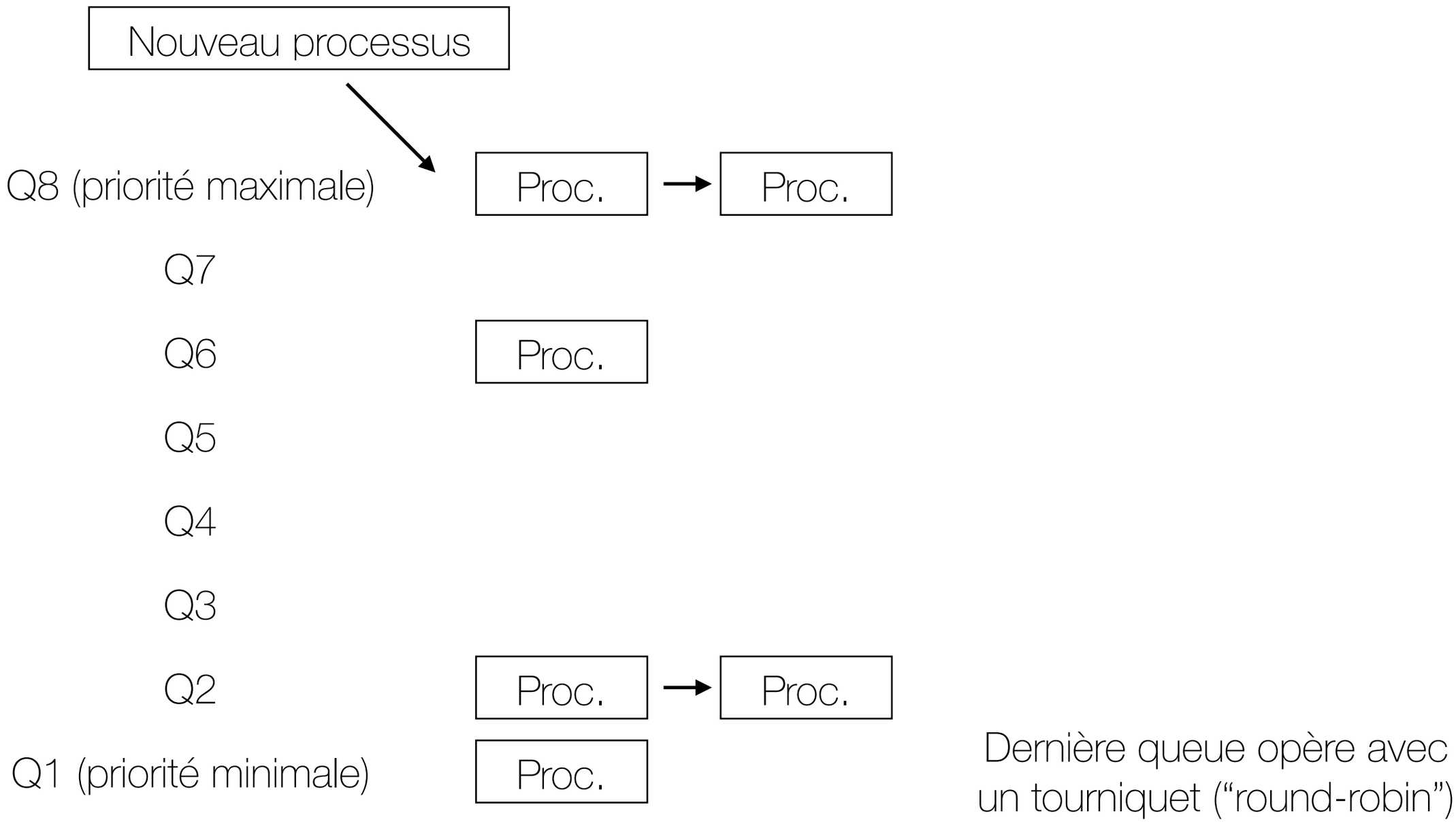
- **File avec tourniquet (round-robin)**: On exécute plusieurs fois les processus nouvellement admis, puis les processus non terminés sont mis dans un tourniquet.
  - Avantages: Tous les processus ont du temps de CPU; très équitable.
  - Désavantages: Même que tourniquet, mais avec un impact moindre

Algorithmes d’ordonnancement

- **Le plus critique (RTOS surtout)**: Le processus devant être exécuté dans les plus brefs délais est d’abord exécuté.
  - Avantages: Garantit l’exécution d’un processus critique à l’intérieur d’un certain temps.
  - Désavantages: Ne maximise pas l’exécution ni le temps d’exécution, famine possible, le programmeur doit déclarer des priorités

Ordonnancement — Windows

- Algorithme: “Multilevel Feedback Queues”



Ordonnancement — Windows

- Variantes:
  - Nombre de queues
  - L’algorithme d’ordonnancement à utiliser pour chaque queue (peut être différent)
  - Comment assigner un processus à une queue à priorité plus élevée (ou plus faible)

Ordonnancement — Linux

- Algorithme: “Completely Fair Scheduler”
  - Arbre “rouge-noir” — arbre binaire balancé
  - Assigne un “score” à chaque processus basé sur une comparaison avec un OS idéal
    - OS idéal: traite tous les processus en parallèle
- Le score est élevé si la différence entre le temps de traitement alloué au processus et le temps que ça aurait dû prendre est élevé

“Swapping”

- Tous les PCBs sont habituellement en mémoire parce que le dispatcher doit avoir les informations sur les processus rapidement.
- Lorsqu’un processus est inactif depuis longtemps (il est bloqué par une opération très longue par exemple), son PCB peut être mis sur le disque dur pour récupérer de la mémoire. Le PCB sur le disque dur est un swap file.

Références et exercices

- Références
  - Irv Englander: chap. 15.3, 18 (jusqu'à 18.5)
  - William Stallings: section 8.2