BUT 2 / R3.05
PROGRAMMATION SYSTÈME

ORDONNANCEMENT

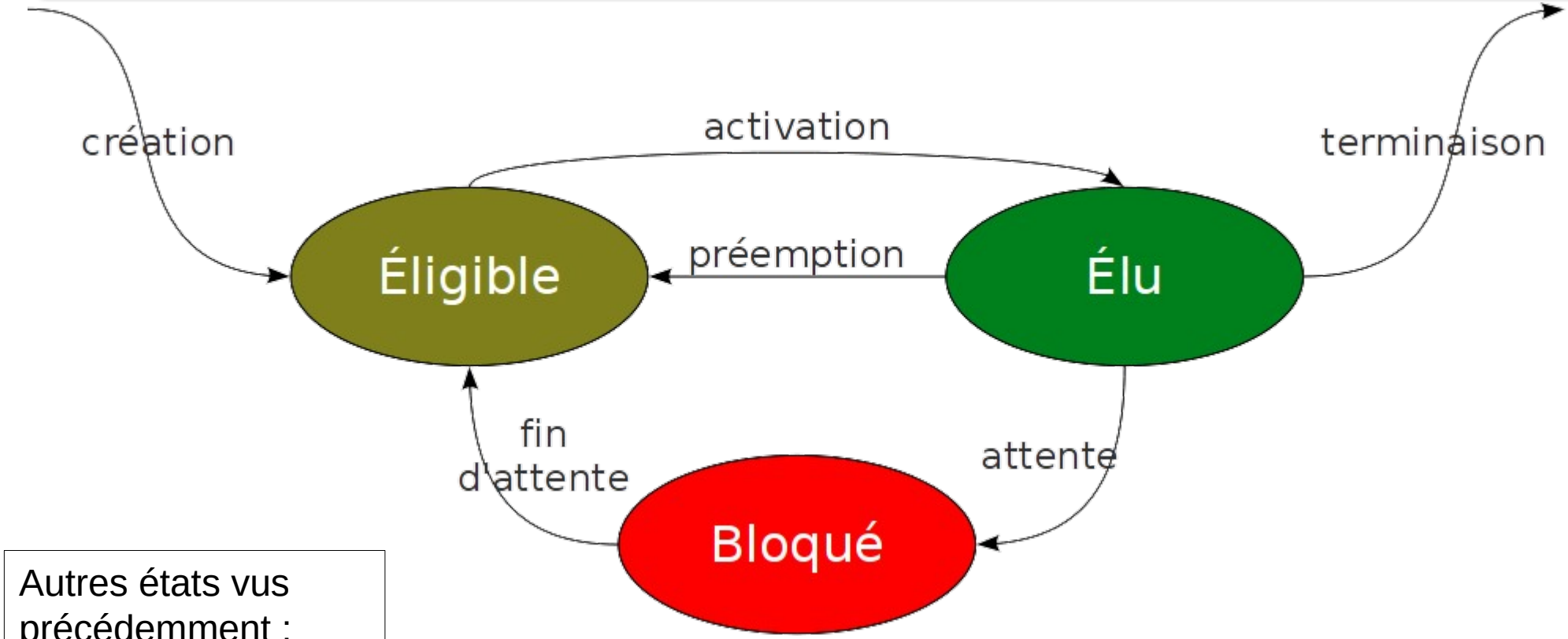
DEFINITION

- Dans un système multi-tâches les processus se partagent un ou plusieurs processeurs/cœurs
- A chaque processus est alloué un laps de temps (**quantum**) durant lequel il est exécuté
- Un processus peut être stoppé :
  - Volontairement (par le processus) ex : appel système, entrée-sortie
  - Involontairement (par le SE) → **préemption**
- L'ordonnancement désigne l'algorithme qui alloue les ressources CPU aux différents processus.
- L'algorithme utilisé est propre au système d'exploitation

CHANGEMENT DE CONTEXTE

- Sur un système multi-tâches, les étapes sont les suivantes:
  - Exécution du processus P1 durant un laps de temps
  - Changement de contexte:
    - Copie de l'environnement du processus P1 en RAM (copie des registres par exemple)
    - Mise en place du processus P2 (copie de l'environnement d'exécution de P2 dans le microprocesseur)
    - Exécution de P2
  - En pratique, on ne passe pas de P1 à P2 directement mais de P1 au SE, SE à P2

ETAT D'UN PROCESSUS (SIMPLIFIÉ)



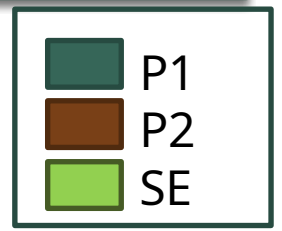
Autres états vus précédemment :

- Zombie
- Orphelin

- En pause (voir signaux SIGSTOP/SIGCONT)
- En attente d'un périphérique (ex. réception réseau)
- En attente de synchronisation (sémaphore, etc.)
- Etc.

EXEMPLE

- 2 processus sont en cours d'exécution



1 processeur:

2 processeurs:

TYPE DE PROCESSUS

2 « types » de processus :

- « **I/O bound** » : qui sont beaucoup en attente car ils demandent/attendent des réponses entrée/sorties
- « **CPU bound** » : qui utilisent tout leur quantum sans se bloquer

Remarque : un processus peut être tantôt l'un ou tantôt l'autre durant son fonctionnement

Il faut que les algorithmes puissent contenter ces deux types de processus

FIFO sans préemption

Pas utilisé par les SE modernes (sauf exception)

- Premier arrivé, premier servi (First In First Out)

- Les processus rendent la main « de leur plein gré »
  - Lorsqu'ils se terminent
  - Lorsqu'ils se bloquent
  - Lorsqu'ils font un appel système type *yield*

→ *aujourd'hui, avec les structures multi-coeurs/multi-processeurs, on peut avoir des algorithmes qui auront au final plus ou moins ce comportement (certains processus peuvent monopoliser un cœur)*

→ *peut être utilisé dans certains cas. Ex : processus « temps réel » sous Linux*

L'ALGORITHME FIFO AVEC PRÉEMPTION (TOURNIQUET)

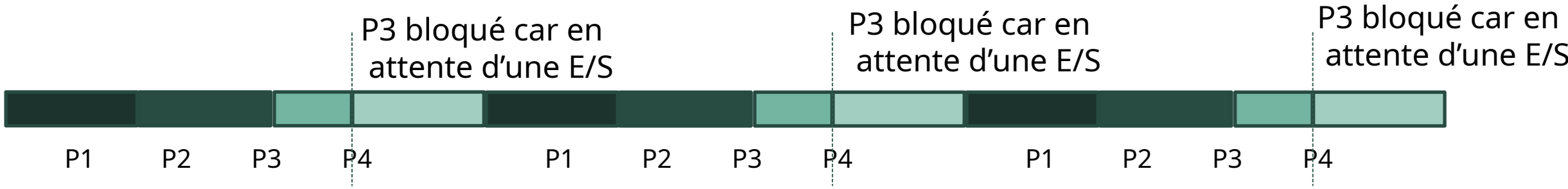
« Round Robin » ou « ruban rond », ou « tourniquet »

- Alloue un **quantum** à chaque processus
  → temps durant lequel il peut exécuter ses instructions

- Chaque processus accède au CPU à tour de rôle

L'ALGORITHME FIFO AVEC PRÉEMPTION (2)

- Pas de quantum optimal (dépend du nombre de processus, etc.)
- Équitable sauf pour des processus nécessitant des interactions fréquentes avec les E/S (ou mis en attente souvent)
- Exemple:



Le processus P3 a moins de CPU alloué que les autres à cause de ses blocages.

Algorithmes avec anticipation

- Shortest Job Next
  → on exécute d'abord le processus qui nécessite le moins de temps

- Avec préemption :
  - Si un nouveau processus arrive, on compare les temps restants de chaque processus à exécuter, et on préempte éventuellement le processus en cours

- Variante en prenant en compte le temps d'attente :
  - On divise le temps restant par le temps qu'il a passé à attendre

→ tous ces algos nécessitent de connaître/d'avoir une estimation du temps nécessaire pour accomplir les tâches. En pratique : difficile voire impossible

Multi-level feedback queue

Utilisé dans Windows 10

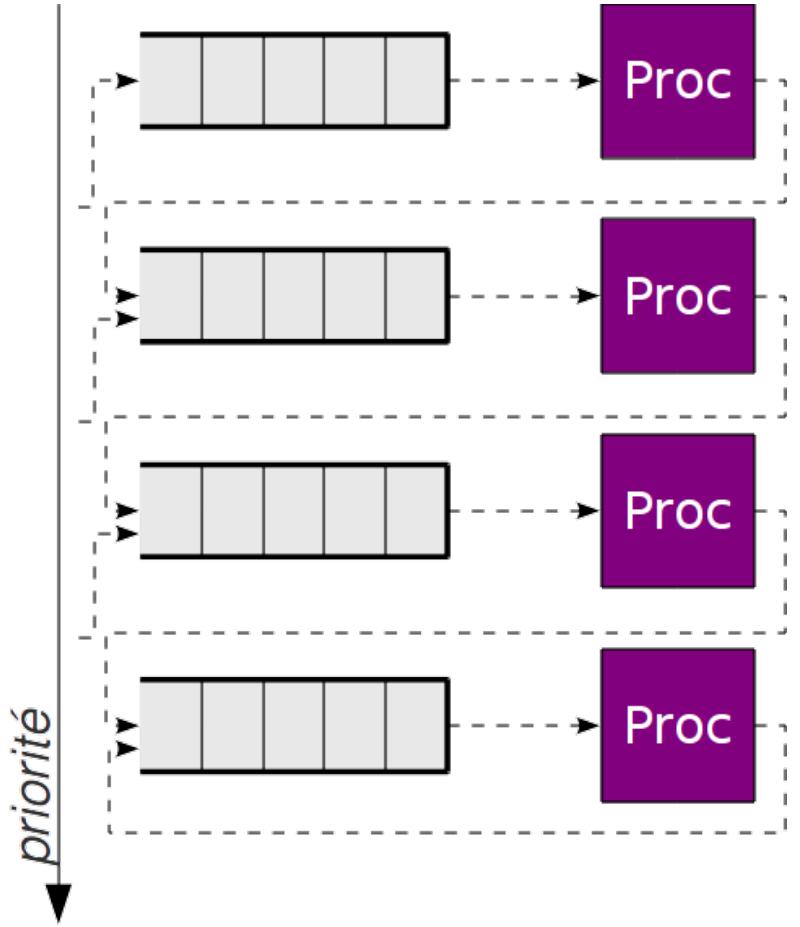
Idée très simplifiée :

- Plusieurs files de priorité
  → chacune fonctionne en « tourniquet » préemptif
  → la file du haut est la plus prioritaire

- Si un processus rend lui-même la main (ex : attente d'entrée-sortie), il reste dans sa file

- Si le quantum d'un processus se termine, il passe dans le niveau inférieur

- Si un processus attend trop longtemps dans une file à basse priorité, ou s'il attend souvent des réponses d'entrée-sortie, il remonte



Completely Fair Scheduler

- **Algorithme d'ordonnancement actuel du noyau Linux (depuis 2007)**

  Idée très simplifiée :

- À chaque changement de contexte :
  - le processus qui a reçu le moins de temps CPU est choisi
    → score normalisé par rapport aux valeurs des autres processus en attente
    → score pondéré par une notion de **priorité** des processus

  - On calcule le quantum alloué au processus en fonction du nombre de processus en attente

- On stocke les processus en attente grâce à un « arbre rouge-noir »

PRIORITÉ

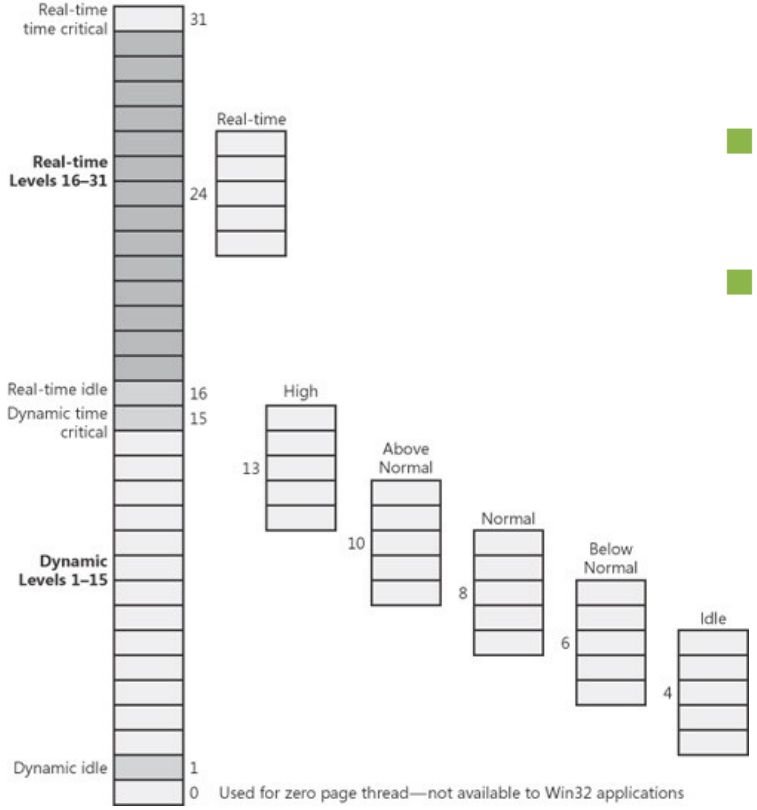
- Possibilité de donner des priorités aux processus
  - Cela signifie des allocations de temps (quantum) ou un nombre de quantums qui diffèrent suivant la priorité
- Deux types de priorité
  - Interne: priorité fixée par le SE (peut changer au fil du temps)
  - Externe: Priorité définie par l'utilisateur

WINDOWS

- Définit des classes de priorité
  - 32 priorités et 6 classes (plus une pour le SE)
  - Par défaut celle du processus qu'il la lancé
  - Peut-être modifié avec
    - start (commande windows qui permet de lancer un processus)
    - CreateProcess (system call)
    - SetPriority (system call) pour un processus déjà créé

WINDOWS (2)

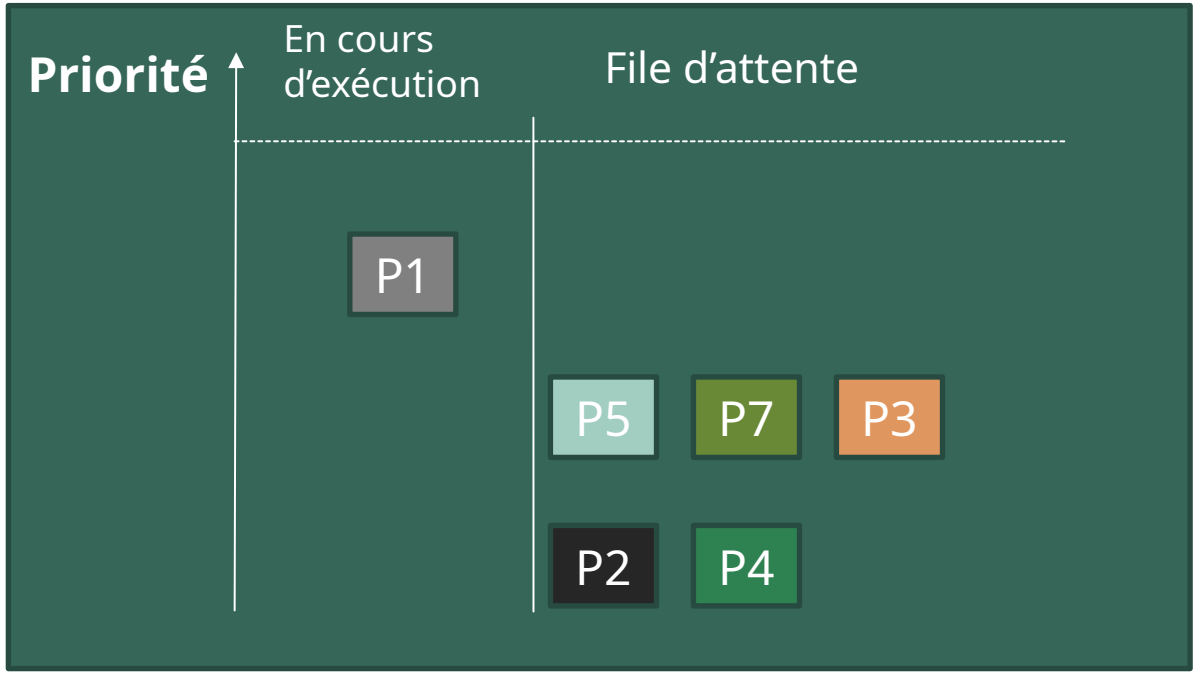
- Un processus est lancé avec la priorité de sa classe (celle du milieu: 24, 13, 10, 8, 6, 4)
- Windows adapte la priorité d'un processus en fonction de son activité, des temps systèmes associés et des attentes (E/S, etc.)



- Les priorités restent dans la classe du processus: par exemple entre 11 et 15 pour la classe « High »
- Un changement de priorité aide à l'équité de traitement: attente (E/S, synchronisation, etc.), moins de temps CPU alloué, etc.

WINDOWS (3)

- Les priorités sont strictes: seuls les processus de priorités plus élevés sont exécutés. Mais:
  - les processus sont souvent en attente,
  - les processus se terminent,
  - et les processus changent de priorité dans leur classe.



Autant de files que de priorité (donc 31).

FIFO préemptif au sein d'une priorité donnée.

Le processus P1 est exécuté. Il est le seul dans sa priorité donc garde le CPU.

P5 sera exécuté lorsque P1 se terminera, ou sera en attente ou si windows change sa priorité (où celle d'un autre processus).

LINUX

- Linux utilise des priorités `nice` (de -20 à +19)
  - Par défaut 0
  - *Niceness* basse = priorité haute
  - Un processus hérite de la priorité de son père
    - Possibilité de la changer lors du lancement commande ou appel système `nice()` / en cours d'exécution appel système *setpriority()*
  - Utilisateur: de 0 à 19
  - Root: toutes les priorités

- À partir de la *niceness*, le SE associe une priorité (*priority*) entre 0 et 139
  - 0 à 99 pour le SE « temps réel »
  - 100 à 139 pour les utilisateurs
  - Au lancement du processus : priority = 120 + niceness
  - Le SE peut ensuite changer la priorité au fur et à mesure de l'exécution

LINUX (2)

- Algorithmes dépendant de la version du noyau
- Actuellement: CFS (Completely Fair Scheduling Algorithm)
- Allocation de quantum à chaque processus (temps processeur)
- Le quantum dépend de la priorité du processus et des priorités de tous les autres
  - Utilise un poids fonction des priorités des processus prêts

QUESTIONS

- À quel des cinq « A » l'ordonnancement est associé ?
- Sous Linux, quelle est la différence entre la *niceness* et la *priority* ?
- En quoi consiste un « changement de contexte », et quand a-t-il lieu ?
- En l'absence de blocage (si chaque processus utilise tout son quantum), l'algorithme FIFO avec préemption est équitable (alloue le même temps CPU à tous les processus)
  - Vrai / Faux

MANIPULATIONS

1) Lister les processus en cours d'exécution Windows / Linux

   - Sous linux : `ps -el` et `top`

- Sous windows : Ctrl-Alt-Sup puis gestionnaire des tâches

2) Lancer des processus avec différentes priorités

   - Sous Linux : lancer gedit avec une priorité de 10 (commande nice). Vérifier que la priorité est conforme avec ps ou top.

   - Sous windows : dans une console dos lancez deux notepad avec deux familles de priorités.

3) start /normal notepad
4) start /high notepad

5) Aller dans le gestionnaire des taches, onglet détails, clic droit sur le processus et vérifier dans le menu « définir la priorité » que celles-ci sont bien « high » et « normal »

MANIPULATIONS SUITE

Sous Linux, les informations relatives à un processus se situent dans le dossier

/proc/X/

Où **X** est son pid

pour afficher le pid du shell, tapez : ***echo $$***

Notamment :

- Dossier **fd**
- Fichier **sched**

Indiquez/retrouvez

- La priorité (champ prio – 120)
- Le temps d'exécution de ce processus (sum_exec_runtime)
- Le nombre de changements de contexte
- A quoi correspondent les champs nr_voluntary_switches et nr_involontary_switches