ESIR SE-S8

# Unité 1: Le Noyau

François Taïani

(Source : Isabelle Puaut, ISTIC)

# Préambule

- Prenez 5 minutes pour trouver 3 raisons pour utiliser un système d'exploitation

# Préambule

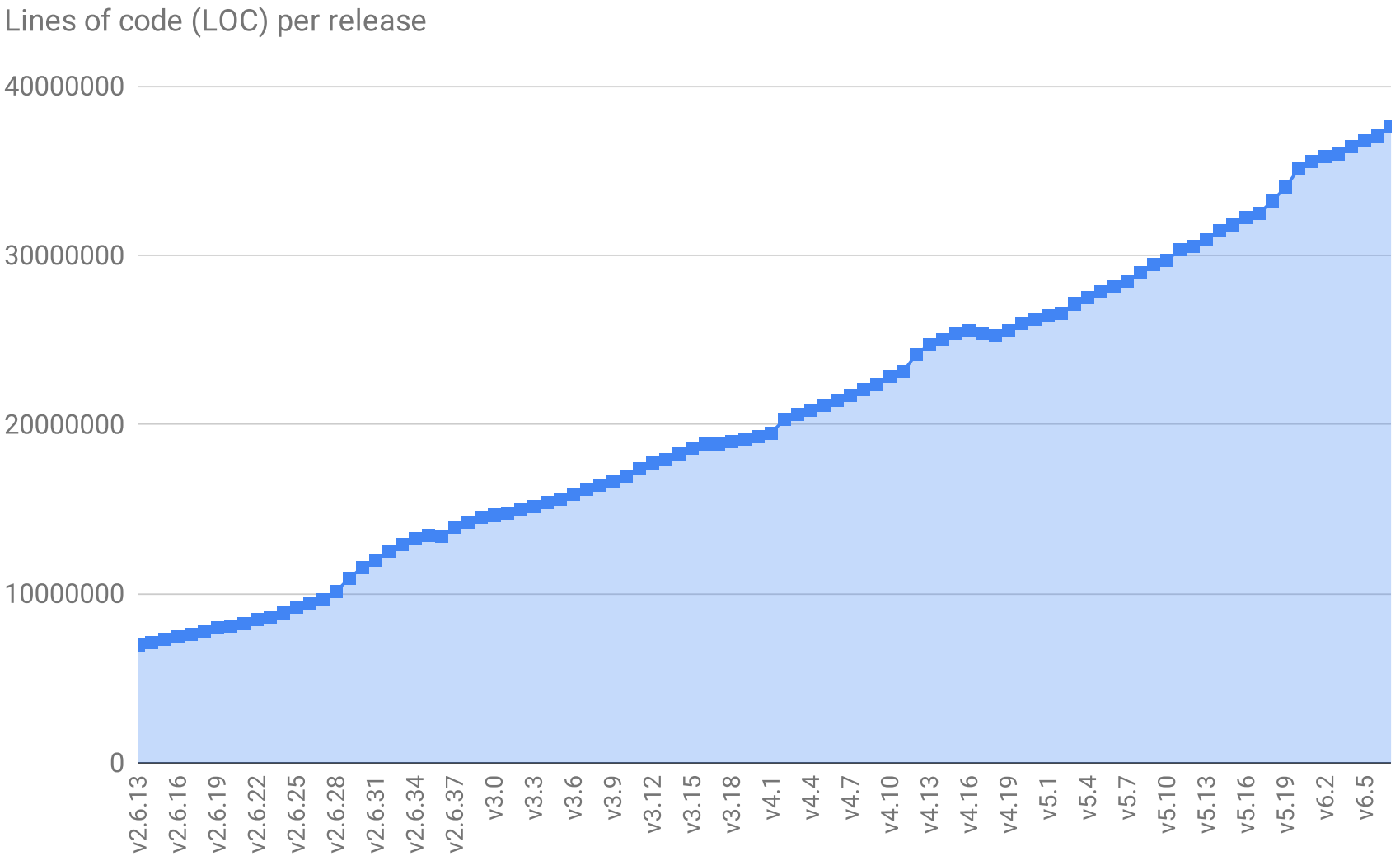
- Motivation pour l'utilisation d'un OS
  - ➔ Réutilisation de services récurrents (E/S, fichiers, …)
  - ➔ Abstraction du matériel (API standardisée)
  - ➔ Partage des ressources
  - ➔ Protection du matériel
  - ➔ Sécurité : protection contre du logiciel malveillant
  - ➔ Robustesse : limitation de l'impact de bugs
  - ➔ …

# Quizz

- Combien de lignes de code contient le noyau Linux 6.7 (8 January 2024) ?

# Quizz

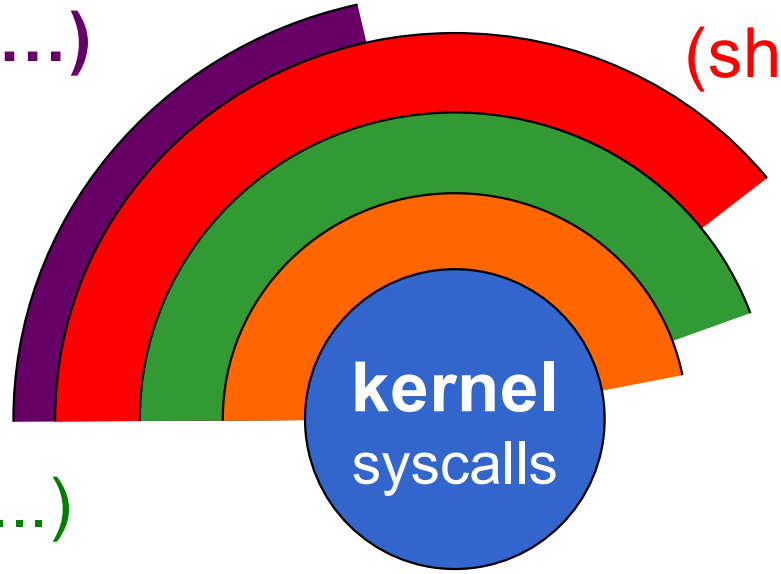
- Linux 6.7:
  - ➔ 37,614,492 lines of code (~ 37M, much of it is driver code)



# Rappel : OS ≠ noyau

**GUI (X11, Desktop, …)**

**Shell / CLI**
(sh, bash, ksh, zsh, ...)




**System tools**
(gcc, init, cron, ld, ...)

**System libs**
(libc, libpthread, ...)

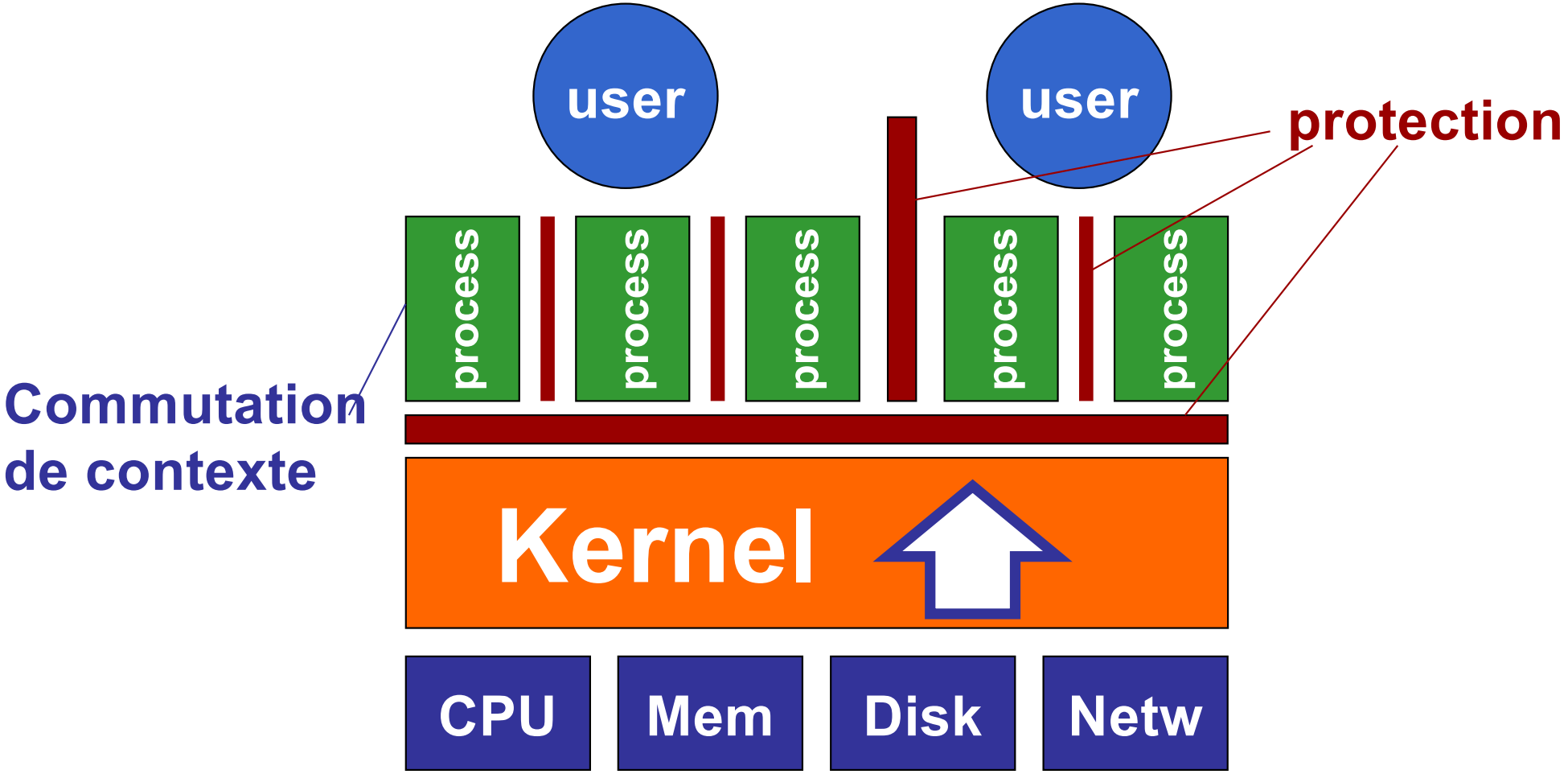
- Noyau = kernel, la partie la plus critique de l'OS

# Plan de l'unité

- Mécanismes de contrôle de l’exécution
- Représentation des processus
- Mise en œuvre des ordonnanceurs
- Mise en œuvre de la synchronisation
  - ➔ Exclusion mutuelle
  - ➔ Sémaphores
  - ➔ Particularités des architectures multi-cœurs

# Noyau d'un OS

- **Contrôle l'exécution** des programmes utilisateurs



# Mécanismes de contrôle de l'exécution

Deux grands objectifs du noyau

- **Protection** : isolation
  - ➔ Système vis à vis de l'usager
  - ➔ Usagers vs usagers
- **Commutation du contexte** d'exécution
  - ➔ Partage des **cœurs** matériels (limités) entre les processus s'exécutant (nombreux)
  - ➔ Partage de la **mémoire** physique (➔ mémoire virtuelle)
  - ➔ Partage des **périphériques d'E/S** (NIC, clavier, écran,…)

# Mécanismes de protection

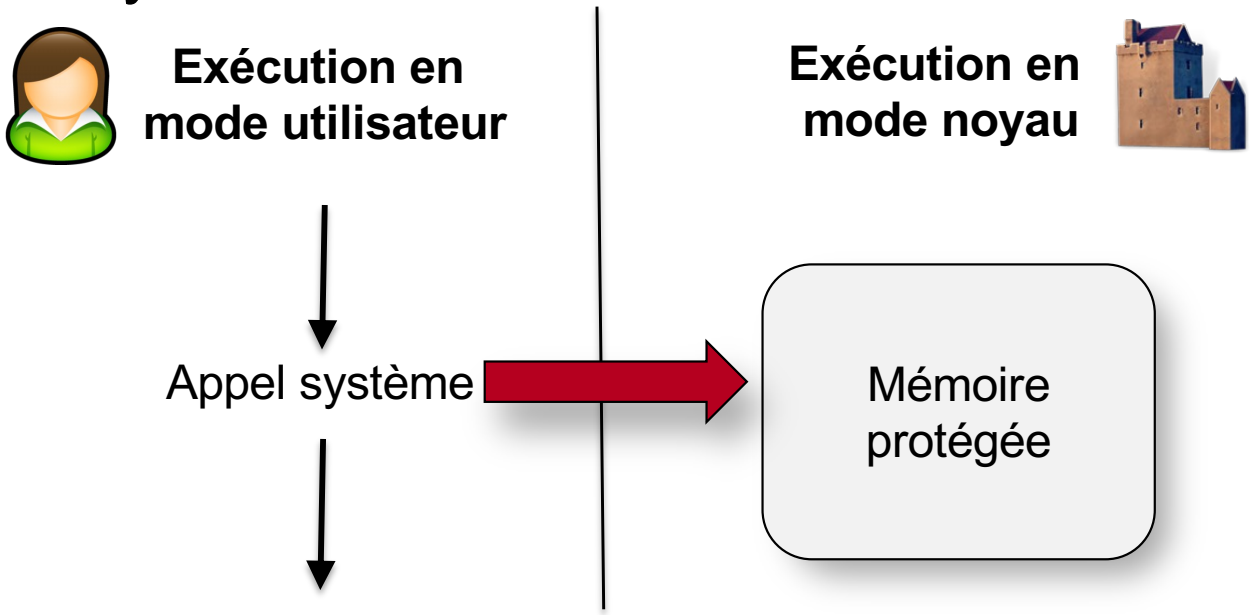
- Contrôle de l'**accès à la mémoire**
- Contrôle de l'**utilisation des instructions**
  - ➔ Exemple : *cli, sti, in, out, hlt (x86)*
- **Modes d'exécution** pour le processeur
  - ➔ **Mode noyau** (superviseur) exécution de toutes les instructions est possible
  - ➔ **Mode utilisateur** (userland) exécution de certaines instructions et accès à certaines zones mémoire) interdit
  - ➔ Mode courant : mot d'état du processeur (registre, SR)
- Conjointement à **l'adressage virtuel** (usagers vs usagers)

# Changement de mode d'exécution
# Appels système

- Réalisation de certaines opérations (entrées/sorties, …) nécessite de passer en **mode noyau**
- Passage en mode noyau doit être contrôlé ⇒ **Instruction spécialisée** de passage en mode noyau, appel système
  - ➔ `int` ou `syscall` en x86,
    `syscall` en MIPS,
    `trap` en 68k

# Changement de mode d'exécution
# Appels système

- Passage en mode noyau + branchement à une adresse fixée




  - ➔ x86/Linux : `syscall` (`int` 0x80 until v. 2.5)
    - **Numéro appel** système dans `rax` (ex 1 = exit)
    - **Paramètres** dans des registres (`rbx`, `rcx`, `rdx`, `rsi` et `rdi`)
    - (section 2 du **man** pour la liste des appels système)

# Rappel : exemple cours SYS

```
SECTION     .data
message:    db  "Hello, World!", 10      ; note the newline at the end
msgLen:     equ $-message

SECTION    .text
  GLOBAL    _start

_start:
  mov       rax, 1                    ; system call for write
  mov       rdi, 1                    ; file handle 1 is stdout
  mov       rsi, message              ; address of string to output
  mov       rdx, msgLen               ; number of bytes
  syscall                             ; invoke operating system to do the write
  mov       rax, 60                   ; system call for exit
  mov       rdi, 0                    ; exit code 0, equiv to xor rdi, rdi
  syscall                             ; invoke operating system to exit
```

```
$ nasm -felf64 helloworld.asm
$ ld helloworld.o
$ ./a.out
Hello, World!
$
```

# Changement de mode d'exécution Appels système

- Remarques
  - **Protection au niveau mémoire** indispensable
    - Ne sert a rien de forcer l'utilisateur à passer par un appel système s'il peut modifier le noyau, …
  - Le traitement d'une **interruption** impose également le passage en mode noyau

# Changement de mode d'exécution
# Changements de contexte

- Contexte d'exécution du processeur
  - ➔ Informations accessibles par le processeur à un instant donné, stockées dans ses registres
    - Mot d'état (SR - FLAGS)
    - Compteur ordinal (PC ou IP)
    - Sommet de pile (SP), pointeur de frame (FP ou BP), (en x86-64 rsp, rbp)
    - Registres généraux (entiers, flottants, etc) … en x86 rax, rbx, rcx, rdx, rsi, rdi et bien d'autres

# Changement de mode d'exécution
# Changements de contexte

- Pourquoi la commutation de contexte ?
  - ➔ Plus de processus que de cœurs
    (un seul jeu de registres par cœur)
  - ➔ Suspension d'un processus → reprise au même point
  - ➔ Opération de commutation de contexte
    - Sauvegarde du contexte d'exécution courant
    - Mise en place d'un nouveau contexte d'exécution
      (restauration du contexte)
  - ➔ Remarques
    - La mémoire n'est pas sauvegardée (cf. mémoire virtuelle)
    - Chaque processus a l'illusion d'un processeur dédié

# Changement de mode d'exécution
# Changements de contexte

- Où sauvegarder le contexte d'exécution ?
  - ➔ En mémoire
  - ➔ A un emplacement ou on peut le retrouver au redémarrage du processus
    - Pile d'exécution du processus
      (+ lien pour retrouver le sommet de pile)
    - Descripteur du processus (voir après)

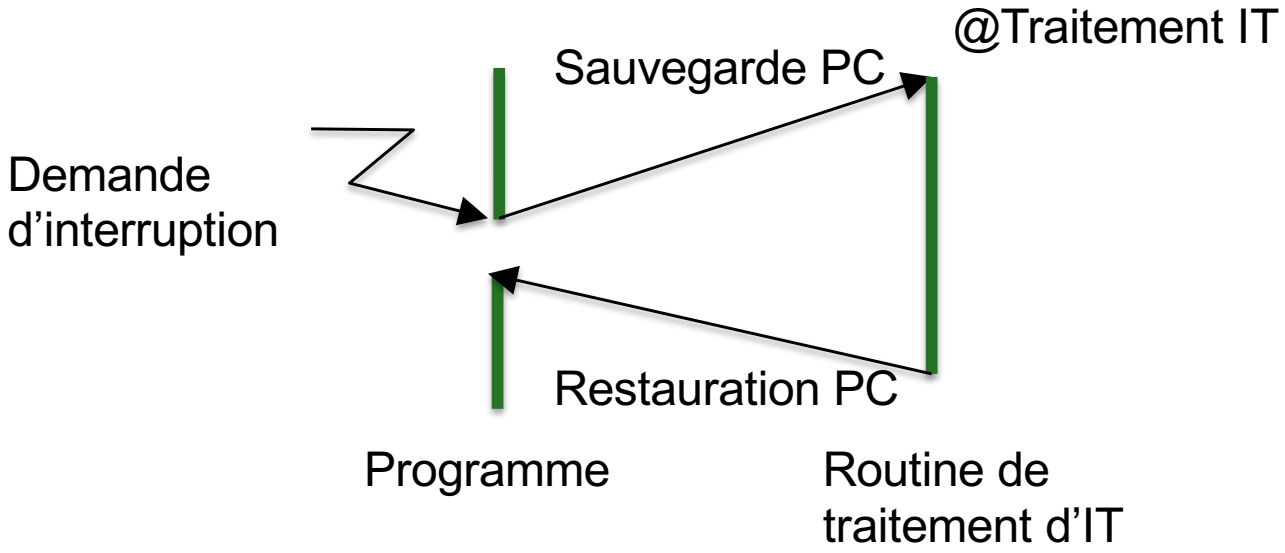
# Mécanismes provoquant une commutation de contexte

- Exception (définition)
  - ➔ On nomme exception tout mécanisme provoquant une commutation de contexte, i. e.
    - Occasionnant l'exécution d'une séquence de code **extérieure** au programme
    - Avec retour **possible** ultérieur

# Types d'exceptions : interruptions

- Cause **externe** au programme, **non demandée** explicitement
- Un niveau d'interruption peut être :
  (interrupt level = interrupt type)
  - ➔ Armé/désarmé (activé/désactivé) : suppression de la source de l'interruption (signal d'interruption)
    - Obtenu par configuration du contrôleur de périphérique
  - ➔ Masqué/démasqué : configuration du processeur pour qu'il n'effectue pas de déroutement en cas de signal d'interruption
    - Stocké dans le mot d'état du processeur
    - Instructions spécifiques (instructions privilégiées **cli/sti**)

# Types d'exceptions : interruptions




- Contexte minimal sauvegardé en cas d'interruption (PC,SR) par le matériel
- Une routine d'interruption n'a pas de contexte
  - ➔ Pas de blocage possible
  - ➔ Pas d'appel système bloquant autorisé

# Types d'exceptions : exceptions matérielles

- Cause **interne** au programme, **non demandée** explicitement
- Détection d'une situation d'exécution anormale
- Exemples
  - ➔ Division par zéro, débordement
  - ➔ Accès à une zone mémoire invalide
  - ➔ Exécution d'une instruction interdite
- Routine de traitement associée (signal)
  - ➔ Remédie à l'anomalie si possible

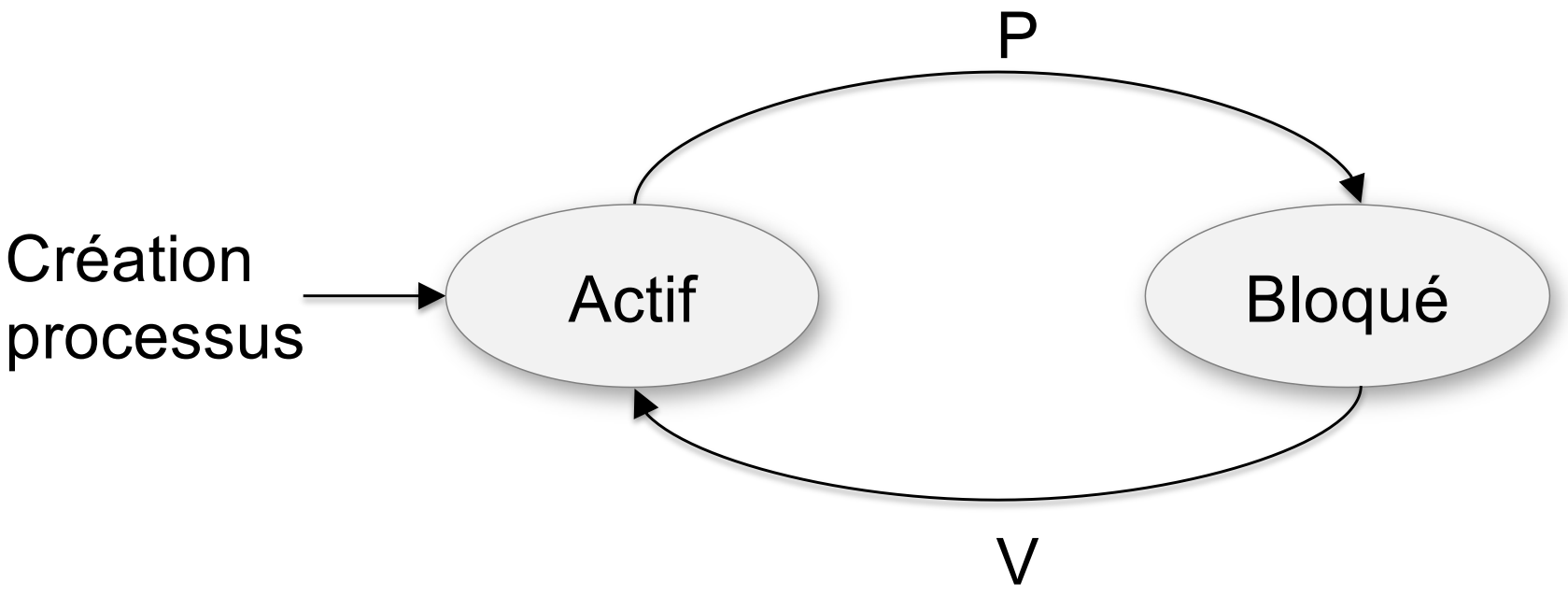
# Types d'exceptions : appels système (syscalls)

- Cause **interne** au programme, demande **explicite**
  - ➔ Passage contrôlé en mode système, schéma d'exécution associé
    1. Sauvegarde du contexte
    2. Exécution de l'appel système
    3. Restauration du contexte
- Remarques
  - ➔ Quel contexte est restauré en 3 ? Dépend de l'appel système réalisé (bloquant, non bloquant)
  - ➔ Contexte restauré au retour d'une interruption ?
    - Processus interrompu ou autre processus ?
    - Dépendant de la mise en œuvre du noyau

# Noyau de synchronisation Représentation des processus

- Etats d'un processus
  - ➔ Etats logiques d'un processus (en faisant abstraction du partage du processeur entre processus)
    - **Actif** : peut logiquement s'exécuter et s'exécute car il dispose d'un processeur dédié pour s'exécuter
    - **Bloqué** : ne peut pas logiquement s'exécuter (attente liée à une synchronisation)

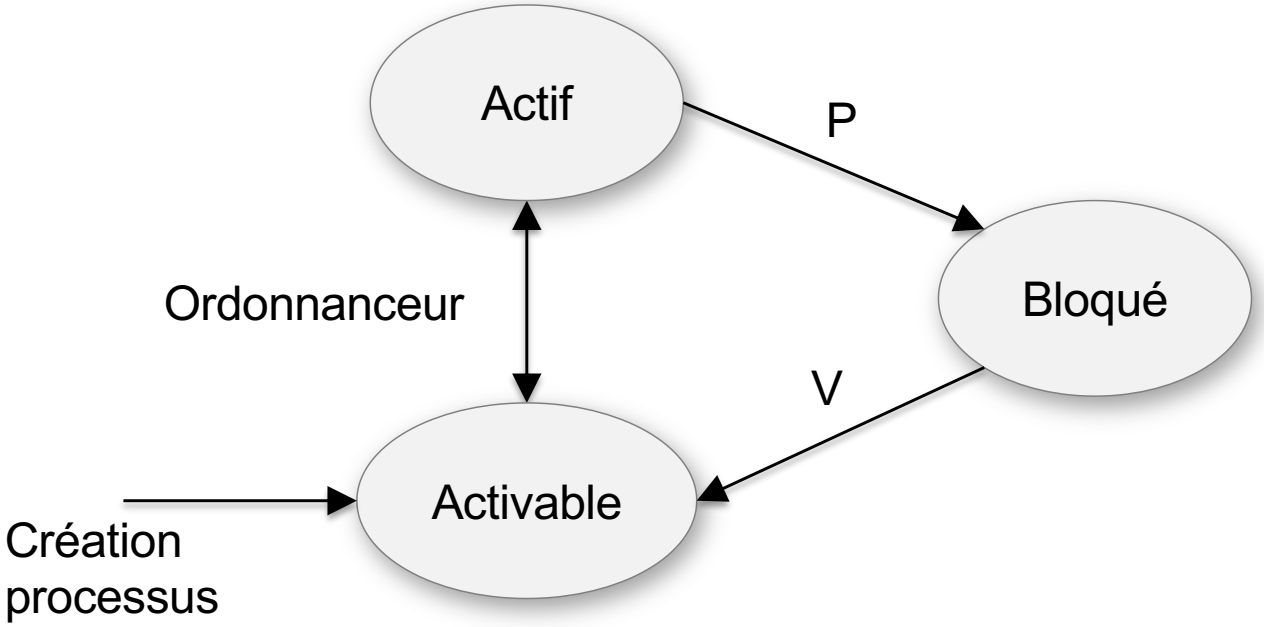
# Noyau de synchronisation
# Représentation des processus



# Noyau de synchronisation
# Représentation des processus

- Etats d'un processus
  - ➔ En supposant un seul cœur disponible
    - Un seul processus actif à un instant donné
    - Découpage de l'état actif en deux sous-états
      - **Actif** : peut logiquement s'exécuter et s'exécute car il dispose à cet instant du processeur réel
      - **Activable** (éligible) : peut logiquement s'exécuter mais ne s'exécute pas car ils ne dispose pas à cet instant du processeur réel
      - **Bloqué** : ne peut pas logiquement s'exécuter
    - NB : plusieurs actifs en multicœurs / multiprocesseurs

# Noyau de synchronisation
# Représentation des processus




➔ Transitions Actif – Activable : ordonnanceur du noyau

# Noyau de synchronisation
# Représentation des processus

- Etats d'un processus affichés
  (commande **`ps`** sous MacOS / Linux)
  - ➔ **I (Idle)** : endormi depuis plus de 20 secondes
  - ➔ **R (Ready/Runnable)** : actif ou activable
  - ➔ **S (Sleeping)** : endormi depuis moins de 20 secondes
  - ➔ **U** : dans une attente non interruptible
  - ➔ **Z (Zombie)** : terminé alors que son père ne l'est pas (ne consomme pas de ressource, juste un descripteur)

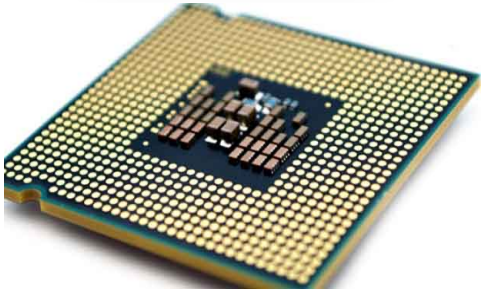
# Contexte d'un processus

- Processus actif
  - ➔ Contexte d'exécution dans les registres d'un cœur, qui le fait évoluer
- Autre processus (activable, bloqué)
  - ➔ Contexte d'exécution sauvegardé en mémoire

# Contexte d'un processus

**Processus P1 (actif)**

PC
rax, rbx,
rcx, etc…



Registres du coeur

**Processus P2 (activable)**

```
4fff: mov   edi,eax
4501: call  40050b <foo>
4506: mov   0x4[rbp],eax
4509: leave
450a: ret
```

PC
rax, rbx,
rcx, etc…

**Processus P3 (activable)**

```
1f8b: cmp [rbp-8],0
1f8f: jne 12 <_main+0x21>
1f95: mov [rbp-12],0
1f9c: jmp 7 <_main+0x28>)
1fa1: mov [rbp-12],1
```

PC
rax, rbx,
rcx, etc…

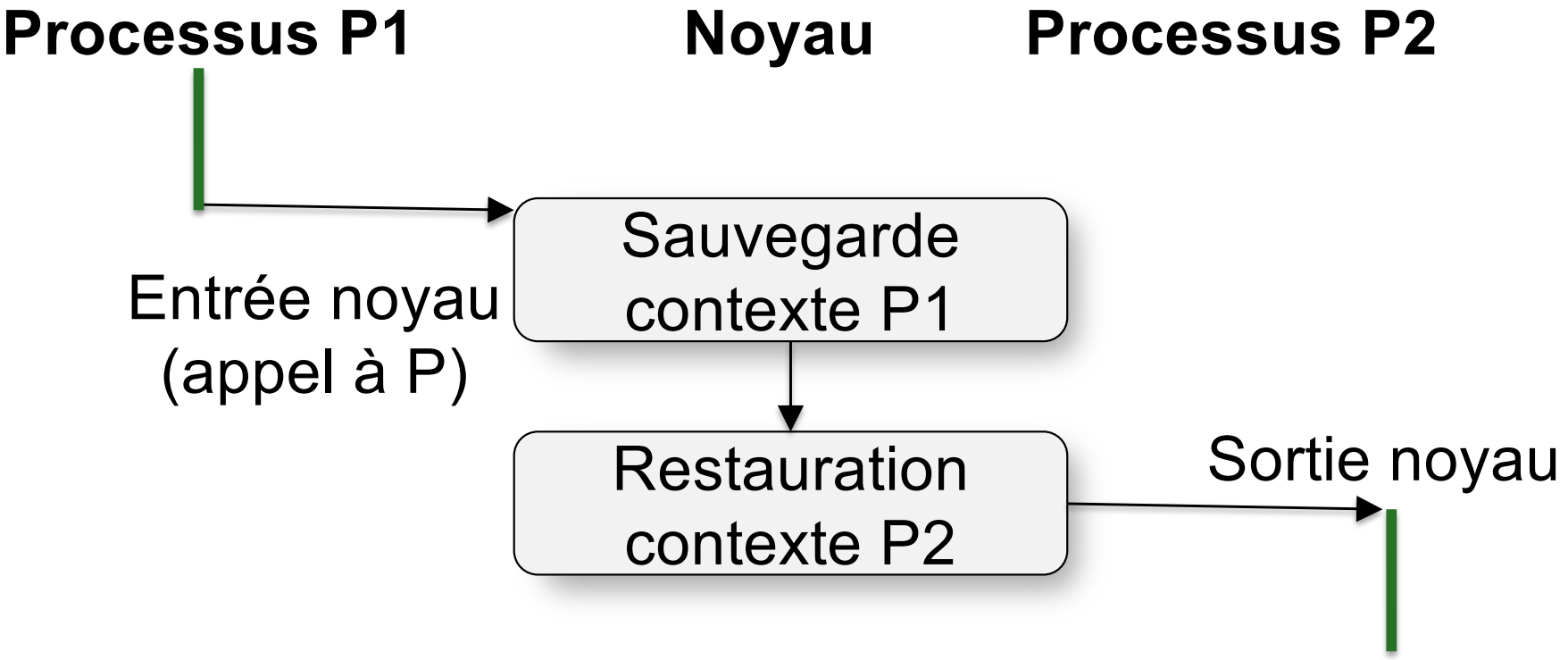
Zones de sauvegarde de contexte (valeurs des registres au moment de la suspension)

# Commutation de contexte

- Association allocation/dé-allocation du processeur
  - ➔ **Perte** d'un cœur par un processus
    - Sauvegarde de son contexte d'exécution
    - Registres du processeur → zone de sauvegarde du contexte
  - ➔ **Allocation** du cœur concerné à un nouveau processus
    - Restauration du contexte sauvegardé au préalable
    - Zone de sauvegarde du contexte → registres du processeur

# Commutation de contexte

- Exemple : commutation de contexte lors d'un appel système bloquant P



# Commutation de contexte

- Remarques
  - C'est **l'ordonnanceur du système d'exploitation** qui sélectionne le processus à activer (ici, P2) parmi les processus activables
  - Contrairement à un appel de fonction, on ne ressort pas du noyau nécessairement par la fonction appelée
  - On peut ne sauvegarder le contexte que lorsque l'on sait que l'appel système est bloquant

# Ordonnancement de processus

- Objectif
  - ➔ Décider parmi les processus prêts lequel exécuter
- Classification
  - ➔ Préemptif / Non préemptif
    - Non préemptif : ne perd le processeur qu'en cas de blocage explicite
    - Préemptif : peut perdre le processeur à tout instant sans l'avoir demandé
  - ➔ Partage de temps (round-robin) : quantum de temps
  - ➔ Priorités
    - Fixes / dynamiques

# Descripteur de processus

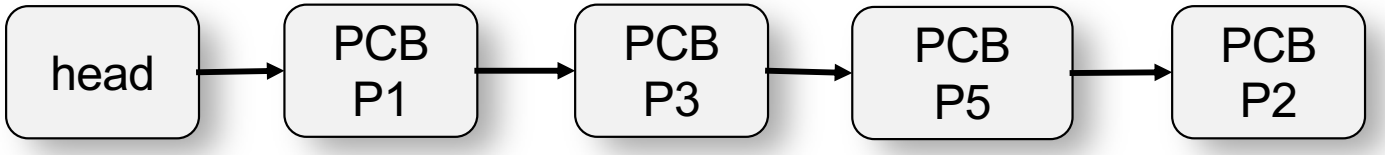
- Descripteur par processus
  (PCB, *Process Control Block*)
  - ➔ Caractéristiques propres du processus
    (algorithme d'ordonnancement, priorité...)
  - ➔ Caractéristiques décrivant son état d'exécution
    (état, temps CPU utilisé...)
  - ➔ Zone de sauvegarde de son contexte (ou son adresse)
- Descripteurs chaînés entre eux pour regrouper les processus de caractéristiques identiques
  - ➔ File des processus activables
  - ➔ File des processus bloqués (ex: sur un sémaphore/verrou)

# Descripteur de processus

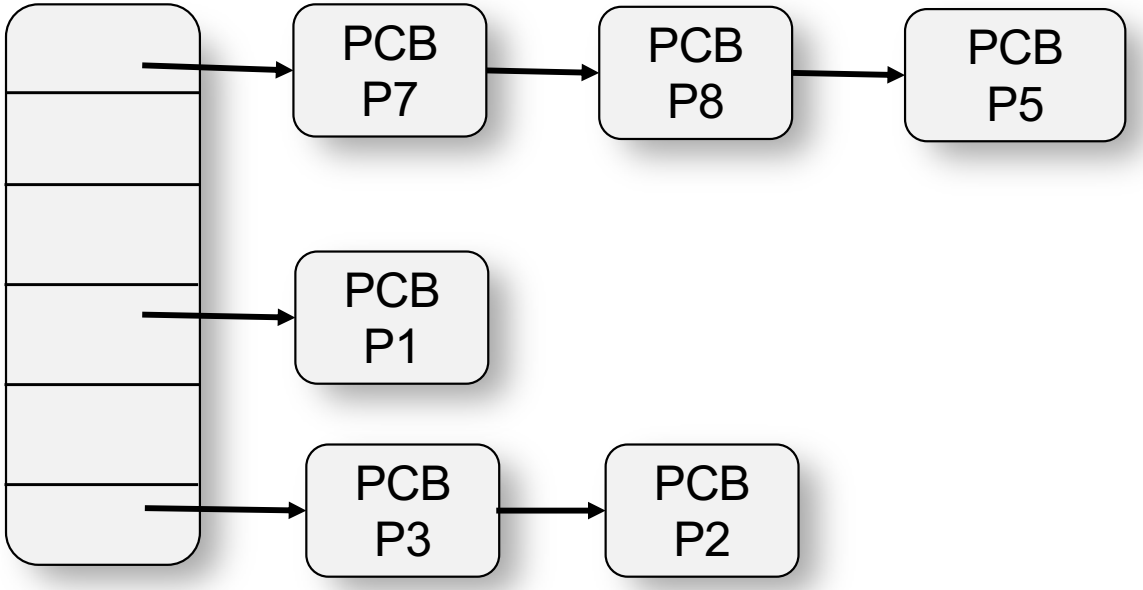
- Remarques
  - ➔ La structure de données dépendra de **l'ordonnancement** mis en œuvre
  - ➔ Dans les systèmes à **processus "lourds"**, le descripteur du processus contient en plus le contexte mémoire du processus (pointeur sur la **table des pages**, cf unité gestion mémoire)

# Descripteur de processus

- Tourniquet (round-robin) sans priorités




- Priorités fixes



# Ordonnancement Linux

- Ordonnancement temps-réel
  - ➔ **SCHED_FIFO**: à base de priorité statique, non préemptif à un niveau de priorité donné
  - ➔ **SCHED_RR**: variante : partage de temps (quantum) à priorité égale
- Ordonnancement généraliste

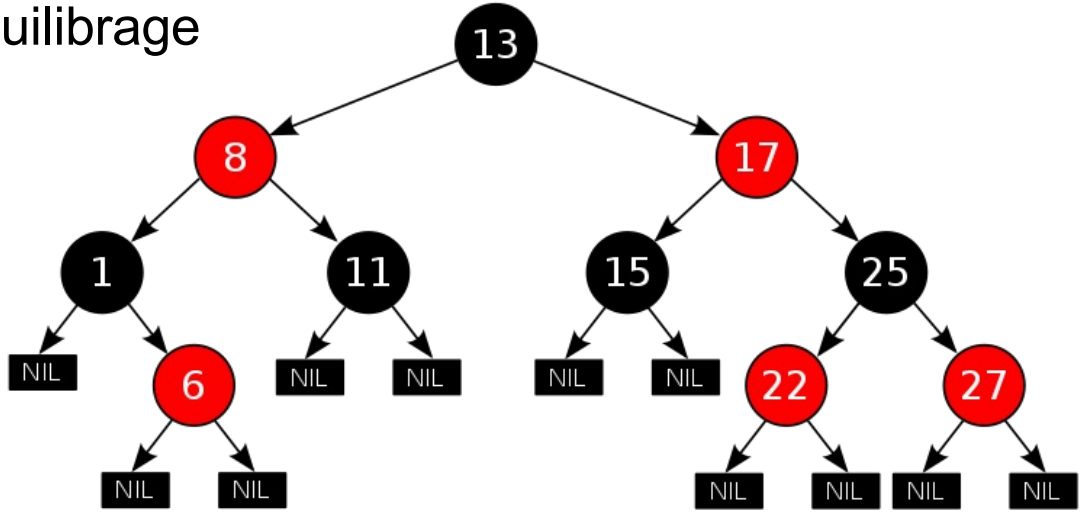
  (- prioritaire que tâches temps-réel)
  - ➔ **SCHED_NORMAL** (ou SCHED_OTHER) : **Completely Fair Scheduler** (CFS) – par défaut, voir détails après
  - ➔ **SCHED_BATCH** : pénalité d'ordonnancement, adapté au traitement par lot
  - ➔ **SCHED_IDLE** : exécution en arrière plan
  - ➔ (SCHED_BATCH et SCHED_IDLE conçues pour les tâches de très faible priorité)

# Ordonnancement Linux : SCHED_NORMAL (CFS)

- Objectif : équité du temps alloué aux processus
- Moyen
  - ➔ Tri des processus selon une valeur représentative du *manque* de ces processus en temps processeur, par rapport à du *multitâche idéal* (équité stricte)
- Structure de données utilisée : arbre rouge-noir

# Ordonnancement Linux : SCHED_NORMAL (CFS)

- Arbres rouge-noir (Red/black tree)
  - Nœuds rouges ou noirs, **racine noire**
  - Quand nœud rouge, fils noirs. Feuilles (nœuds NIL) noires
  - Arbre **binaire** :
    max 2 fils, valeur fils gauche ≤ valeur nœud ≤ valeur fils droit
  - Pseudo équilibré : pire profondeur ≤ 2 * plus petite profondeur
  - Efficace dans le pire cas en insertion, recherche et destruction
    - Arbres binaires « de base » très mauvais dans le pire cas – rééquilibrage



# Exclusion mutuelle (cf. SPP)

- Variables partagées entre plusieurs processus ⇒ manipulation en exclusion mutuelle
  - ➔ En mode utilisateur mais aussi noyau (files de descripteurs de processus, etc.)
- Espace utilisateur : sémaphores (cf. SPP)
- Mais besoin d'un mécanisme de bas niveau
  - ➔ **Problème de récursivité** : on ne peut pas rendre indivisible les opérations sur les sémaphores en utilisant les sémaphores eux-mêmes
  - ➔ **Sémaphores trop lourds** pour être utilisés dans un noyau, ou les sections critiques sont souvent nombreuses et de courte durée

# Exclusion mutuelle
# a. Masquage des interruptions

- Uniquement sur **processeur mono-cœur** (!)
- Permet au processeur d'ignorer (temporairement) les interruptions pendant une période
- Pas d'interruption ⇒ pas de déroutement de l'exécution sur le processeur ⇒ le processus garde le processeur pour lui seul
- Code

```
...
masquer_IT;          (1)
section critique;    (2)
démasquer_IT;        (3)
...                  (4)
```

# Exclusion mutuelle
# a. Masquage des interruptions

- Solution très rapide ⇒ intéressante à un niveau très bas dans le système
- Non généralisable
  - ➔ Aux sections critiques longues : risque de perte d'interruptions
  - ➔ Aux processus utilisateur : problème de sécurité (oubli de démasquer ou démasquage tardif ⇒ risque de compromettre des E/S en cours pour tout le système) - impossible : cli/sti sont privilégiées
  - ➔ Aux systèmes multi-cœur : le masquage ne concerne que le cœur sur lequel l'instruction de masquage est exécutée

# Exclusion mutuelle
# b. Attente active (cf. SPP)

Verrous tournants (spin lock)

- Solutions algorithmiques sans support matériel
  - ➔ Algorithme de Peterson
  - ➔ (utilise uniquement mémoire lire/écrire atomique)
- Solution algorithmiques avec support matériel
  - ➔ Instruction spéciales : test-and-set, compare-and-swap

# Exclusion mutuelle
# b. Attente active (cf. SPP)

Inconvénients verrous tournants :

- Attente active : utilisation 100% d'un cœur
- Si oubli de relâche : cœur perdu

Conséquence

- API uniquement disponible dans le noyau
- Utilisé uniquement pour sections critiques
  - ➔ Très courtes
  - ➔ Garanties de se terminer (pas d'appels bloquants, etc.)

# Exclusion mutuelle
# b. Attente passive (cf. SPP)

- Idée : file d'attente pour les processus bloqués
- Collaboration avec **l'ordonnanceur** du noyaux
  - ➔ Processus dans la file d'attente = bloqués
- Accès à la file d'attente doivent être **atomiques**
  - ➔ Utilisation de verrous tournants
- Inconvénient : blocage = changement de contexte
- Avantage : pas d'attente active !
- Dans le noyaux Linux
  - ➔ Implémentés comme **sémaphores** (généralisent les verrous, voir SPP)

# Spinning ou blocking ?

- Attente passive (sémaphores)
  - ➔ Pas de monopolisation du processeur pendant la phase d'attente
  - ➔ Mais, changement de contexte
- Attente active (spin-locks)
  - ➔ Monopolisation du processeur
  - ➔ Mais, pas changement de contexte
- Spin-locks peuvent être utilisés
  - ➔ Si on sait que la durée d'attente est très courte
  - ➔ Pour l'atomicité de bas niveau en multi-cœurs

# À Retenir

- Rôles principaux du noyau :
  - ➔ Protection (isolation, impacts sur la sécurité et stabilité)
  - ➔ Partage (multiplexage)
- Mode noyau / mode utilisateur
  - ➔ Mécanisme clé des appels systèmes (syscalls)
- Contexte d'un processus
- Commutation de contexte
  - ➔ Causes: interruptions, exceptions, et appels systèmes
- Ordonnancement
  - ➔ Exemple d'ordonnanceurs
- Synchronisation: passive / active