# Les Systèmes d'Exploitation

## Les processus

Gérard Padiou

Département Informatique et Mathématiques appliquées
ENSEEIHT

Septembre 2010

# plan



# Plan



# La notion de processus

## Origine

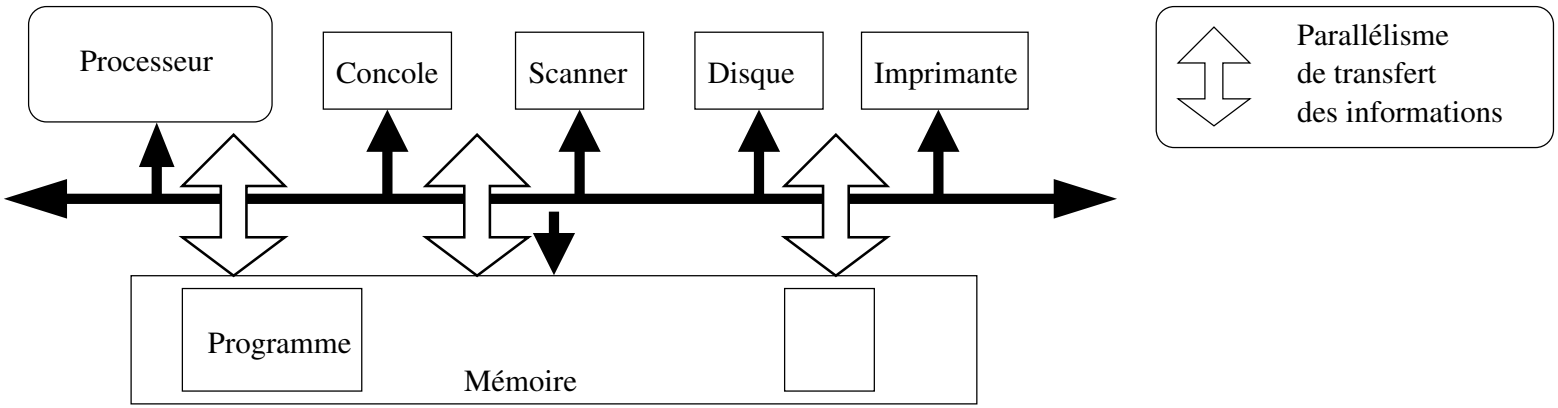
Comment assurer le contrôle de l'exécution d'un programme ?

### Propriétés à garantir

- Le confinement de ces exécutions : contrôle fin des interactions, interactions dues au partage de ressources ;
- Mais aussi permettre des interactions contrôlées entre programmes :
  ☞ communication « producteur-consommateur »
- La robustesse du système face aux erreurs ;
- Un environnement d'exécution portable : indépendance vis-à-vis du matériel ; ☞ Machine « système »
- Le parallélisme : plusieurs programmes en cours d'exécution : ☞ partage équitable des ressources entre ces exécutions ;

# La notion de processus

## Le parallélisme




- La configuration matérielle autorise le parallélisme des flux d'information entre mémoire centrale et périphériques ;
- Un programme en cours d'exécution peut être bloqué logiquement :

  ☞ Idée : continuer un autre programme

# Une définition avec une vision « système d'exploitation »

## Processus = Exécution d'un programme

- Pour le noyau, une structure de donnée comme une autre ...
  ☞ Table des descripteurs de processus

### Rôles d'un processus

- Unité d'exécution ;
- Unité d'allocation de ressources :
  ☞ concurrence d'accès à une ressource critique
- Unité de coopération, de communication :
  ☞ partage de ressources
- Unité comptable : un processus travaille pour un usager ;
- Unité de protection.

# Gestion des processus

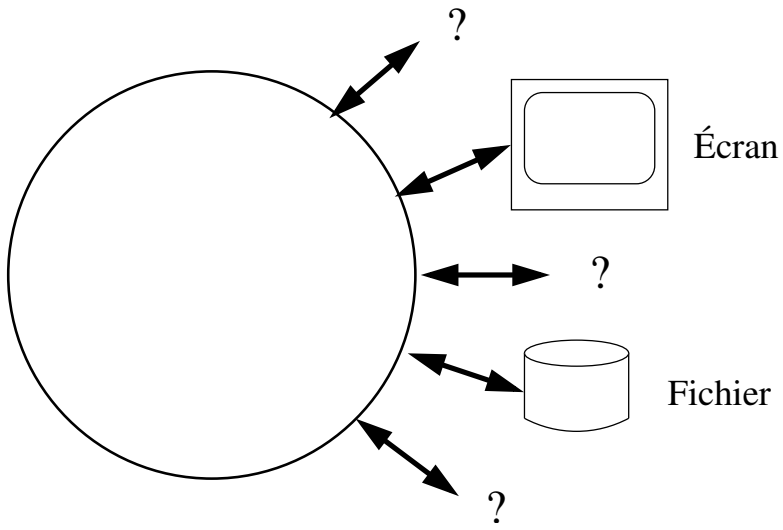
## L'environnement d'exécution d'un processus

- Le processus s'exécute dans l'environnement support fourni par le noyau

  ☞ Machine « système »

- Un processus est un client vis-à-vis du noyau : il demande des services par appel de primitives : requête d'entrée/sortie, requête mémoire, attente d'un événement, ...
- Un processus communique avec son environnement :
  - Par interruptions ou déroutements : ☞ signaux Unix
  - Par flots de données : ☞ connexion par canaux

# Environnement de communication par flots




- Un processus peut « ouvrir/fermer » un ensemble de connexions à des ressources sources ou puits de données via une notion de canal, de port de communication ;
- Un processus « hérite » d'un environnement de communication standard lorsqu'il est créé.

# Environnement de communication par flots

## Flots de données et redirections

- Les ressources physiques (fichiers, imprimante, clavier, fenêtre, etc) sont abstraites via une interface générique de « flots » de données
- Le processus ouvre/ferme des flots de données dynamiquement via des canaux, ports
- Possibilité de redirection : un flot peut être reconnecté à (redirigé sur) un canal différent.

# Sémantiques de communication par flots

## Modes de fonctionnement des primitives d'échange

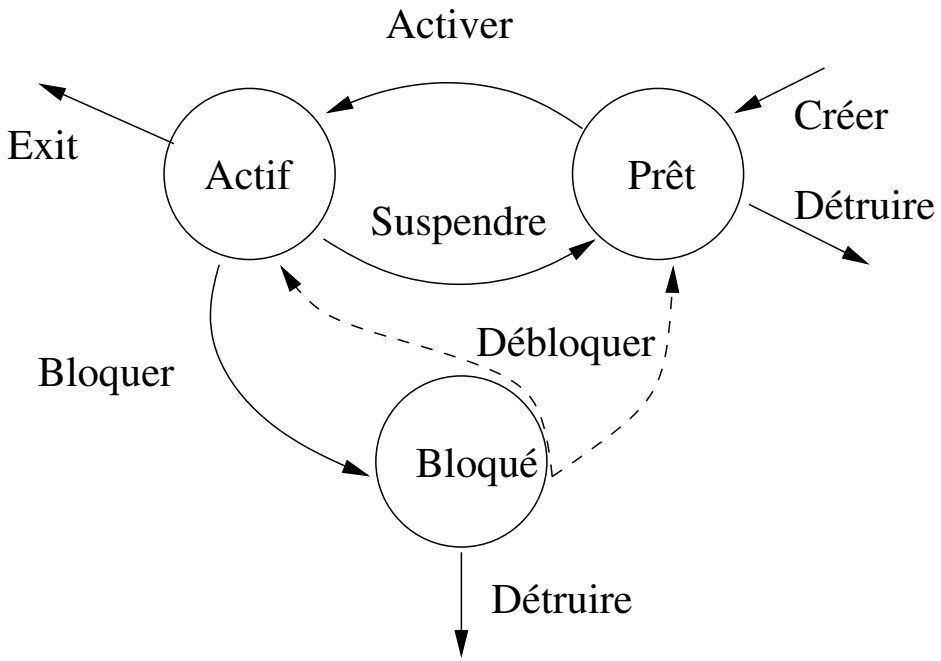
- Mode immédiat (*Best-effort*) : la primitive exécute immédiatement l'opération demandée dans la mesure du possible (pas de blocage) :
  ☞ Résultat : Echec, Exécution sans effet, partielle ou totale
- Mode synchrone (procédural) : la primitive implante un comportement client/serveur classique : elle se termine lorsque l'opération complète a pu être exécutée (ou a échoué).
- Mode asynchrone : la primitive implante un comportement client/serveur asynchrone : elle dépose une requête et se termine. Une opération ultérieure devra tester la terminaison effective (ou l'échec) de l'opération.

# Plan



# Une vision abstraite d'un processus

## Son graphe de transitions d'état




- Etat prêt : Exécution suspendue ;
- Etat actif : Exécution effective du programme ;
- Etat bloqué : Exécution bloquée sur condition logique.

# Problème de base

## Comment suspendre un processus ? le continuer ultérieurement

- Capter un état stable du programme en cours d'exécution (entre 2 instructions) : PC, registres, indicateurs, pointeur de pile, ...

  ☞ Contexte d'exécution

- Suspendre/Continuer un processus :

  $\equiv$ Sauvegarder/Restaurer son contexte d'exécution.

# Gestion des processus

## Quelques principes de base

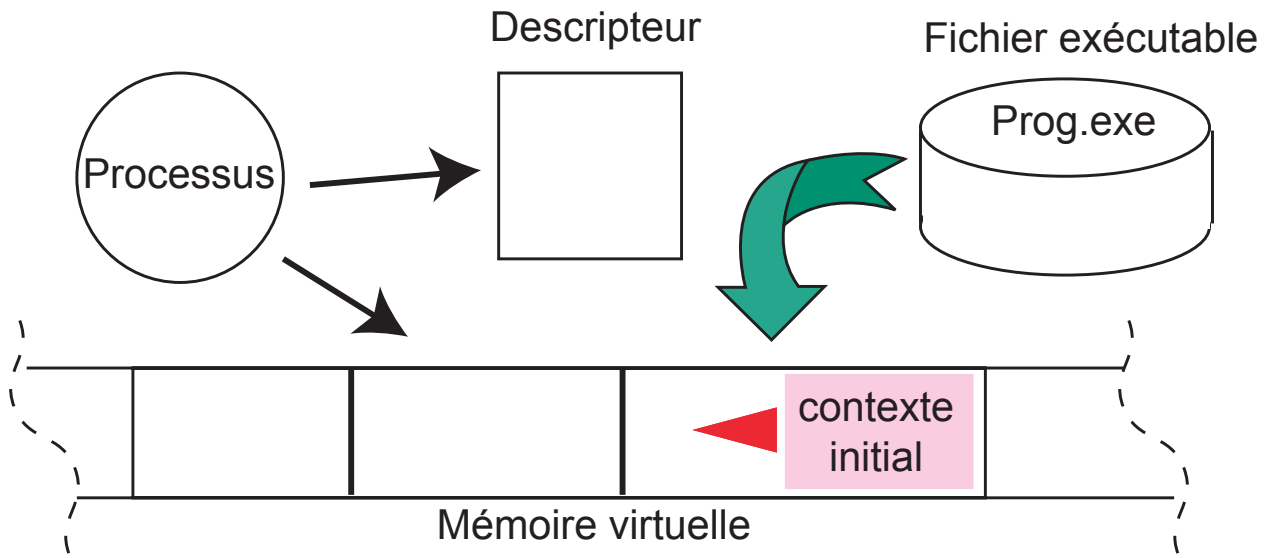
- Création de « père en fils » ;
- Le noyau attribue un identifiant unique au processus ;
- Un processus « racine » initial unique ;
- Gestion d'un arbre de processus (relation père $\rightarrow$ fils) ;
- Notion de priorité ;
- Un processus « roule », s'exécute toujours pour un usager ;
- Parallélisme interne possible (☞ Notion de thread) ;
- Limite sur le nombre de processus créés (descripteurs) ;
- Existence de processus « démons » (« système »).

# Primitives de gestion des processus

## Jeu de primitives génériques

- Créer un processus :
  CreateProcess de Windows, fork d'Unix
- Terminer un processus :
  ExitProcess de Windows, exit d'Unix
- Bloquer/débloquer un processus :
  masquées dans d'autres primitives
- Commuter de processus.

# La représentation d'un processus



# La terminaison d'un processus

Que se passe-t-il en « fin de programme principal » ?

## Un problème de point de contrôle

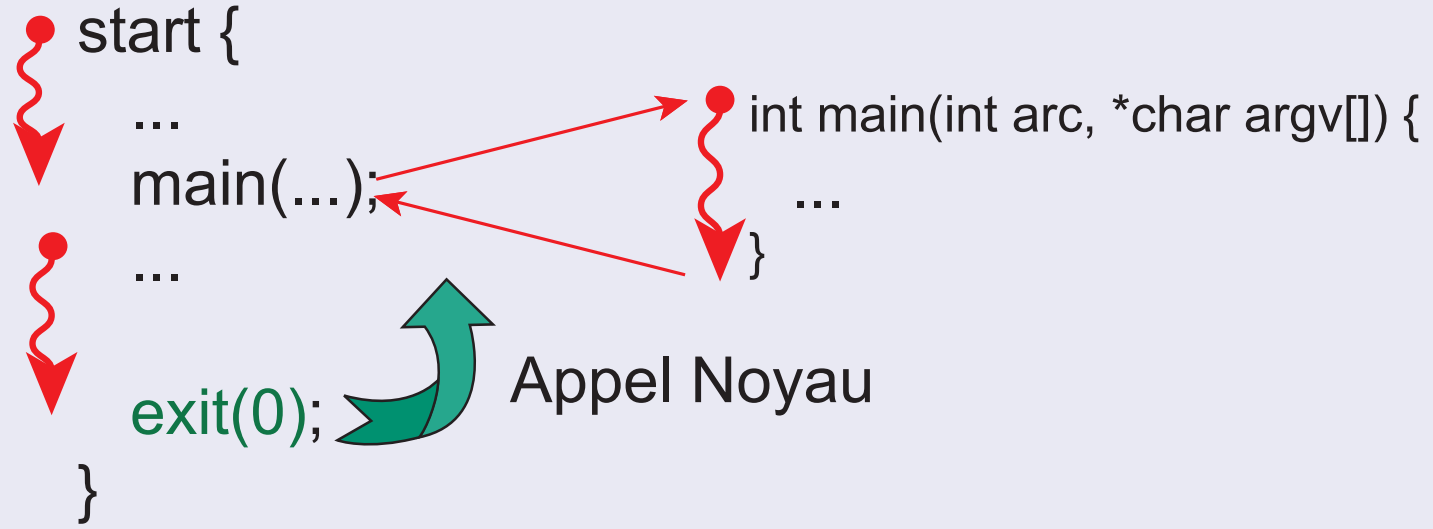
```
int main(int arc, *char argv[]) {
    ...
}
```

?

# La terminaison d'un processus

## Notion d'enveloppe

### Par défaut : un appel caché de primitive



# Primitives de gestion des processus

## Exemple Windows : Création d'un processus

```
BOOL CreateProcess (
 LPCTSTR lpApplicationName, // → programme exécutable
 LPTSTR lpCommandLine, // → ligne de commande
 LPSECURITY_ATTRIBUTES lpProcessAttributes
 LPSECURITY_ATTRIBUTES lpThreadAttributes
 BOOL bInheritHandles, // indicateurs d'héritage
 DWORD dwCreationFlags, // priorité, nouvelle fenêtre,...
 LPVOID lpEnvironment, // → environnement
 LPCTSTR lpCurrentDirectory,
 LPSTARTUPINFO lpStartupInfo, // fenêtre, redirections
 LPPROCESS_INFORMATION lpProcessInformation // résultat
) ;
```

# Primitives de gestion des processus

## Exemple Windows : Création d'un processus (suite)

Informations résultats de la création d'un processus :

```
typedef struct _PROCESS_INFORMATION{
  HANDLE hProcess ; //
  HANDLE hThread ;
  DWORD dwProcessId ;
  DWORD dwThreadId ;
} PROCESS_INFORMATION, *LPPROCESS_INFORMATION ;
```

# Primitives de gestion des processus

## Exemple Windows : Fin d'un processus

- Fin d'un processus par appel explicite d'une primitive Exit :

```
VOID ExitProcess(
    UINT uExitCode // code d'exit
);
```

- Tester l'état d'un processus :

```
BOOL GetExitCodeProcess(
    HANDLE hProcess, // processus cible
    LPDWORD lpExitCode // → résultat
);
```

- Obtenir son identifiant unique :

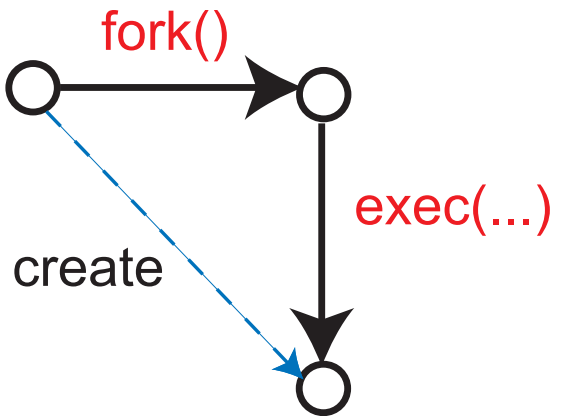
```
DWORD GetProcessId(
    HANDLE Process
);
```

# Primitives de gestion des processus

## Exemple Unix : Duplication d'un processus et commutation de programme

### Séparation en 2 primitives

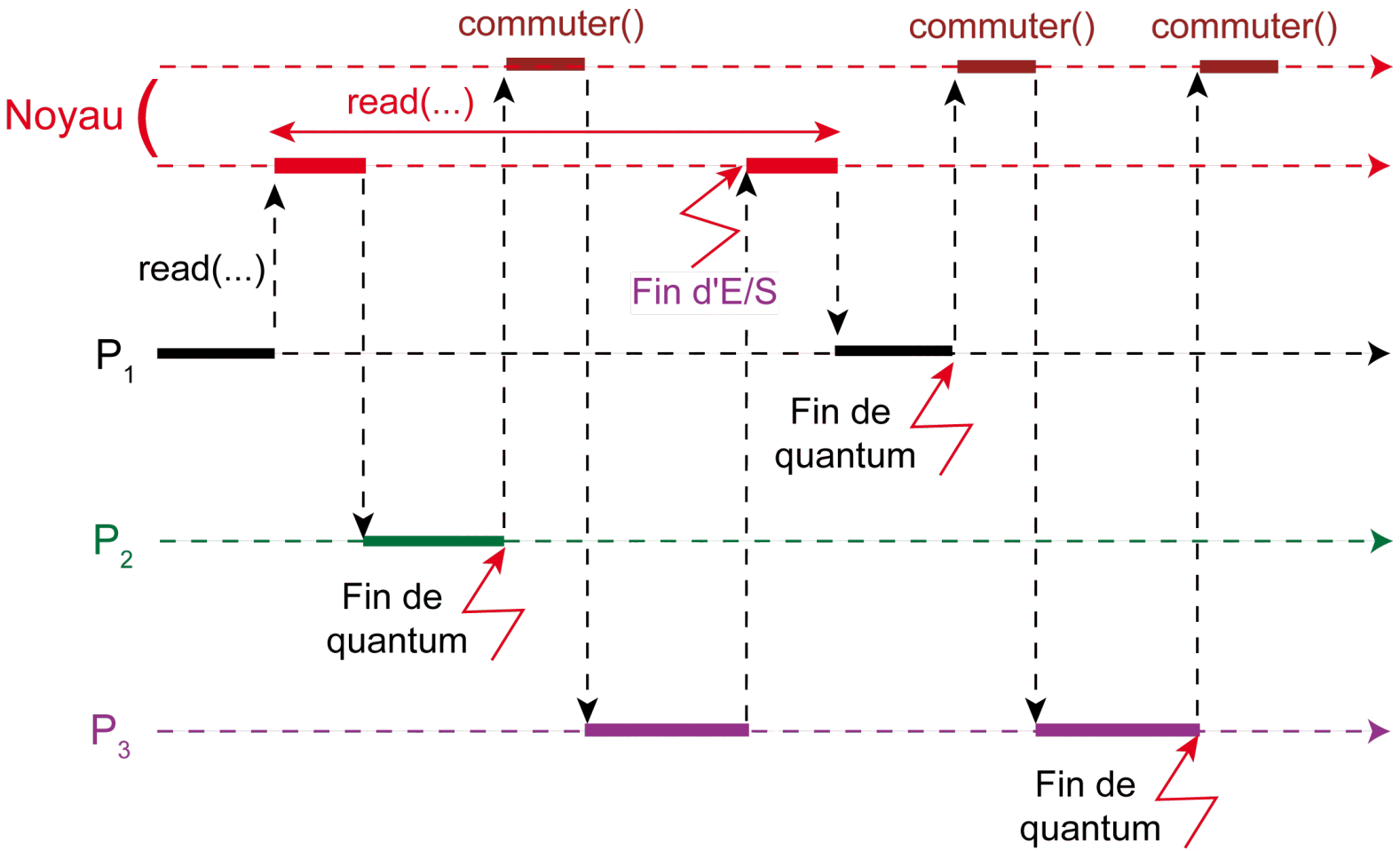
- Création d'un processus fils :
  ☞ Héritage implicite
- Commutation de programme :
  ☞ Un processus peut
  ...changer de programme




### Exemple

```
if (fork()) {
  /* code exécuté par le fils */
  exec("prog_fils",...);
} else {
  /* code exécuté par le père */}
```

# La commutation des processus

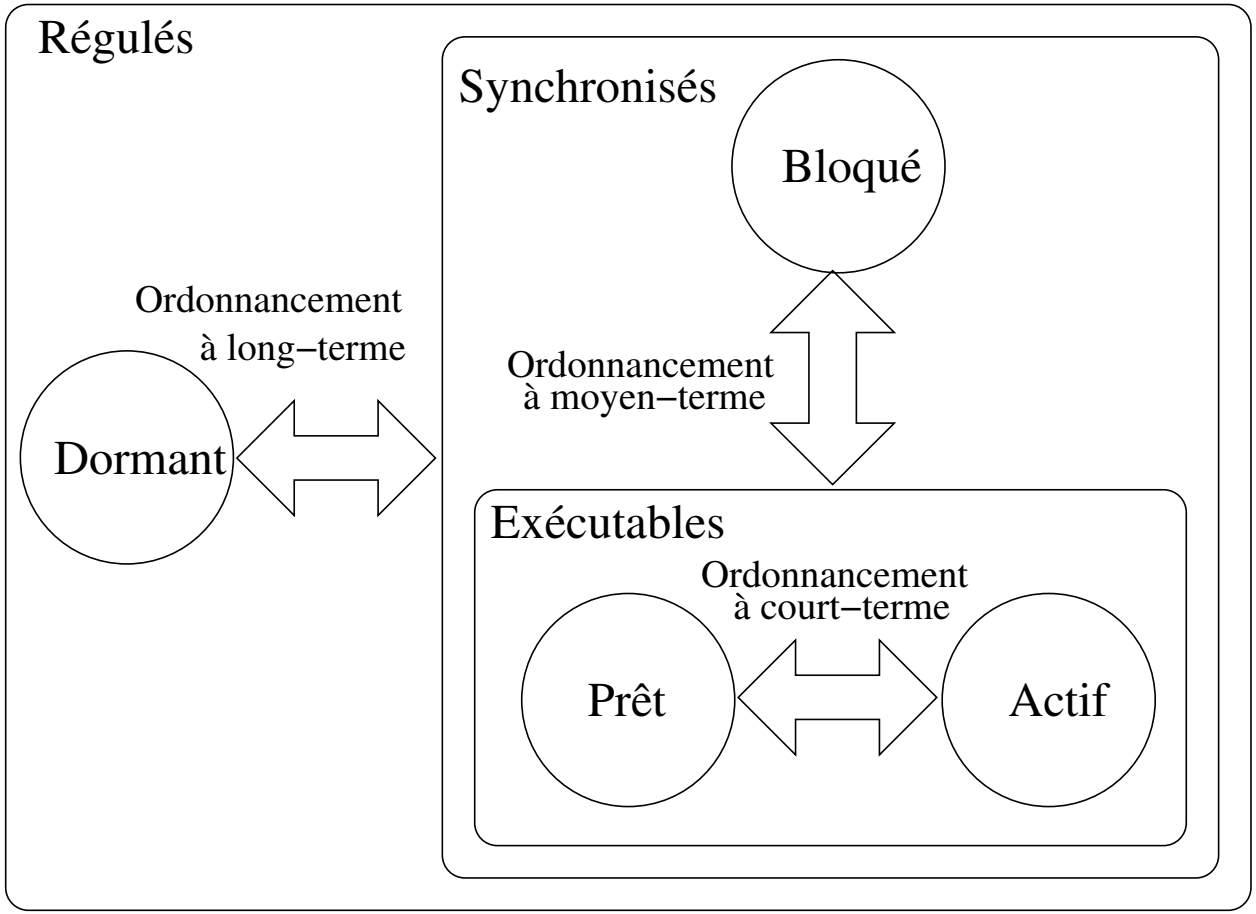


# Plan



# Ordonnancement des processus

## Les différents niveaux d'ordonnancement



# Ordonnancement des processus

## Les différents niveaux d'ordonnancement

### Compétition entre groupes pour obtenir :

- Les *exécutables* ☞ du temps processeur
- Les *synchronisés* ☞ lorsqu'une condition logique qu'ils attendaient devient vraie ;
- Les *régulés* ☞ pour le droit de commencer.

### Gestion de files d'attente de processus

- File des processus prêts ;
- Files de processus bloqués associées à des conditions ;
- File(s) de requêtes de création de processus nouveaux.

# Ordonnancement des processus

L'ordonnancement à court terme ≡ allocation des processeurs

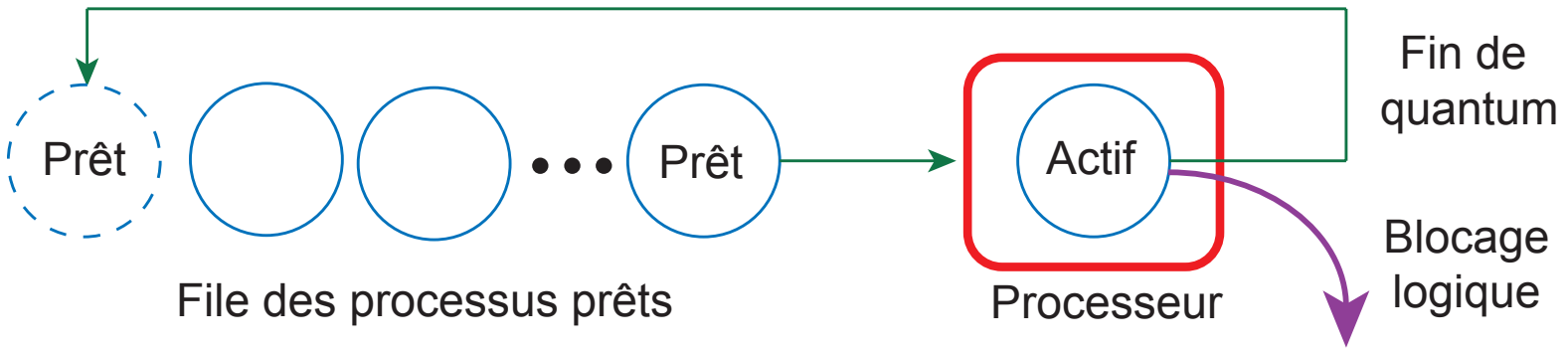
## Les processus exécutables

- Fonction : gérer l'ensemble des processus exécutables ;
- Objectif : Partager équitablement le temps processeur entre les processus selon éventuellement des critères de priorité, d'urgence.
- Assure l'allocation d'un processeur à un processus prêt
- Gère les transitions prêt $\leftrightarrow$ actif
- Plusieurs stratégies d'ordonnancement possibles
  - Cas des systèmes classiques : quantum et tourniquet
  - Cas des systèmes Temps réel : échéance et priorité.

# Ordonnancement à court terme

## La stratégie du tourniquet (round-robin)

- Notion de quantum ≡ durée maximale d'activité continue ;
- Préemption du processeur au profit d'un autre processus prêt
  - soit en fin de quantum ;
  - soit sur blocage logique du processus actif.
- Adaptation possible de la durée du quantum au profil d'exécution (calcul ou entrées/sorties).



# Ordonnancement à court terme

## Stratégies orientées Temps Réel

### Processus périodiques appelés tâches

- Stratégie RMS (Rate Monotonic Scheduling) : choix du processus le plus prioritaire : chaque processus (tâche) a une priorité fixe attribuée selon l'ordre croissant de leur fréquence d'exécution ;
- Stratégie EDF (Earliest Deadline First) : choix du processus prêt ayant l'échéance la plus proche ;

# Ordonnancement à moyen terme

## Les processus synchronisés

- Fonction : gérer l'ensemble des processus synchronisés ;
- Objectif : Assurer le réveil équitable des processus bloqués lorsqu'une condition logique devient vraie et permet de reprendre leur exécution ;
- Gère les transitions bloqué $\leftrightarrow$ prêt (ou actif)
- À une condition de blocage, peut correspondre plusieurs processus bloqués :
  ☞ files de processus bloqués sur une condition

☞ Mécanismes de synchronisation

# Ordonnancement à long terme

## Les processus régulés

- Fonction : gérer l'ensemble des processus régulés ;
- Objectif : Equilibrage de charge en contrôlant la création de nouveaux processus ;
- Produit des processus exécutables (dans l'état prêt) ;
- Idée de base : création possible si les ressources libres qui pourront être allouées au processus sont suffisantes :
  ☞ exemple : notion de working set

# Conclusion

## Concept de processus

- Concept de base des systèmes d'exploitation ;
- Deux problèmes de base : la concurrence et la coopération entre processus éventuellement distants :
  ☞ Synchronisation des processus
- Concept formalisé sous diverses formes, notamment les algèbres de processus.