Université de la Manouba
Institut Supérieur des Arts Multimédias de Manouba

Cycle d'Ingénieur

Informatique Appliquée au Multimédia
1ère année

Chapitre 3

GESTION DE PROCESSUS

N. Ben Ali

Année Universitaire

Plan

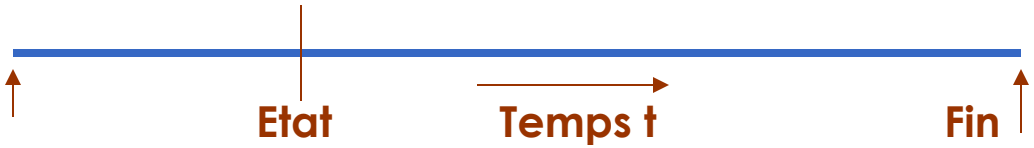
- Notions Fondamentales
- L'ordonnancement des processus
- Synchronisation des processus

Notions Fondamentales

1. Notion de processus

1.1 Définitions

- Qu'est ce qu'un processus ?
  - Un programme **en cours** d'exécution par un **processeur**
- Plus formellement :
  - Entité **dynamique** représentant l'exécution d'un programme sur un processeur
- Un processus est donc :
  - Créé à **un instant donné**, a un **état** qui évolue au cours du temps et qui disparaît, en général, au bout d'un temps fini



1. Notion de processus

1.1 Définitions

- Système informatique :
  - **Plusieurs programmes** en mémoire
  - Le processeur exécute les instructions tantôt pour l'un tantôt pour l'autre
- Intérêt de la notion de processus
  - **Abstraction** de la notion **d'exécution séquentielle**
    - Exécution indépendante de la disponibilité effective d 'un processeur physique
  - Représentation des activités parallèles et de leurs interactions
- Exemples de processus :
  - L'exécution d 'un programme
  - Copie d 'un fichier sur disque
  - la transmission d 'une séquence de données sur un réseau
- Un processus correspond à un seul programme en cours d'exécution ?
  - Non, il peut être à plus et moins
  - + : plusieurs processus peuvent être dérivés d'un même programme
  - - : un seul programme peut utiliser plusieurs processus
    - la commande de compilation cc -- exécute beaucoup de choses

1. Notion de processus

1.2 Programme vs processus

- Un programme **est une suite d'instructions :**
  - objet statique
- Un processus **est un programme en exécution:**
  - objet dynamique : programme + contexte
  - Instance dynamique d'un programme et incarne le fil d'exécution de celui-ci dans un espace d'adressage protégé
- Un programme peut être exécuté plusieurs fois dans des conditions différentes
- **Mode d'exécution**
  - Exécution simultanée de copies d'un même programme.
  - Exécution simultanée de la même copie d'un même programme (« réentrance »).
  - Cas intermédiaires : partager seulement le code, les données en lecture seule.
- Vous pouvez utiliser le même programme que moi, mais ca ne sera pas le même processus que moi
- Même différence qu'entre classe d'objet et instance d'objet

1. Notion de processus

1.3 Petite analogie

- Informaticien fait un gâteau d'anniversaire pour sa fille
  - programme ⇒ recette de cuisine
  - données entrées ⇒ ingrédients
  - ressources nécessaires ⇒ ustensiles
  - processeur ⇒ informaticien
  - processus ⇒ activité de transformation des ingrédients en gâteau
- Fils de l'informaticien se fait piquer par une abeille
  - interruption du travail ⇒ sauvegarde de l'état du processus
  - programme ⇒ livre de première urgence
  - processus ⇒ soin à apporter, calmer son fils
- Reprise du travail de cuisinier
  - restitution de l'état du processus

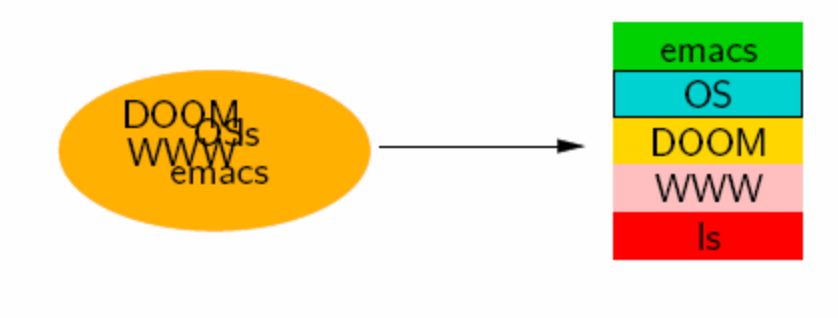
1. Notion de processus

1.3 Petite analogie

- Fille de l'informaticien annonce de nouveaux invités
  - fabriquer 2 gâteaux
  - 2 processus distincts sur le même programme
  - une seule recette ⇒ programme réentrant
  - séparation des données propres de chacun des processus
  - zones de variables distinctes

2. Systèmes Multitâche

- L'ordinateur a des activités différentes :
  - Comment les faire cohabiter simplement ?
  - L'OS s'occupe de chacun de la même façon, chacun ne s'occupe que de l'OS

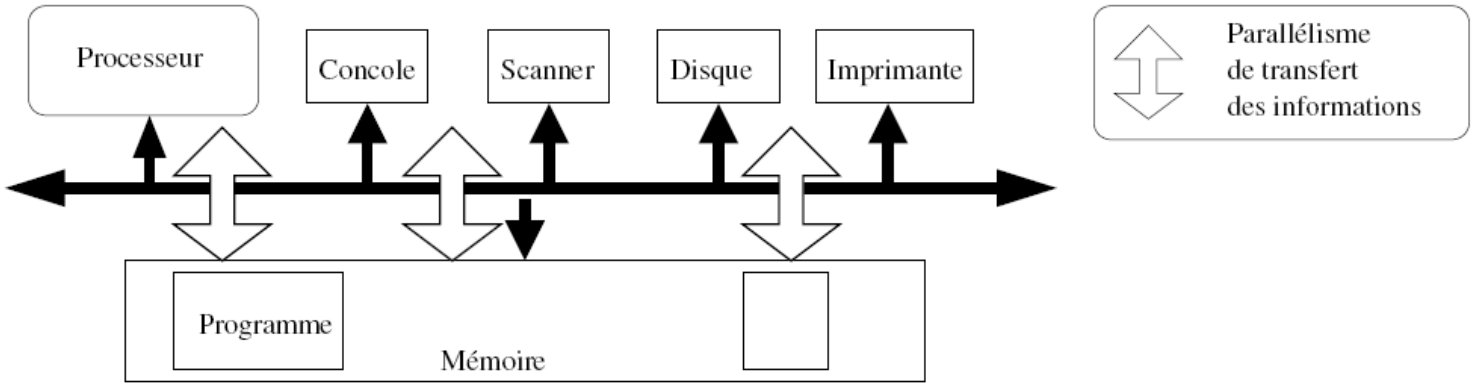


  - La décomposition est une réponse classique à la complexité
    - Intérêt des processus

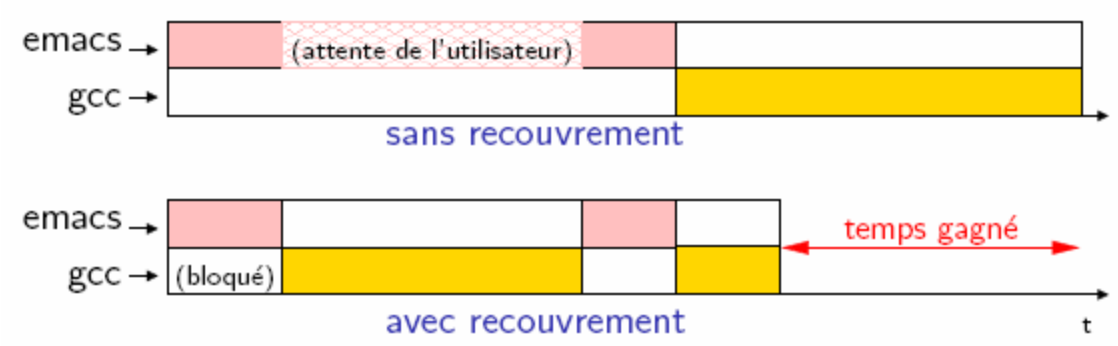
2. Systèmes Multitâche

2.1 Motivations

- 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 : Pourquoi ne pas continuer un autre programme



- Les **communications** bloquent les processus :
  - communication au sens large : réseau, disque, utilisateur, autre programme)
  - ⇒ recouvrement des calculs et des communications



- Que faire quand deux processus sont prêts à s'exécuter ?
  - Si deux processeurs, tout va bien.

  - Sinon, FCFS ? Mauvaise interactivité!

  - Pseudo-parallélisme = chacun son tour

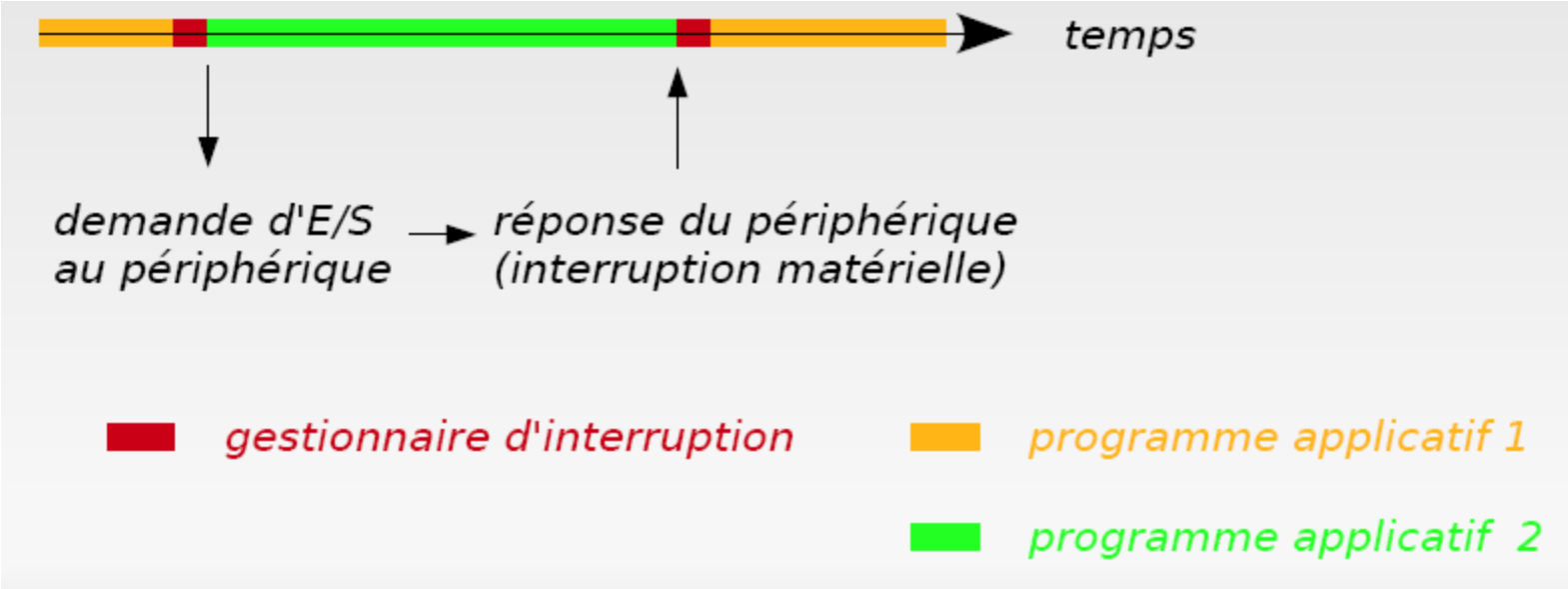
  - Autre exécution pseudo-parallèle

- **Multiprogrammation** & **Pseudo-parallélisme :**
  - Un OS doit, en général, traiter plusieurs processus en même temps
  - Il y a un seul processeur (la plupart du temps)
- **Entrelacement** des exécutions :
  - simuler une **exécution parallèle**:
    - À tout moment l'OS ne traite qu'un seul processus à la fois, il s'interrompt et passe au suivant.
    - La **commutation** étant très rapide
    - Illusion d'un traitement simultané

2. Systèmes Multitâche

2.2 Parallélisme et pseudo parallélisme

- Quand est ce qu'il y a commutation ?
  - Temps partagé :
    - 2 ou plusieurs processus s'exécutant simultanément
    - Pseudio-parallélisme
- Ou bien lorsqu'un processus demande une opération d'entrée/sortie, un autre processus peut s'exécuter en attendant que l'opération aboutisse



- **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, ...

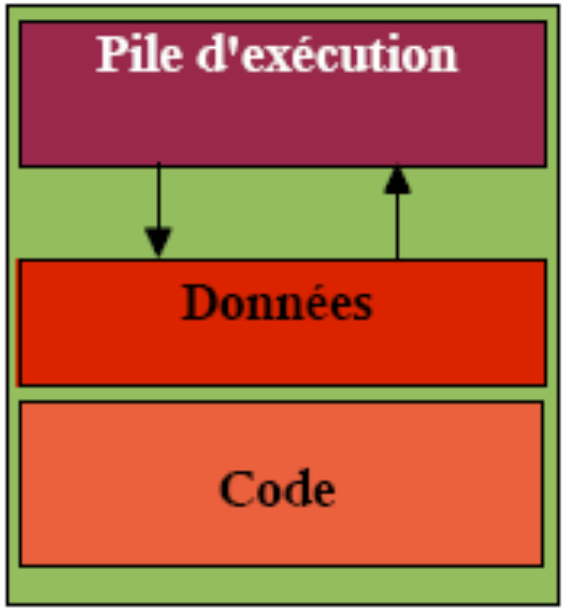
⇒ Contexte d'exécution

- Suspendre/Continuer un processus :
  - Sauvegarder/Restaurer son contexte d'exécution.

2. Processus : Concepts de Base

2.3 Contexte d'un processus

- Un processus en mémoire centrale :
  - L'état de la mémoire centrale associé à un processus est défini par le contenu de 3 **segments** (le code, la pile et les données) ***et* d'un contexte.**



- **Les segments d'un processus**
  - ***Segment code -- Lecture seulement***
    - Contient les instructions : en langage d'assemblage
    - Invariant (toute la durée d'exécution du processus).
  - ***Segment données -- Lecture/Ecriture***
    - Contient les variables globales + données statiques (constantes), qui sont initialisées à la compilation ainsi que les allocations dynamiques de mémoire.
  - ***Pile d'exécution -- Lecture/Ecriture***
    - Un programme est constitué d'un ensemble de fonctions (procédures) qui s'échangent d 'informations
    - *Contient les variables échangées + variables locales*
    - *File d'attente gérée selon LIFO*

2. Processus : Concepts de Base

- Un processus est donc caractérisé par:
  - Une image binaire
    - Un fichier exécutable pouvant être chargé en mémoire.
  - Un contexte d'exécution
    - Code, valeurs de registres, caches processeur
  - Un espace mémoire d'adressage
    - Données (variables globales, locales, constantes)
    - Espace mémoire virtuel
  - Un ensemble de ressources
    - Fichiers ouverts
    - Ressources de communications interprocessus
    - Ressources matériels
    - Alertes/signaux en attente

2. Processus : Concepts de Base

2.3 Contexte d'un processus

- Un processus possède **une fiche signalétique** permettant à l'OS de disposer des informations liées à ce processus.
  - Cette fiche signalétique est le ***Process Control Block (PCB)***
- L'OS détient une table, contenant la liste de tous les processus:
  - chaque entrée conserve une fiche **PCB**.
  - Le nombre des emplacements dans cette table des processus est limité pour chaque système et pour chaque utilisateur.
- Le PCB permet de gérer les ressources système de manière *virtuelle*.
  - La **virtualisation** **des ressources** permet aux processus de faire "comme s'il disposait chacun d'une machine complète".
    - Mémoire
    - Registres
    - Matériel
    - Fichiers

2. Processus : Concepts de Base

- Afin que la table des processus ne soit énorme:
  - La plupart des OS disposent de **2 zones** par processus
    - La **première** :
      - n'est jamais swappée
      - contient les informations critiques dont le système a tjs besoin
      - **zone proc** (process)
    - La deuxième :
      - Swappable, contenue dans l'espace d'adressage du processus
      - contient les infos dont le processus n'aura besoin que lorsque il sera en possession du processeur
      - **zone u** (user)

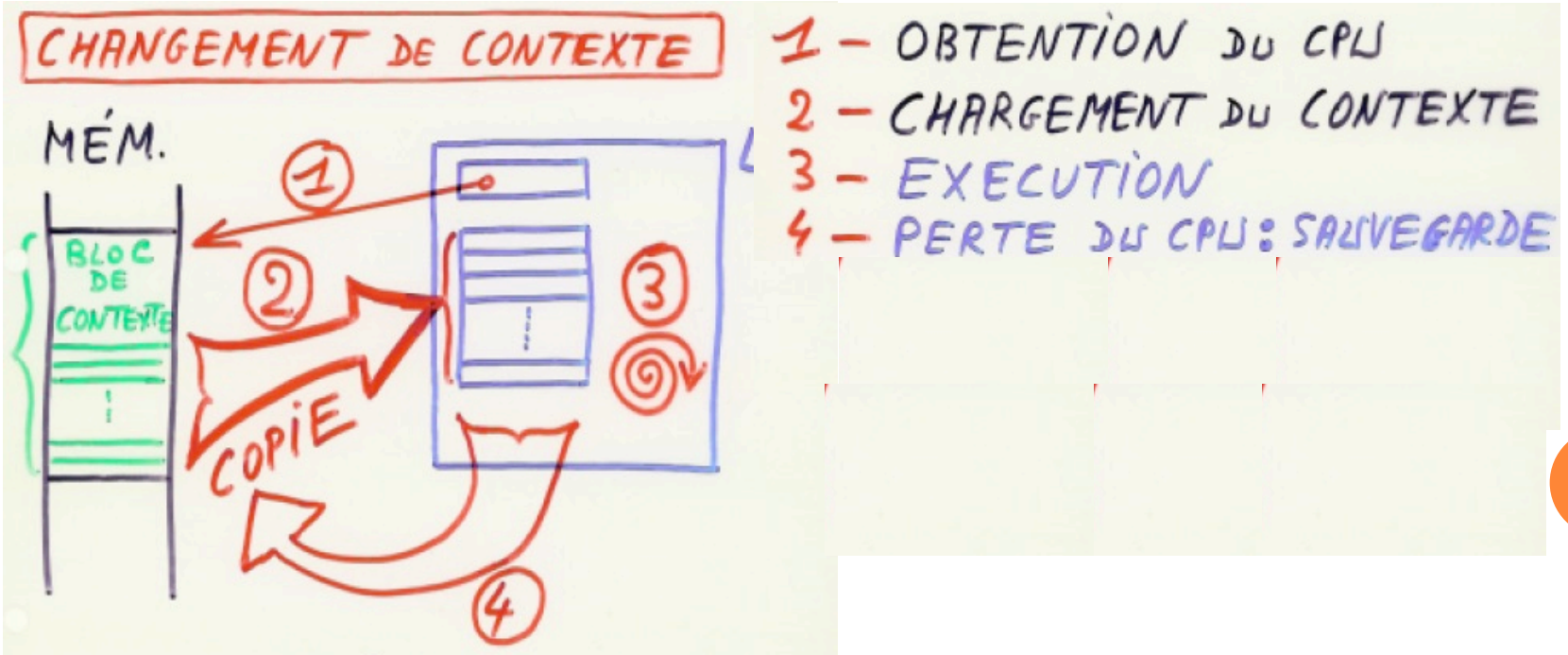
- Informations du PCB (1)
  - **Identité**
    - Identificateur de processus (PID)
      - numéro unique (à un instant donné, mais réutilisable)
    - Informations de généalogie
      - processus parent (PPID), éventuellement processus enfants
    - Information de droits
      - utilisateur propriétaire du processus
      - utilisateur *effectif (augmentation ou diminution de droits), exemples :*
        - droits d'administration « locale »
        - serveurs lancés par l'administrateur
        - serveur d'application

- Informations du PCB (2)
  - **Exécution**
    - État du processus
      - cf. transparents suivants
    - Contexte processeur
      - état des registres du processeur pour ce processus
      - Mot d'état (PSW - Program Status Word) : Etat d'exécution (Actif/Attente, Mode de fonctionnement : Superviseur/utilisateur, Masque d'interruption
      - Contexte accessible en mémoire
    - Autres informations d'ordonnancement: priorité
  - **Ressources**
    - Informations sur la mémoire utilisée / utilisable
      - tables de pages (cf. la gestion de la mémoire)
    - Informations sur le temps passé : temps réel, temps utilisateur
    - Liste des fichiers ouverts
      - fichiers classiques, mais aussi périphériques, moyens de communication avec d'autres processus (*pipe...)*
    - Autres ressources utilisées...

2. Processus : Concepts de Base

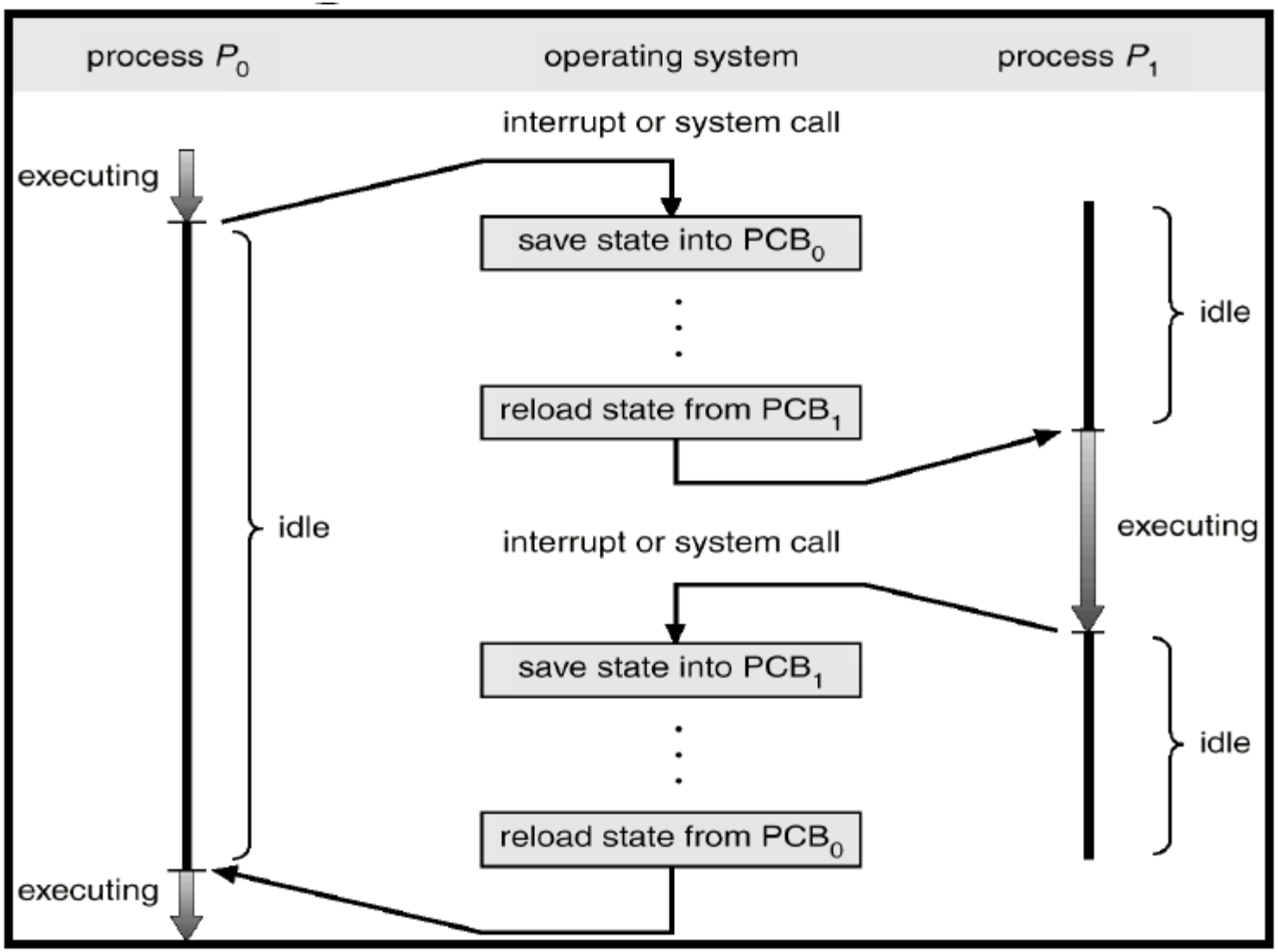
2.4 Commutation de contexte

- **Context Switching**
- Mécanisme matériel **indivisible :**
  - **Sauvegarde du contexte matériel (PSW)** Process Status Word) dans le contexte logiciel (**PCB)**
  - **Chargement d'un nouveau PSW à partir du PCB** d'un autre processus



2. Processus : Concepts de Base

2.4 Commutation de contexte



2. Processus : Concepts de Base

- Une activité (sur votre PC) est réalisée lorsqu'il y a
  - Initialisation du système
  - Exécution d'un appel système provoquant la création d'un processus
  - Activité de l'utilisateur qui provoque cette création
- **Techniquement**: il y a un appel système par le processus en cours d'exécution qui provoque la création d'un autre processus
- Comment les voir ?
  - Ctrl+alt+del sous Windows/ps sous UNIX

2. Processus : Concepts de Base

2.5 Création d'un processus

- Un processus parent crée des processus fils, qui, à leur tour, peuvent créer d'autres processus, formant ainsi **un arbre de processus**
- **Partage de Ressources**
  - Les parents et les fils partagent toutes les ressources
  - Les fils partagent un sous-ensemble des ressources du parent
  - Le parent et les fils ne partagent aucune ressource
- **Exécution**
  - Le parent et les fils s'exécutent simultanément
  - Le parent attend la terminaison des fils
- **Espace d'Adressage**
  - Le fils duplique le parent
  - Le fils a un programme différent du parent
- **Exemples UNIX**
  - Appel système **fork** crée de nouveaux processus
  - Appel système **exec** utilisé après un **fork** pour remplacer la mémoire du processus parent par un nouveau programme

2. Processus : Concepts de Base

2.6 Terminaison d'un processus

- Un **arrêt** est
  - **Volontaire** (encodé dans le programme)
  - **Involontaire** (dû à une erreur fatale, dû à une demande) : dans ce cas, le code de l'OS provoque la fin du processus.
- Le processus exécute la dernière expression et demande à l'OS de décider (**exit**)
  - Données de terminaison du fils renvoyées au parent intéréssé (via **wait**)
  - Ressources systèmes libérées par l'OS
- Le parent peut terminer l'exécution des processus fils (signal **abort)**
  - Le fils a dépassé les ressources allouées
  - La tâche du fils n'est plus utile
  - Si le parent se termine
    - Certains OSs ne permettent pas aux fils de continuer
    - Tous les fils terminés – terminaison en cascade
- Certains processus ne se terminent pas :
  - **démons**: réalisant des fonctions du système

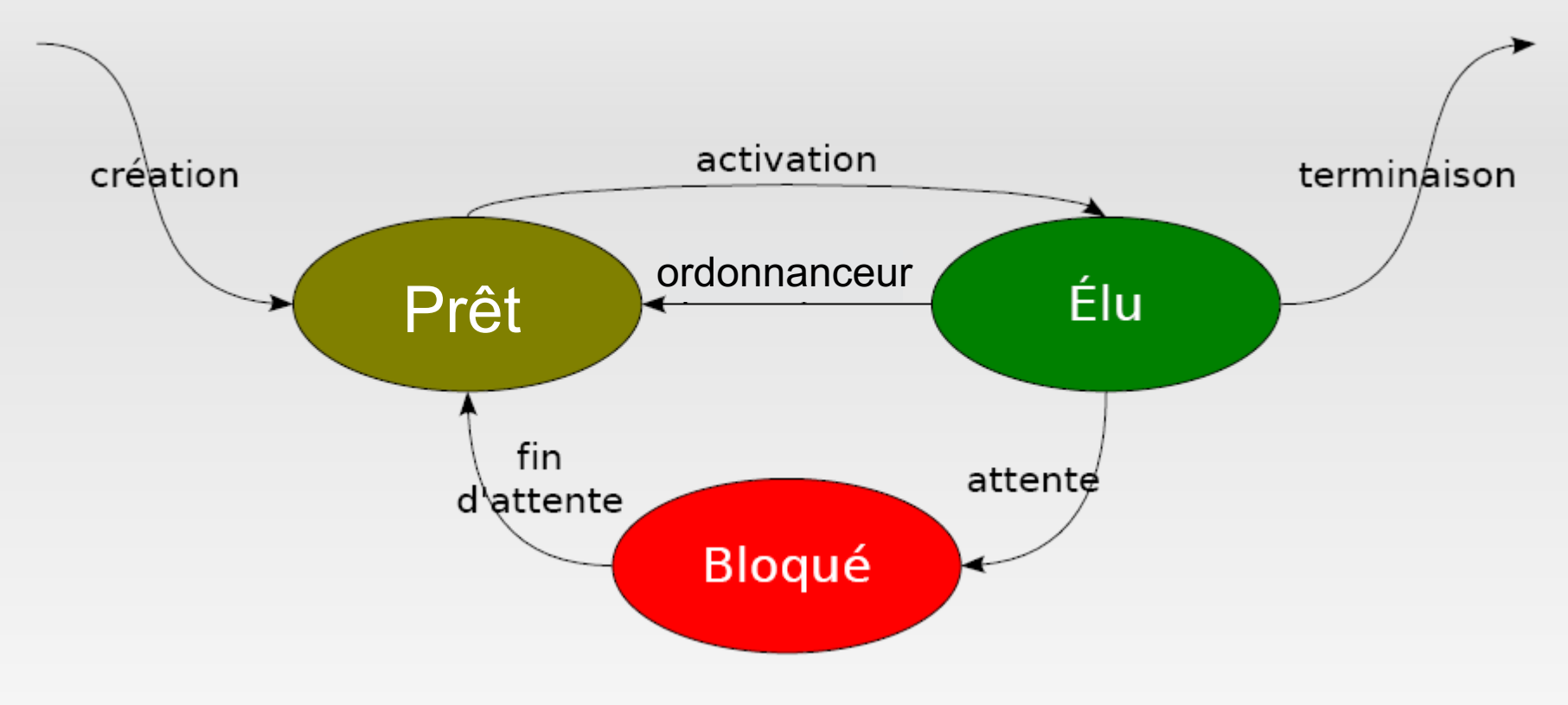
2. Processus : Concepts de Base

- Selon les systèmes, le nombre et le nom des états peut varier.
- Tous les systèmes comportent au minimum les trois états suivants :
  - **Élu ou Actif** : en train de s'exécuter sur le/un processeur
  - **Prêt ou Éligible** :
    - en attente de pouvoir s'exécuter sur le/un processeur
    - le processus dispose de toutes les ressources nécessaires à son exécution à l'exception du processeur.
  - **Bloqué** :
    - en attente d'un événement (ex: interruption)
      - Entrée /sortie
      - Attente d'un signal
      - Ressources non disponibles
    - ne peut donc pas s'exécuter pour l'instant

2. Processus : Concepts de Base

2.7 Etats d'un processus

- Diagramme des transitions



2. Processus : Concepts de Base

2.7 Etats d'un processus

- Transitions
  - **actif → bloqué :**
    - action volontaire (lecture sur disque)
    - Mise en attente
    - Résultat de la synchronisation de processus
  - **bloque → prêt**
    - action extérieure au processus (ressource disponible)
    - Réveil du processus bloqué après disponibilité de l'événement bloquant
    - Résultat de la synchronisation des processus
  - **prêt ←→ actif**
    - Décision de l'allocateur du processeur
    - Ordonnanceur, dispatcher, scheduler
- La mémoire ne peut contenir tous les processus prêts (id. pour bloqués):
  - un certain nombre d'entre eux sont déplacés sur le disque.
  - un processus peut alors être **prêt (bloqué) en mémoire** ou **prêt (bloqué) sur le disque**

3. Scénario exemple

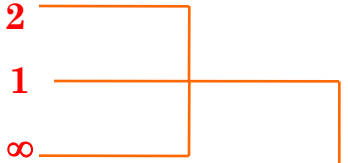
- 3 processus (**P1 actif, P2 et P3 prêts)**
  - P1 fait un read() avant l'épuisement de son quantum
  - Passage en mode noyau et exécution de l'appel système read
  - P1 passe à **bloqué**
  - Sauvegarde du contexte de P1
  - Appel de l'ordonnanceur
  - **P2 est élu**
    - Le contexte de P2 est restauré
    - P2 passe à **actif**
    - L'horloge est programmé pour un nouveau quantum

3. Scénario exemple

- P2 consomme entièrement son quantum de temps et ne fait pas d'E-S
- Interruption de l'horloge
  - Passage en mode noyau, exécution du gérant d'interruption
  - P2 passe à **prêt**, **sauvegarde du contexte**
  - Appel de l'ordonnanceur
- P3 passe à **actif**, **son contexte est restauré,**
- suite de son exécution (en mode utilisateur)
- La donnée demandée par P1 arrive:
  - interruption du contrôleur de disque
  - Passage en mode noyau et exécution du gestionnaire d'interruption.
  - La donnée est placée en mémoire, accessible à P1
- P1 passe à **prêt** **(on parle de réveil)**
- P3 passe aussi à **prêt**
- Appel de l'ordonnanceur

4. Relation entre processus

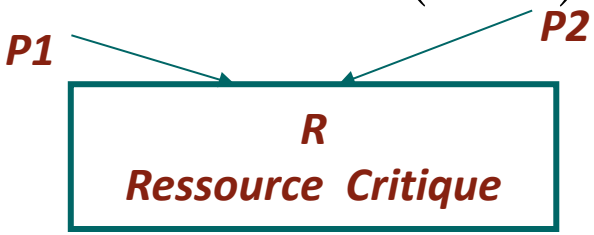
- Sur l'exemple
  - ressources du proc. de réalisation 1er gâteau: ustensiles, denrées
  - ressources du proc. de réalisation 2ème gâteau : ustensiles, denrées
  - manque de ressources ⇒conflit entre les processus
    - four ⇒ assez grand pour 2 gâteaux
    - batteur ⇒ un processus à la fois
    - horloge ⇒ partagée par tous



- **Ressource** : toute entité dont a besoin un processus pour s'exécuter
  - processeur physique, mémoire centrale, périphériques,
  - données momentanément indisponibles, événement, etc.
- Nombre de **points d'accès**
  - **nombre de processus** possédant la ressource à un instant donné
  - si un seul :
    - *ressource **critique**, processus en **exclusion mutuelle***

4. Relation entre processus

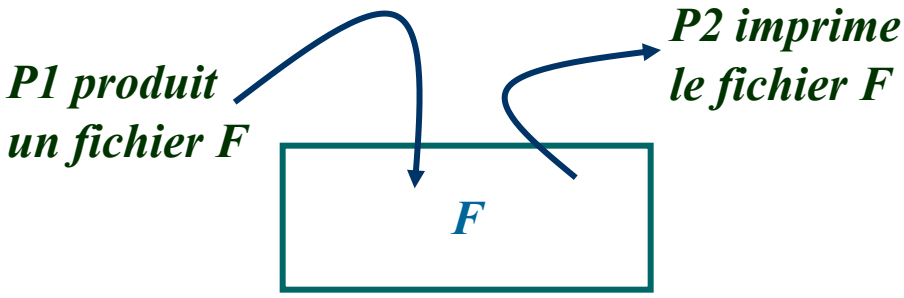
- Il existe entre les processus un certain nombre de relations:
  - **interactions**;
    - peuvent être de compétition ou de coopérations
- **Compétition**
  - plusieurs processus doivent utiliser simultanément une ressource à accès exclusif (1 seul processus à la fois),
  - encore appelée **ressource critique**.
  - **Ex.**
    - L 'usage du processeur (pseudo-parallélisme)
    - Accès à un périphérique (imprimante)
    - 2 processus en compétition sont dits en *exclusion mutuelle* pour cette ressource.
  - **Solution :** Faire attendre les processeurs demandeurs que l 'occupant actuel ait fini (FIFO)



Ordre d'utilisation : Indifférent
P1; P2 ou P2; P1

4. Relation entre processus

- **Coopération**
  - Situation dans laquelle plusieurs processus **collaborent à une tâche commune** et doivent se **synchroniser** pour réaliser cette tache.
  - Deux processus qui coopèrent peuvent également se trouver en exclusion mutuelle pour une ressource commune.



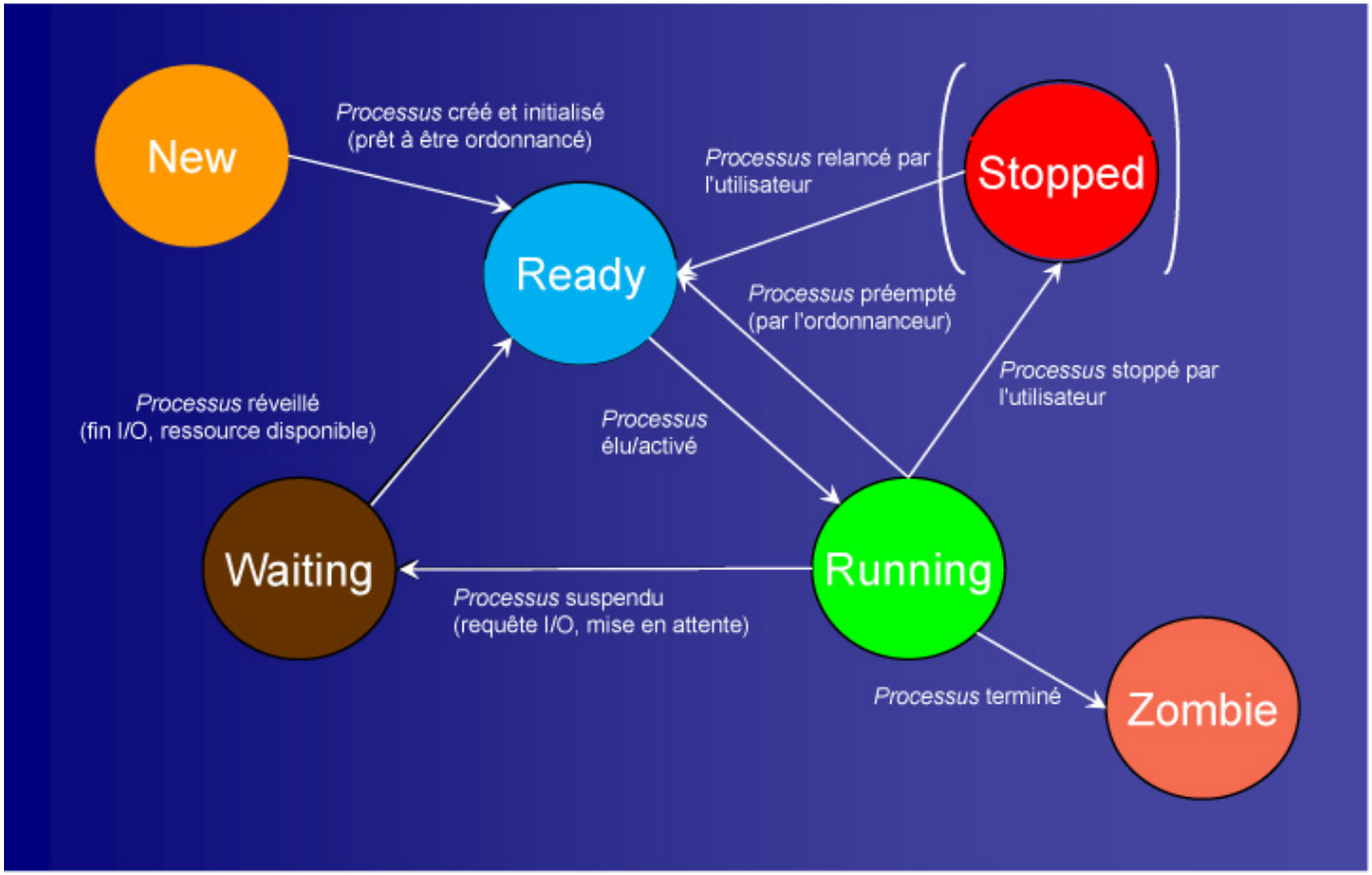
P2 ne peut s'exécuter que si:
P1 a terminé son exécution
P1 < P2

4. Relation entre processus

- **Synchronisation**
  - Un processus doit attendre qu'un autre processus ait franchi un certain point de son exécution :
    - point de synchronisation
  - Imposer des contraintes de synchronisation aux processus
    - Précédence des processus
    - Conditions de franchissement de certains points critiques

Ordonnancement des Processus

Précédemment



Ordonnancement

- Dans un système multitâches :
  - De nombreux processus attendent qu'un événement se produise
    - Ils n'ont pas besoin du processeur…
    - mais doivent pouvoir l'obtenir en priorité dès que l'événement attendu se produit
  - Certains processus souhaitent utiliser le CPU longtemps et de façon intensive
    - Ils souhaitent garder le processeur le plus longtemps possible
    - Conflit d'intérêt !
  - **Ordonnanceur** = arbitre + chef d'orchestre …

- À long terme
  - Quels processus vont être exécutés?
- A moyen terme
  - Quels processus sont résidents en mémoire ?
- A court terme
  - Quel processus est exécuté par le processus

- **Ordonnancement à long terme**
  - contrôler la **création/destruction** des processus.
  - la décision de créer un nouveau processus peut être soumise à condition :
    - Ex: le nombre de processus existants est inférieur à une limite fixée.
  - La création d'un processus peut avoir un impact sur ce qui se passe dans le système pendant une durée longue :
    - si le processus doit exécuter un calcul de plusieurs heures …

Ordonnancement à moyen terme

- contrôle le blocage/déblocage des processus
  - allocation/libération des autres ressources que le processeur nécessaires à l'exécution du processus.
  - Évincer ou rappeler en mémoire des processus
- A chaque demande de ressource physique ou logique peut être associée une file d'attente de processus bloqués
- **Stratégie d'ordonnancement** :
  - choisir quel(s) processus débloquer lorsque la ressource attendue sera disponible.
  - Ce choix a donc un impact sur ce qui s'exécute dans le système dans les quelques secondes qui suivent.

Ordonnancement à court terme

- contrôler **l'allocation** de la ressource **processeur**.
- répartir selon une **stratégie d'ordonnancement** adéquate le temps processeur disponible entre les différents processus exécutables
  - Gérer la transition : prêt ⇔ actif.
- Le choix d'un processus dans la file détermine ce qui se passe dans le système pour quelques centaines de millisecondes.

⇒ **Scheduler à court terme**

- A quels moments le scheduler à court-terme est-il exécuter ?
  1. Lorsque un nouveau processus est créé.
  2. Lorsque l'exécution du processus qui détenait le CPU est terminée.
  3. Lorsque le processus en cours d'exécution se bloque.
     - parce qu'il y a demande d'opération d'E/S,
     - demande d'une ressource non disponible
     - il se met en attente d'un résultat qui doit être fourni par un autre processus.
  4. Lorsqu'une interruption d'E/S intervient, et que sa gestion est terminée.

1. Introduction

1.2 Niveaux d'ordonnanceur

- Le SE permet 2 types de décisions sur le processeur :
  - Ordonnancement des processus :
    - Le **scheduler** choisit quel est le processus qui doit tourner
    - Problème: Dans quel ordre les processus sont servis
  - **Allocation** du processeur :
    - l'allocateur (**dispatcher**) lance l'exécution du processus choisi par le scheduler
    - La routine système, qui s'occupe de l'allocation du processeur à un processus sélectionné par le scheduleur du processeur.
- Ordonnanceur - Scheduler
  - Objectif :
    - Sur un intervalle de temps assez grand, faire progresser tous les processus, tout en ayant, à un instant donné, un seul processus actif
  - Multi-programmation :
    - réalisée par l'ordonnanceur au niveau le plus bas du système:
      - activée par des interruptions d'horloge, de disque et de terminaux.

1. Introduction

1.3 Objectifs à atteindre

- **Critères**
  - **Taux d'utilisation du CPU (efficacité):**
    - Rapport temps CPU / temps écoulé
      - Il faut tenir compte des entrées/sorties.
      - Lié au degré de multiprogrammation
    - En pratique, on obtient des valeurs comprises entre 40% et 90%
  - **Le débit (throughput):**
    - nombre de processus utilisateurs traités en moyenne par unité de temps
  - **Le temps de traitement moyen**
    - Moyenne des intervalles de temps séparant la soumission d'une tâche de sa fin d'exécution.
  - **Le temps de traitement total** (délai de rotation /turnaround)
    - temps de vie d'un processus dans le système entre la soumission du processus jusqu'à sa terminaison
    - T (FA_process_Prêts) + T (Occup_UC) + T (FA_E/S) + Temps(MS)
  - **Le temps d'attente :**
    - Temps passé dans l'état ready

1. Introduction

1.3 Objectifs à atteindre

- L' algorithme d'ordonnancement doit garantir :
  - Respect de la **priorité**
    - La plupart des systèmes permettent d'accorder des priorités différentes aux processus…
    - Priorité peut être statique ou dynamique (change au cours du temps) …
  - Respect de **l'équité**
    - Deux processus qui ont le même niveau de priorité doivent pouvoir utiliser le CPU aussi souvent l'un que l'autre
  - **Efficacité** :
    - le processeur doit être utilisé à 100%
  - Et
    - Temps de réponse :
      - l 'utilisateur devant sa machine ne doit pas trop attendre (mode interactif)
    - Temps d'exécution :
      - une séquence d'instructions ne doit pas trop durer (minimiser le temps d'attente en traitement par lots)
    - Rendement (débit):
      - il faut faire le plus de choses en une unité de temps

- Variables selon le système :
  - Pour les systèmes batch
    - Maximiser le nombre de jobs par heure
    - Minimiser le temps entre acceptation et terminaison
    - Maximiser le temps d'utilisation du CPU
  - Pour les systèmes interactifs
    - Minimiser le temps de réponse
    - Proportionnaliser le temps de réponse à la complexité perçue de la tâche
  - Pour les systèmes temps réels
    - Respecter les contraintes de temps
    - Prédiction de la qualité de service

- Dans un monde « idéal » (statistiquement) :
  - le hasard fait bien les choses : les processus endormis ne se réveillent pas tous en même temps
- Dans la réalité :
  - c'est souvent le contraire car les activités des processus sont « corrélées » :
    - les processus ne se réveillent pas au hasard
- 2 familles d'algorithmes **:**
  - **Sans réquisition :**
    - **c'est aux processus de relâcher** volontairement la ressource
  - **Avec réquisition :**
    - **l'algorithme est capable de récupérer la** ressource détenue par un processus au profit d'un autre

1. Introduction

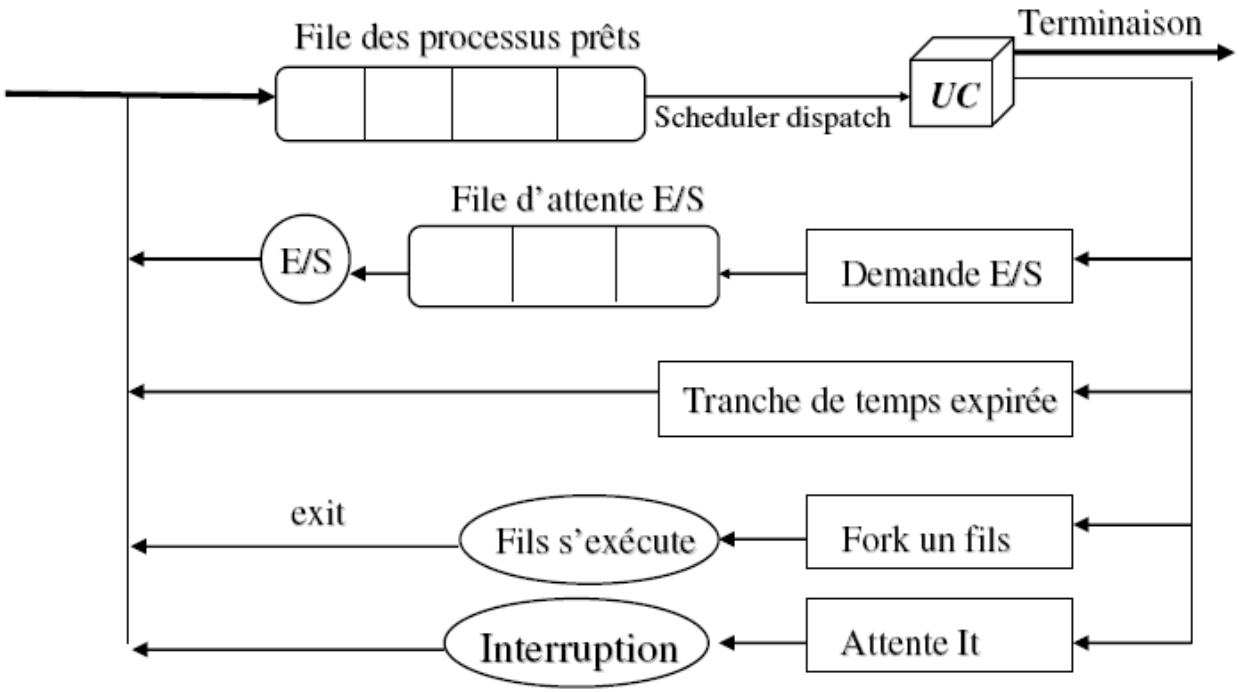
1.4 Types d'ordonnanceur

- Ordonnancement **non préemptif**
  - Ressource allouée à une entité jusqu'à ce qu'elle n'en ait plus besoin
    - **Inconvénients** :
      - Ne peut convenir aux activités temps-réel
      - Convient difficilement aux activités interactives :
        - Obligation de programmer des applications « sociables »
        - Tolérable dans un système faiblement multi-tâches (Win 3.x, 95, …)
    - **Avantages** :
      - Facile à mettre en oeuvre
      - Pas besoin de mécanismes matériels spécifiques
  - **Mise en œuvre** :
    - Au moment de la libération de la ressource :
      - Le (ex-)détenteur de la ressource invoque l'algorithme d'ordonnancement
      - Cette action peut être réalisée à l'insu du programmeur
    - L'algorithme **choisit l'utilisateur suivant**
    - L'algorithme déclenche la **commutation de contexte**

1. Introduction

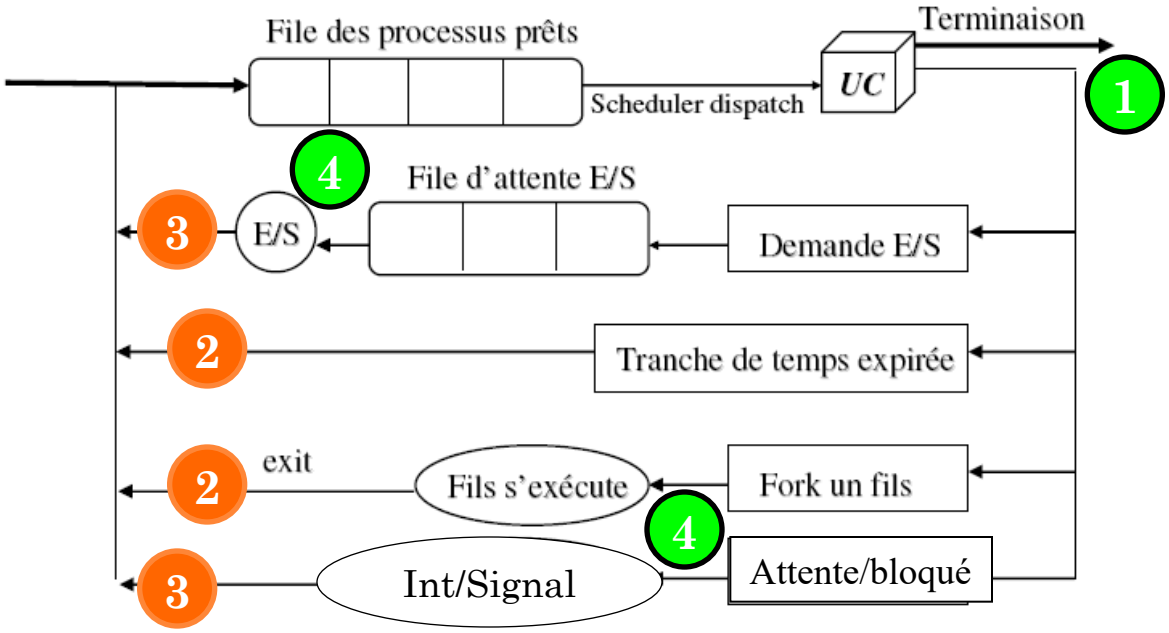
- **Ordonnancement préemptif**
  - **Motivations** : Politiques sans réquisition mal adaptées voire inadaptées à certaines activités (Temps réel, Interactivité)
  - Réquisition :
    - Forcer le partage du temps d'utilisation (modulo contraintes de priorités)
      - amélioration temps de traitement maximum
      - détérioration temps de traitement moyen
  - **Mise en œuvre**
    - Le détenteur de la ressource peut être interrompu avant d'avoir terminé
      - Lorsqu'un délai maximal expire
      - Lorsqu'une entité de priorité plus élevée demande la ressource
    - La politique d'ordonnancement choisit la nouvelle entité
    - L'entité interrompue est mise en sommeil
  - c'est la politique utilisée dans les systèmes « à temps partagé » (time-sharing) Unix, NT,

- Représentation d'un ordonnanceur



1. Introduction

1.4 Types d'ordonnanceur



- Cas 1 et 4 .
  - Il y a ordonnancement
  - Ordonnancement **non préemptif ou sans réquisition**
  - Ne pas retirer le processeur à un processus tant que celui-ci n'est pas bloqué par une E-S
- Cas 2 et 3:
  - Il peut y avoir ordonnancement
  - Ordonnancement **préemptif ou avec réquisition**
  - Suspension d'un processus en exécution, à n'importe quel moment

2. Stratégies d'ordonnancement

- Les demandes d'accès au processeurs sont gérées par des files d'attente.
- Différentes méthodes de gestion de files d'attente :
  - ⇒ stratégies d'ordonnancement différentes
  - ⇒ des comportements de 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**

2. Stratégies d'ordonnancement

Schéma d'algorithme d'ordonnancement

tant que pas de processus élu **faire**

consulter la table des processus

sélectionner celui qui en tête de file

si pas d'élu **alors**

attendre jusqu'à la prochaine interruption

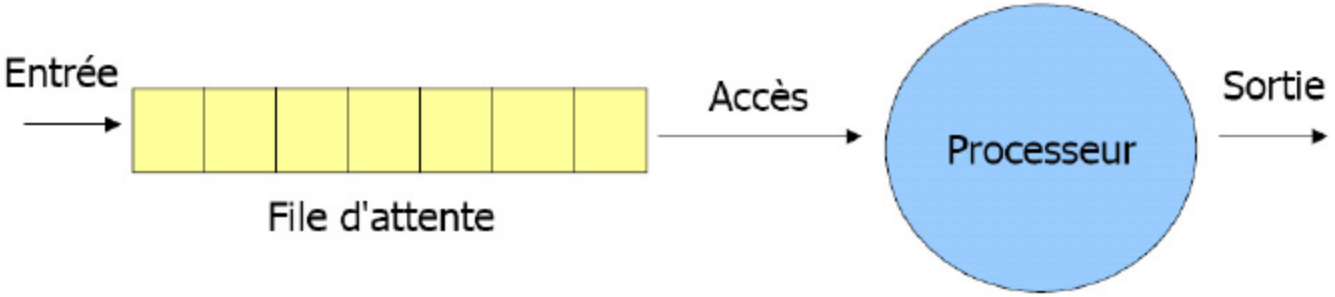
(processeur à l'état latent)

fin si

fin tant que

marquer le processus élu actif

basculer le contexte



2. Stratégies d'ordonnancement

2.1Principes 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 à tous les types d'utilisations
- Pour s'adapter à des cas particuliers, plusieurs principes sont souvent combinés

2. Stratégies d'ordonnancement

- **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**
  - Le temps d'attente est le nombre d'unités de temps durant lesquelles le processus est présent dans la file d'attente (sans être exécuté)
- 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 = d/(a + d)** où
      - 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

2. Stratégies d'ordonnancement

2.2 FCFS : First Come First Serve

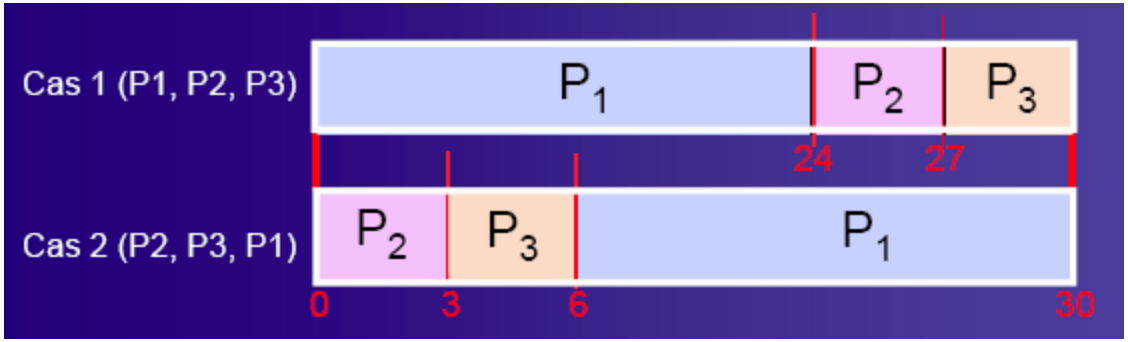
- Allocation selon l'ordre d'arrivée (premier arrivé = premier servi)
  - Non préemptive
  - File d'attente des prêts : FIFO
  - Modèle adapté au partage du processeur par des processus de même priorité (aucun privilège entre les processus)
- Implémentation
  - Un file de processus prêts, la tête de file est le prochain élu,
  - Les nouveaux processus (et processus interrompus) en fin de file
- Le traitement des processus est séquentiel
- Facile à comprendre, à implémenter

2. Stratégies d'ordonnancement

2.2 FCFS : First Come First Serve

- Intrinsèquement équitable pour des processus équivalents
- Diagramme de Gantt

Processus	Durée
P1	24
P2	3
P3	3



- Inconvénient :
  - défavorise les entités ayant besoin d'utiliser la ressource un court laps de temps
    - les processus courts sont pénalisés
      - Le temps d'attente n'est pas proportionnel au temps d'utilisation
      - ⇒ pas équitable,
      - ⇒ temps moyen de traitement élevé

2. Stratégies d'ordonnancement

2.2 FCFS : First Come First Serve

- Application :

Processus	Durée estimée	Date d'arrivée
P1	24	0
P2	8	1
P3	12	2
P4	3	3

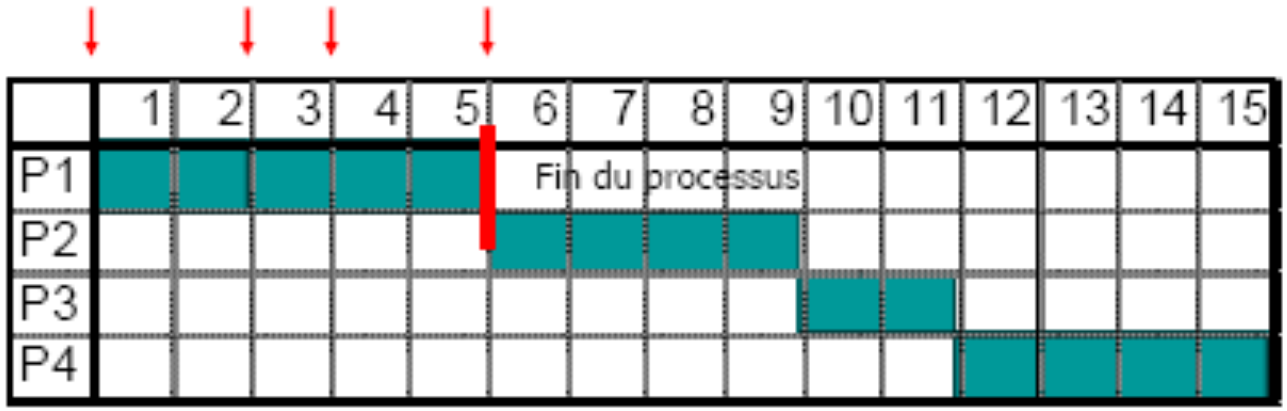
Schématiser l'exécution des processus selon leur ordre d'arrivée.

2. Stratégies d'ordonnancement

2.2 FCFS : First Come First Serve

- Une autre représentation :

		Arrivée	Durée
Processus 1	P1	0	5
Processus 2	P2	2	4
Processus 3	P3	3	2
Processus 4	P4	5	4



2. Stratégies d'ordonnancement

2.2 FCFS : First Come First Serve

- Avec des opération d'E/S

		Arrivée	Durée	E/S	Durée E/S
Processus 1	P1	0	5	1	3
Processus 2	P2	2	4		
Processus 3	P3	3	2		
Processus 4	P4	5	4		

- Exécution d'un processus : périodes CPU+ périodes I/O
- Il faut en tenir compte !

2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

- Également appelé : SJNext et PCTE
- Processus le plus court d'abord
  - la file d'attente est ordonnée non plus de façon chronologique mais en fonction du temps d'exécution nécessaire
    - les travaux courts en tête de file
  - Analogie avec la FA à une caisse de grande surface, où une règle de politesse est de laisser passer quelqu'un qui a peu de choses si l'on a plein dans le caddie
  - Sélection du processus nécessitant le moins de temps d'exécution
    - Connaissance a priori de la longueur du prochain cycle CPU
      - donne toujours la main à celui qui mettra le moins de temps avant de se bloquer / terminer
  - Difficile à implémenter
  - Optimal
- Algorithme sans réquisition
  - Le prochain cycle le plus court est sélectionné
  - En cas d'égalité, on revient au FCFS

2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

		Arrivée	Durée du prochain cycle CPU
Processus 1	P1	0	5
Processus 2	P2	0	4
Processus 3	P3	4	2
Processus 4	P4	3	4

2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

- SRTF : Shortest Remaining Time First
  - Version préemptive du SJF
    - temps restant le plus court
  - Choisir le processus dont le temps d'exécution restant est le plus court
  - Il y a réquisition selon le critère de temps d'exécution restant et l'arrivée d'un processus
  - Possibilité de morcellement d'un processus
    - Nécessité de sauvegarder le temps restant
- Favoriser les processus interactifs.

2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

- SRTF

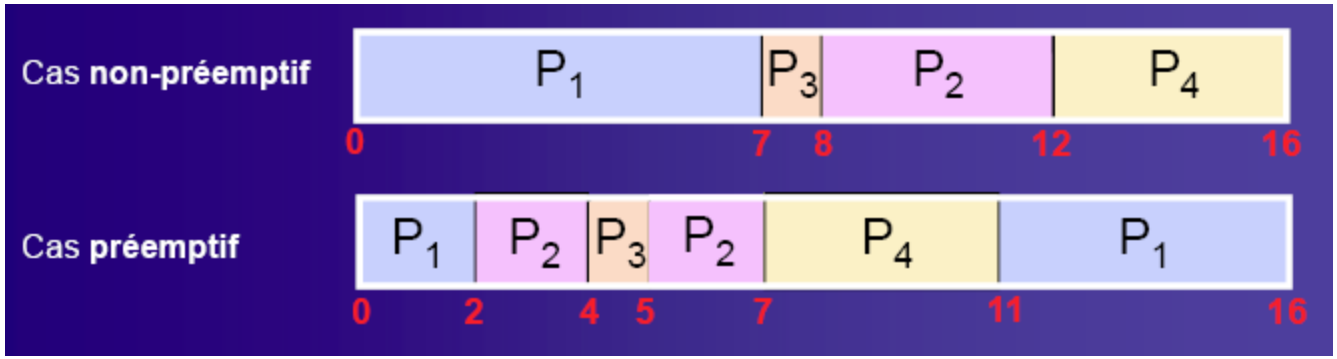
		Arrivée	Durée du prochain cycle CPU
Processus 1	P1	0	7
Processus 2	P2	2	4
Processus 3	P3	4	2
Processus 4	P4	3	5

2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

- SJF vs SRTF

Processus	Durée	Arrivée
P1	7	0.0
P2	4	2.0
P3	1	4.0
P4	4	5.0



2. Stratégies d'ordonnancement

2.3 SJF : Short Job First

- **Avantages**
  - maximise le temps de réponse, le débit
  - Temps d'attente faible pour entités à courte durée d'utilisation

  ⇒ **temps moyen d'attente minimal**
- **Inconvénients**
  - Surcoût
  - Pas réaliste : exige la connaissance / **estimation** à priori des durées d'utilisation : hypothèse forte
  - **Famine** (privation) : les tâches dont la durée d'exécution estimée est longue peuvent attendre leur tour indéfiniment …
- SJF/SRTF est impossible à implémenter en tant que tel.
  - Impossible de connaître les durées à priori
    - faire une approximation.
    - estimation basée sur les durées précédentes.
      - Nécessite l'observation des durées d'exécution.
      - Possibilité de pondérer la dernière observation avec la dernière estimation.

$$e_{n+1}=\alpha t_n+(1-\alpha)e_n$$

Où e_n est la longueur estimée du n^{ième} cycle

α est une constante de contrôle du poids relatif entre 0 et 1

t_n est la durée du cycle telle qu'observée à l'instant n.

2. Stratégies d'ordonnancement

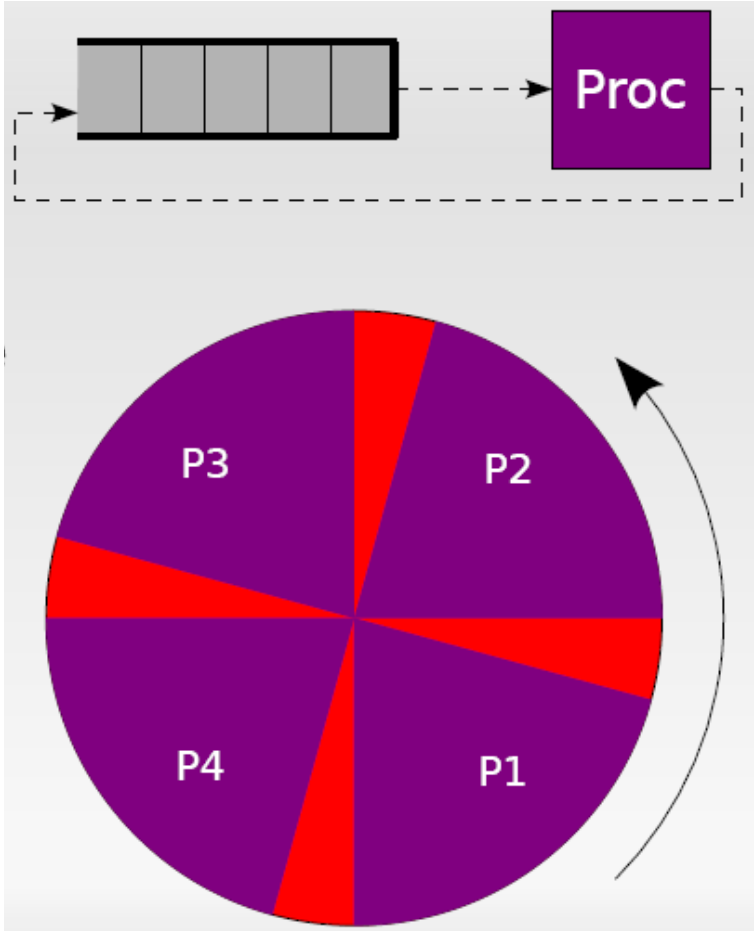
2.3 SJF : Short Job First

- Highest Response Ratio Next
  - Variante du SJF
  - on prend en compte le ratio entre le temps d'exécution et le temps que le processus a passé à attendre
    - Supprime le problème de famine
      - plus un processus attend, plus il augmente ses chances d'obtenir la main
  - Mais suppose toujours la connaissance du temps d'exécution

2. Stratégies d'ordonnancement

2.4 RR : Round Robin

- Politique du **Tourniquet**
  - l'un des algorithmes les plus utilises et des plus fiables
  - Ordonnancement **préemptif**
- **Principe**
  - Il s'agit d'un FCFS avec l'introduction d'une tranche ou quantum de temps (time slice) entre 20 et 50ms
    - Chaque processus possède un quantum de temps pendant lequel il s'exécute
    - Lorsqu'un processus épuise son quantum de temps : au suivant !
    - S'il n'a pas fini : le processus passe en queue du tourniquet et au suivant !
  - Nécessite une horloge

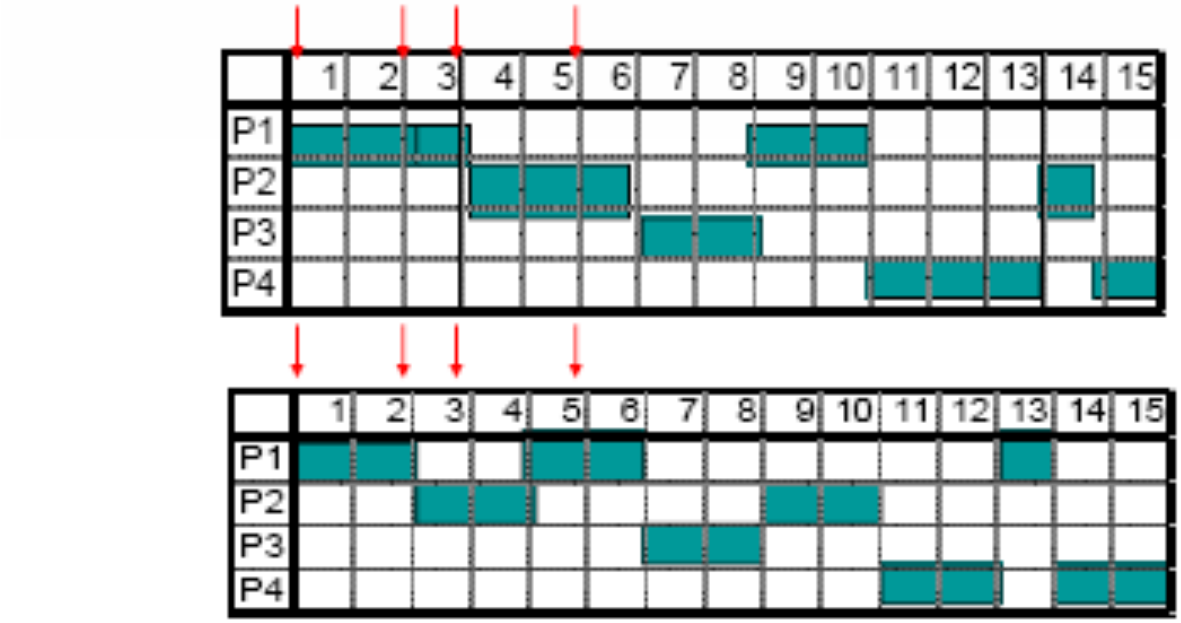


2. Stratégies d'ordonnancement

2.4 RR : Round Robin

- Exemple

		Arrivée	Durée
Processus 1	P1	0	5
Processus 2	P2	2	4
Processus 3	P3	3	2
Processus 4	P4	5	4



2. Stratégies d'ordonnancement

2.4 RR : Round Robin

- **Problème** : **réglage du quantum**
  - petit/grand; fixe/variable; est-il le même pour tous les processus ?
  - Les quanta égaux rendent les différents processus égaux
  - Quantum **trop petit** provoque trop de commutations de processus
    - Le changement de contexte devient coûteux (perte de temps CPU)
  - Quantum **trop grand** :
    - augmentation du temps de réponse d'une commande (même simple)
    - RR dégénère vers FCFS :
      - L'illusion d'exécution concurrente (parallèle) s'estompe
      - Lorsque la durée est trop longue, l'interactivité diminue
  - Le quantum idéal dépend de **la durée moyenne** d'une interaction et du nombre de processus...
- **Avantages**
  - temps de réponse borné, indépendamment de ce que font les processus (calculs ou E/S)
  - équitable
- **Inconvénients**
  - très sensible au choix du quantum
  - défavorise les processus orientés E/S (bloqués *avant la fin de leur quantum)*

2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

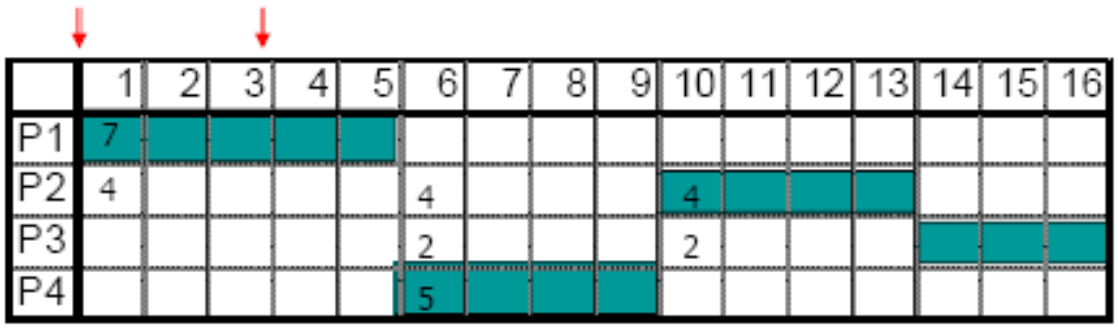
- **Principe** :
  - Associer au processus une **priorité** et choisir le processus dont la priorité est la plus élevée.
    - Un processus ne peut s'exécuter que si aucun processus de priorité supérieure n'est dans l'état *prêt*.
    - Si tous les processus ont la même priorité c'est la politique FIFO qui est appliquée.
- Une **Priorité** peut se définir en fonction de
  - L'espace mémoire utilisé, ou
  - L'importance relative du processus, ou
  - Du moment du dernier accès au CPU ou
  - …
- **Priorité dynamique**
  - **Augmentation** graduelle de la priorité des processus en *basse priorité*
  - évite la **famine** des processus.
- Ordonnancement **non-préemptif ou préemptif:**
  - **A tout instant la ressource est détenue** par l'entité **de plus haute priorité**
  - Il faut donc retirer la ressource à l'entité qui la possède lorsqu'elle n'est plus la plus prioritaire
    - En pratique : Il suffit que l'exécutif regarde la priorité d'une entité qui naît ou se réveille

2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

Exemple 1
version sans préemption

		Arrivée	Durée	priorité
Processus 1	P1	0	5	7
Processus 2	P2	0	4	4
Processus 3	P3	3	3	2
Processus 4	P4	3	4	5

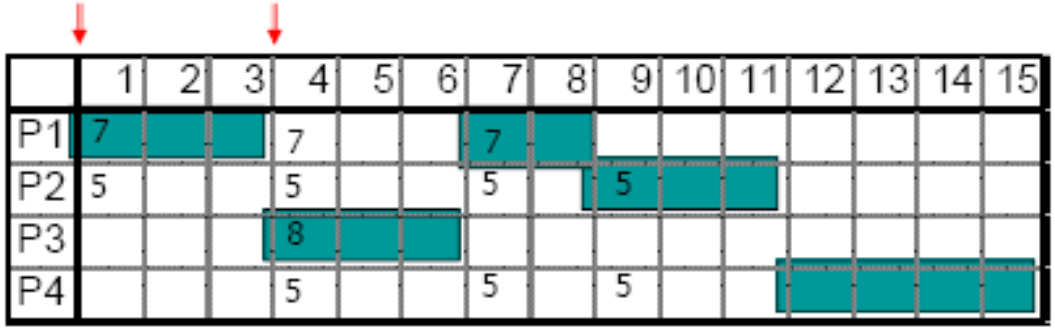


2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

Exemple 2
version avec préemption

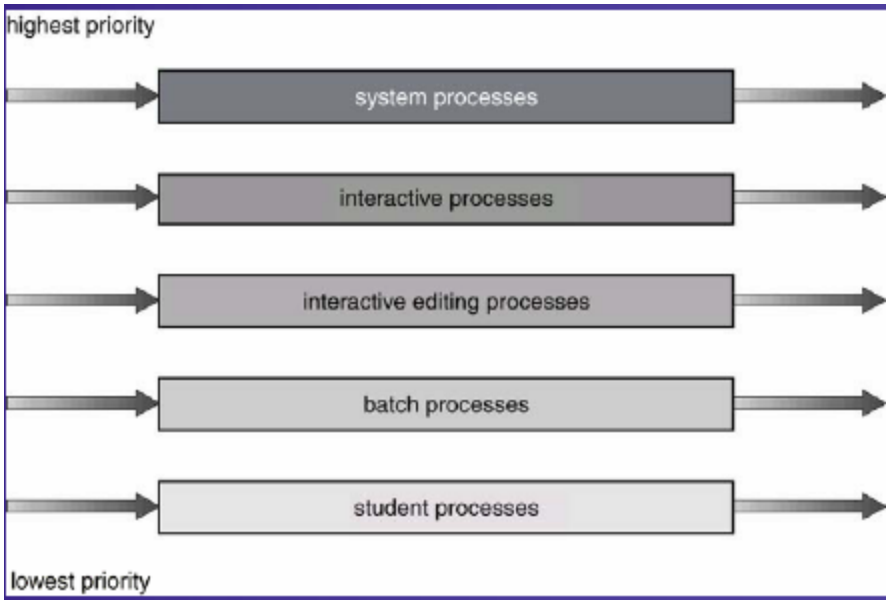
		Arrivée	Durée	priorité
Processus 1	P1	0	5	7
Processus 2	P2	0	3	5
Processus 3	P3	3	3	8
Processus 4	P4	3	4	5



2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

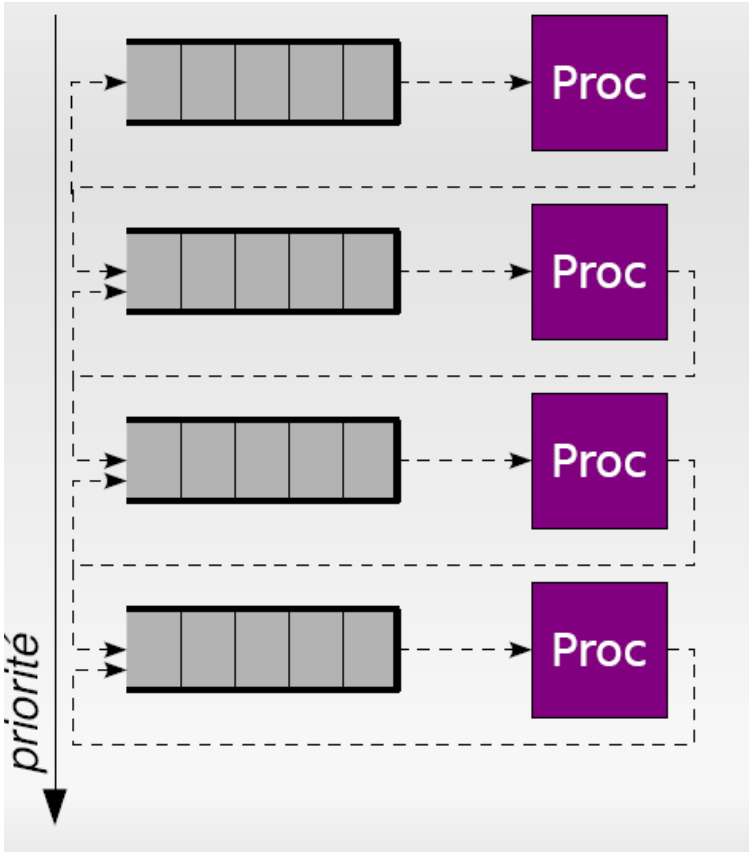
- **Files d'attente multiples**
  - Chaque file d'attente à sa propre politique d'ordonnancement
  - Découpage de la file d'attente des processus prêts en plusieurs files (processus système, interactifs, arrière-plan etc.)
    - Ordonnancement spécifique au sein de chaque file
      - RR, FCFS
    - Ordonnancement des files entre elles:
      - Priorités fixes
      - File *haute priorité: RR*
      - File *basse priorité: FCFS*



2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

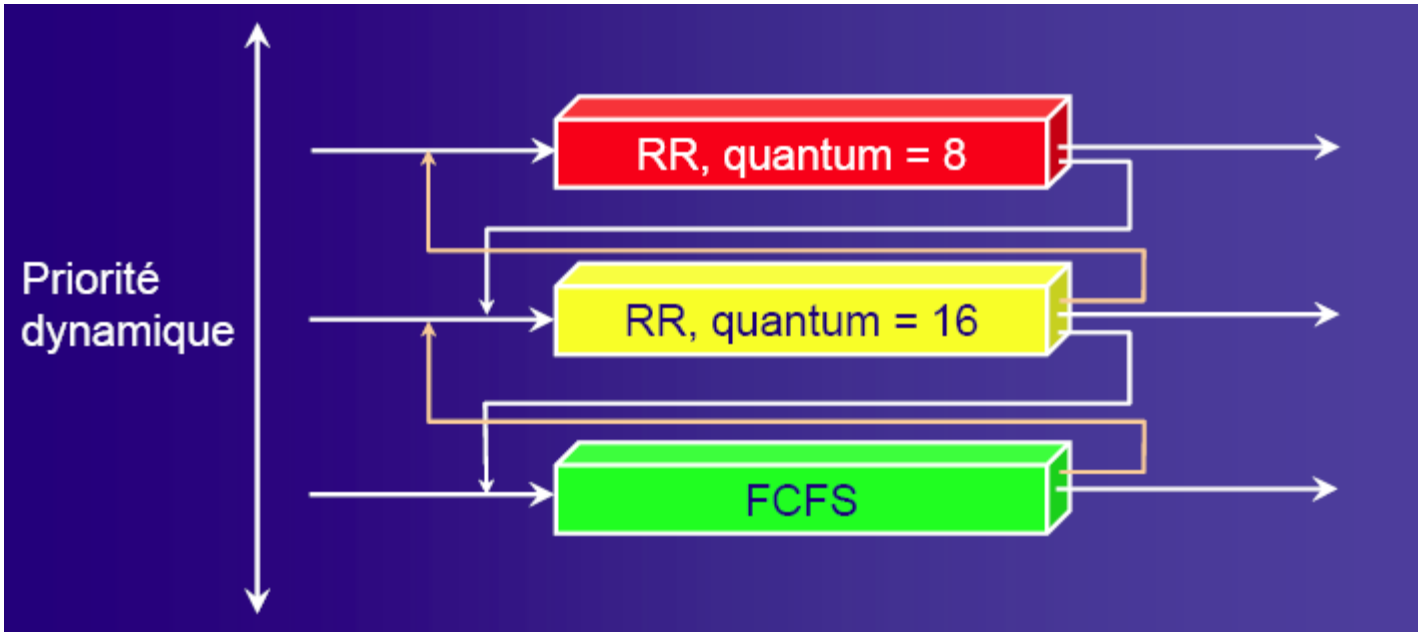
- **Files multiples avec rétroaction**
  - Possibilité de déplacer les processus d'une file d'attente à l'autre
  - Un processus peut être amené à changer de file au cours de son exécution.
  - reste à définir une bonne stratégie de changement.
  - **RR avec priorité :**
    - Lorsqu'un processus se bloque ou se termine, il retourne dans la même file
    - Lorsqu'il épuise son quantum, il passe dans la file suivante
    - Favorise le temps de réponse des processus orientés E/S



2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

- **Files multiples avec rétroaction (Multilevel Feedback)**
  - implémentation du vieillissement
  - dégradation des priorités (ex. cycles longs)



- **Files multiples avec rétroaction**
  - Définir les paramètres suivants :
    - Le **nombre** de niveaux
    - Les **algorithmes** de sélection
      - utilisés pour sélectionner un processus à l'intérieur de chaque file
    - La ou les conditions qui vont provoquer un passage d'un processus dans un niveau inférieur ou supérieur
    - Le fait que l'arrivée d'un processus dans un niveau supérieur préempte ou non un processus d'un niveau inférieur
    - Le mécanisme pour déterminer dans quel niveau se situe un nouveau processus

2. Stratégies d'ordonnancement

2.5 Ordonnancement à base de priorité

- **Exemple : Tourniquet avec priorité**
  - le niveau de priorité maximal (niveau 1)
    - file d'attente des processus arrivants
  - à chaque file d'attente ⇒ un quantum de temps spécifique
    - la file la moins prioritaire $Q_{n+1} \geq Q_n$
  - Fin du quantum :
    - processus non terminé 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
  - Si un nouveau processus dans une file de priorité supérieur à celui de la file d'origine d'un processus élu provoque le vidage de ce processus:
    - réquisition du CPU