Systèmes d'Exploitation & Programmation Concurrente

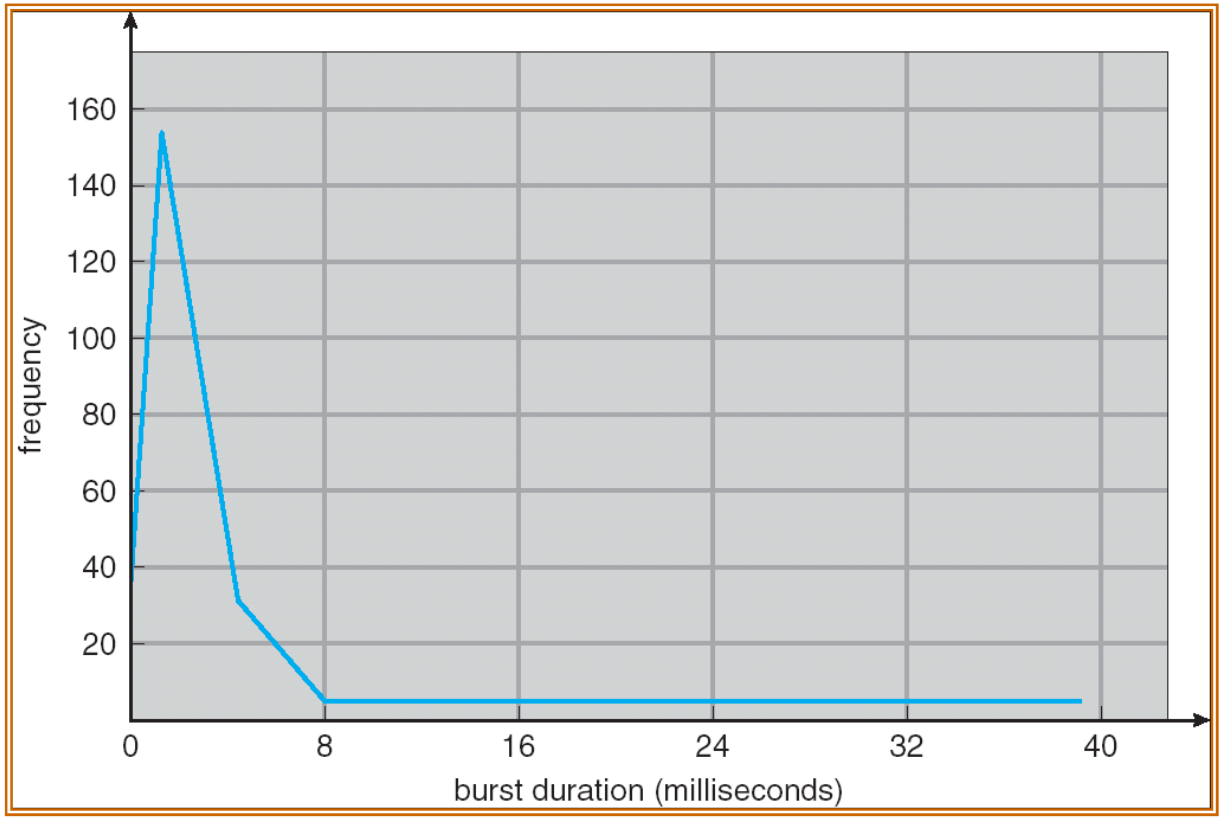
ORDONNANCEMENT DES ROCESSUS

1. Introduction
2. Types d'ordonnancement
3. Modèle simple d'ordonnancement
4. Politiques d'ordonnancement
   - ⇨ Organisations des files d'attente
   - ⇨ Ordonnancement FCFS / SJF/ priorité/RR/ SRTF/ Multi-niveaux
5. Hiérarchie d'ordonnancement
6. Ordonnancement des threads

Généralités

- ✓ Qu'est ce que l'ordonnancement?

  ⇨ Commuter dès qu'un processus se bloque ou se termine



Généralités (suite)

- ✓ Le SE permet 2 types de décisions sur le processeur :
  - ⇨ **Ordonnancement des processus**: L'ordonnanceur (**scheduler**) choisit quel est le processus qui doit tourner
    - ⇨ Problème: Dans quel ordre les processus sont servis?; par exemple, un seul processeur et plusieurs processus.
  - ⇨ **Allocation du processeur**: l'allocateur (**dispatcher**) lance l'exécution du processus choisi par le scheduler
- ✓ Scheduler
  - ⇨ Politique d'ordonnancement?
- ✓ Dispatcher
  - ⇨ Mécanisme d'allocation?

Introduction -- Généralités

- ✓ **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 (dans le processeur).
  - ⇨ **Rôle**: Prendre en charge la **commutation de processus**, qui règle les transitions d'un état à un autre des différents processus.
    - ⇘ **Multi-programmation**: réalisée par l'ordonnanceur au niveau le plus bas du système, elle est activée par des interruptions d'horloge, de disque et de terminaux.

Introduction -- Objectifs de la Multiprogrammation

- ✓ **Maximiser le taux d'utilisation de l'unité centrale (efficacité)** :
  - ⇨ Durée(UC_active)/Durée_totale.
  - ⇨ En pratique, on obtient des valeurs comprises entre 40% et 90% (très mauvais - très bon)
- ✓ **Maximiser le débit** (nombre de processus utilisateurs traités en moyenne par unité de temps).
- ✓ **Minimiser le temps de traitement moyen**
  - ⇨ Moyenne des intervalles de temps séparant la soumission d'une tâche de sa fin d'exécution.
- ✓ **Minimiser le temps de traitement total** (temps de vie d'un processus dans le système)
  - ⇨ Temps(Résidence_FA_process_Prêts) + Temps(Occup_UC) + Temps(At_FA_E/S) + Temps(Résidence_MS)
- ✓ **Minimiser le temps d'attente**
- ✓ *Ces objectifs sont contradictoires! Il faut des compromis -- optimiser pour des moyennes, valeurs min/max ou variance (prévisibilité)?*

Types d' Ordonnancement des Processus

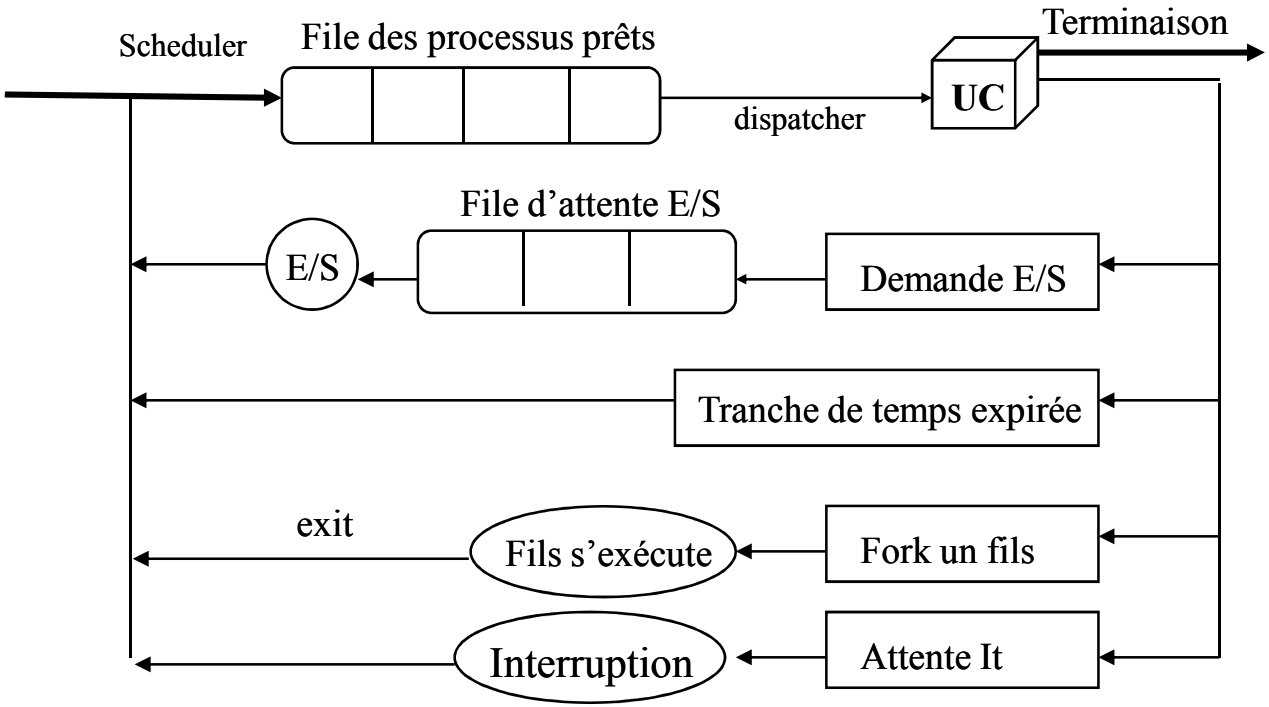
- ✓ Plusieurs processus sont prêts (en RAM) à être exécutés
- ✓ Le SE doit faire un choix -- **algorithme d'ordonnancement** :
  - ⇨ **Equité** : chaque processus doit recevoir sa part du temps processeur
  - ⇨ **Efficacité** : le processeur doit être utilisé à 100%
  - ⇨ **Temps de réponse**: l 'utilisateur 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** ("throughput "): il faut faire le plus de choses en une unité de temps

Types d' Ordonnancement des Processus

- ✓ **Les types d'ordonnancements:** ils sont distingués suivant les transitions permises dans le graphe d'état d'un processus
- ✓ Transition "interrompue" interdite --**Ordonnancement sans réquisition** (exp. Windows (< 95), Windows NT, Mac OS (< 10))
  - un processus est exécuté jusqu'à la fin : inefficace :-(!
  - ⇨ Transition "interrompue" autorisée --**Ordonnancement avec réquisition** (Exp. Windows 95, Mac OS X, UNIX)
    - (temps partagé)
    - à chaque signal d'horloge, le SE reprend la main, décide si le processus courant a consommé son quantum de temps et alloue éventuellement le processeur à un autre processus
  - ⇨ Il existe de nombreux algorithmes d'ordonnancement

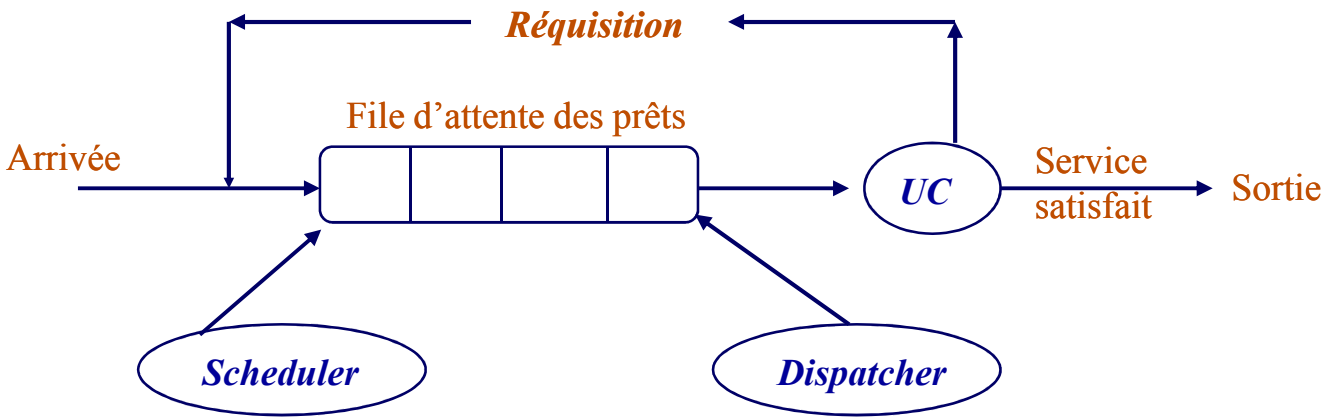
Modèle Simple d'Ordonnancement

✓ **Représentation de l'ordonnancement des processus**



4. Politique d'Ordonnancement – Organisations des FAs

- ✓ **Hypothèse : Un seul processeur + plusieurs processus**
- ✓ Une *file d'attente*, soit *FA*, des processus prêts: Espace réservé dans la mémoire centrale, contenant les informations relatives aux entités que gère la FA (pointer sur les BCP_i).
- ✓ Principe de chaînage d'une FA : avant/ arrière/ mixte
- ✓ Plusieurs processus sont mis dans une FA, et le service demandé leur est fourni tour à tour, en fonction de critères de gestion spécifiques à la FA.
- ✓ Plusieurs situations peuvent se produire



Gère la FA : Arrivée des processus et leur placement

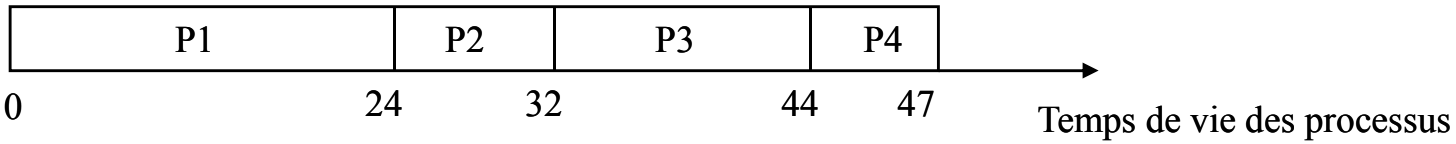
Gère l'allocation du processeur, il peut réquisitionner

Algorithmes sans préemption (FIFO, SJF, Priorité) (1)

- **Algorithme FIFO (First In First Out) -- Premier Arrivé Premier Servi**
  - *Un processus s'exécute jusqu'à sa terminaison, sans retrait forcé de la ressource --**SCHED_FIFO***
  - Même priorité pour les processus (aucun privilège entre les processus)
  - Facile à implanter, mais peu efficace (le choix n'est pas lié à l'utilisation de l'UC)
- **Exemple**

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

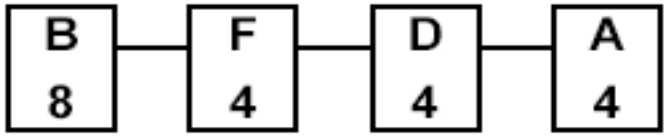
- Schématiser l'exécution des processus selon leur ordre d'arrivée. Pour cela, on utilise le ***DIAGRAMME DE GANTT***



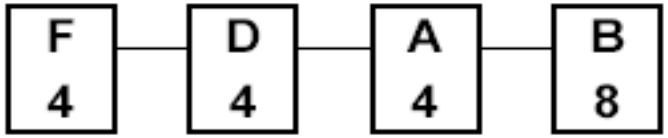
- ***Temps de traitement moyen = [(24 - 0) + (32 - 1) + (44 -2) + (47 -3)]/4 = 35,25***

Algorithmes sans préemption (FIFO, SJF, Priorité) (2)

- ✓ **Algorithme SJF -- Shortest Job First** (STCF -- Shortest Time to Completion First) **: Algorithme du "Plus Court d'Abord " :**
  - ⇨ Suppose la connaissance des temps d'exécution : estimation de la durée de chaque processus en attente
  - ⇨ Les processus sont disponibles simultanément → Algorithme optimal (sans préemption)
  - ⇨ Exécuter le processus le plus court → Minimise le temps moyen d'exécution
  - ⇨ 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



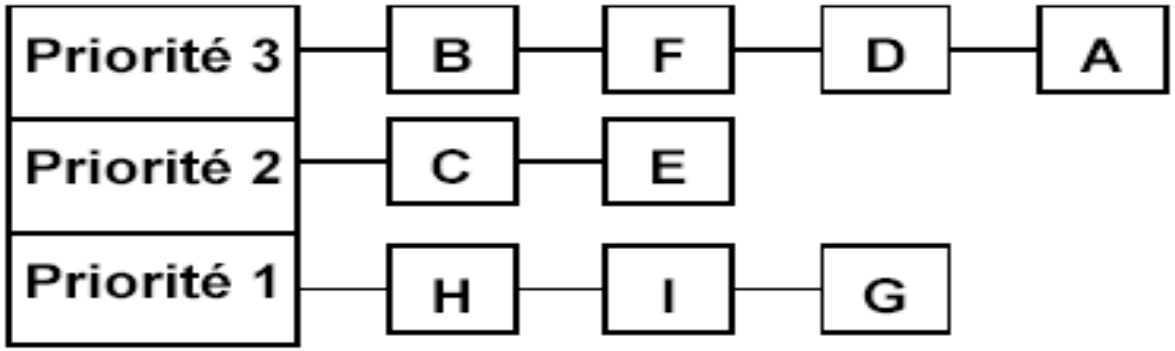
• sans ordonnancement :
Tmoyen = 1/4 (Tb+Tf+Td+Ta) = 14
avec Tb=8, Tf=8+4,
Td=8+4+4, Ta=8+4+4+4



• avec ordonnancement :
Tmoyen = 1/4 (Tf+Td+Ta+Tb) = 11
avec Tf=4, Td=4+4,
Ta=4+4+4, Ta=8+4+4+4

Algorithmes avec préemption (**Priorité**, RR, SRTF) (1)

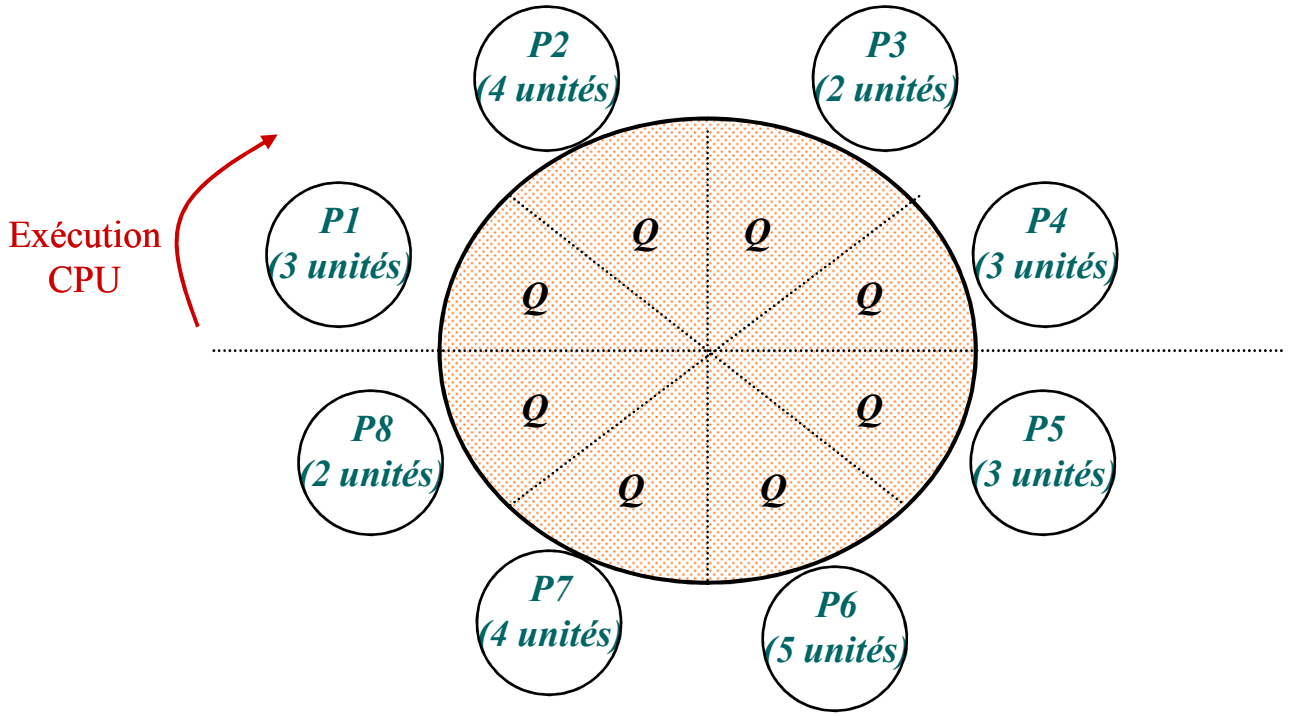
- ***A chaque processus est assignée (***automatiquement par le SE /externe) ***une priorité***
  - *Assignation statique* -- priorités fixes → facile à implanter
  - *Assignation dynamique* : la priorité initiale assignée à un processus peut être ajustée à d 'autres valeurs → difficile à implanter
    - ☹ **Pb. de famine :** un processus de faible priorité peut ne jamais s'exécuter si des processus plus prioritaires se présentent constamment
    - 😐 Recalculer périodiquement le numéro de priorité des processus (plusieurs FA) → la priorité d'un processus décroît (croit) au cours du temps pour ne pas bloquer les autres FA
- **Principe :** On lance le processus ayant la plus grande priorité



Algorithme d'ordonnancement à classes de priorité

Algorithmes avec préemption (Round Robin, Priorité, SRTF) (2)

- ***ROUND ROBIN* (SCHED_RR) :** algorithme tourniquet, l'un des plus utilisés et des plus fiables
  - ⇨ Ordonnancement selon l'ordre FIFO + préemption (Equitable)
  - ⇨ 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 !
- ***Exemple :*** Le quantum de temps, Q, est égale à 2 unités; quel est le temps de traitement moyen?



Algorithme Tourniquet -- Round Robin (suite)

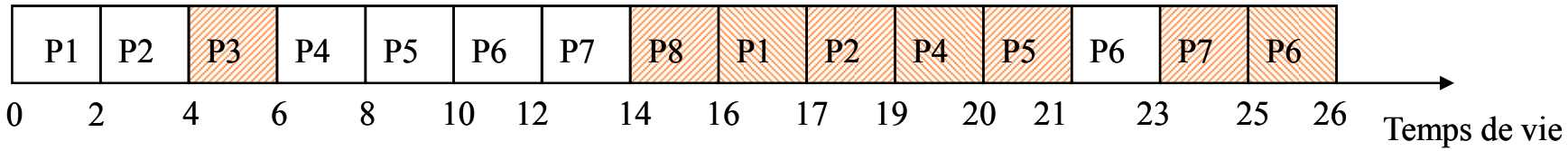


Diagramme de Gantt (Q=2 unités)

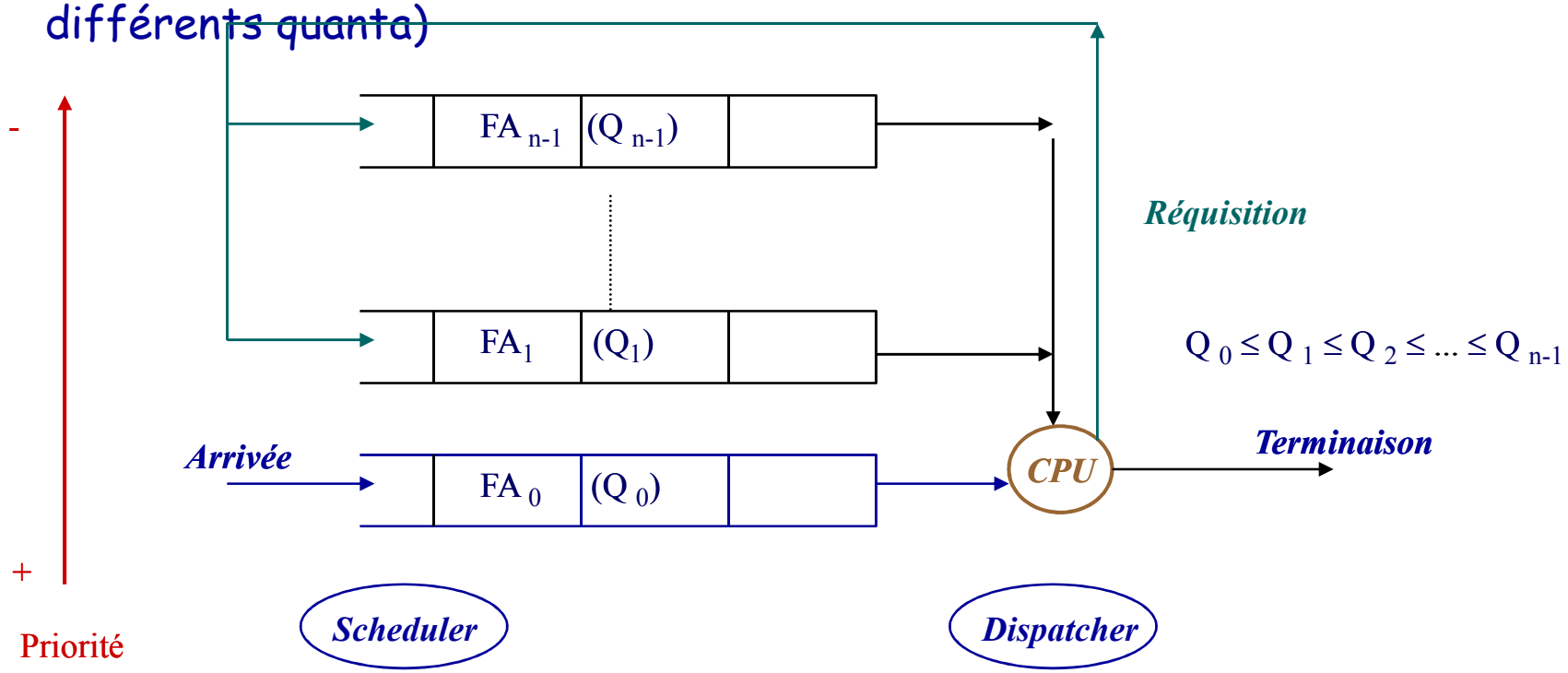
- ✓ ***Temps de traitement moyen =***

  [(17-0) + (19-1) + (6-2) + (20-3) + (21-4) + (26-5) + (25-6) + (16-7)] /8 = ***15,25***

- ✓ ***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 FIFO
  - ⇨ *Réglage correct* : varie d'un système (resp. d'une charge) à un autre et d'un processus à un autre
- ✓ Il existe d 'autres variantes de RR, telle que RR avec priorités

Algorithme Tourniquet avec priorités

- ✓ Le système de gestion possède n FA à différents niveaux de priorités (+ différents quanta)



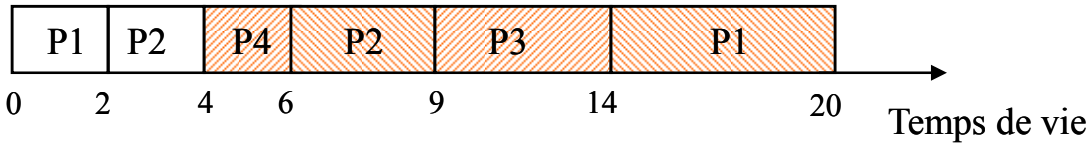
- ✓ A son arrivée, le processus est rangé dans la FA la plus prioritaire FA_0
- ✓ Si un processus dans FA_i épuise son quantum de temps Q_i ($0 \leq i \leq n-2$), il sera placé dans la FA_{i+1} (moins prioritaire)
  - ⇨ *Une FA_i ($0 \leq i \leq n-1$) ne peut être servie que si toutes les FA_j ($0 \leq j < i$) sont vides*
  - ⇨ *un processus qui a traversé toutes les FA sans épuiser son temps de traitement reste dans la FA la moins prioritaire.*

Algorithmes avec préemption (RR, Priorité, SRTF) (3)

- ***Algorithme SRTF (Shortest Remaining Time First) -- SJF avec réquisition***
  - *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*
- ***Exemple :***

Processus	Durée estimée	Date d'arrivée
P1	8	0
P2	5	2
P3	5	3
P4	2	4

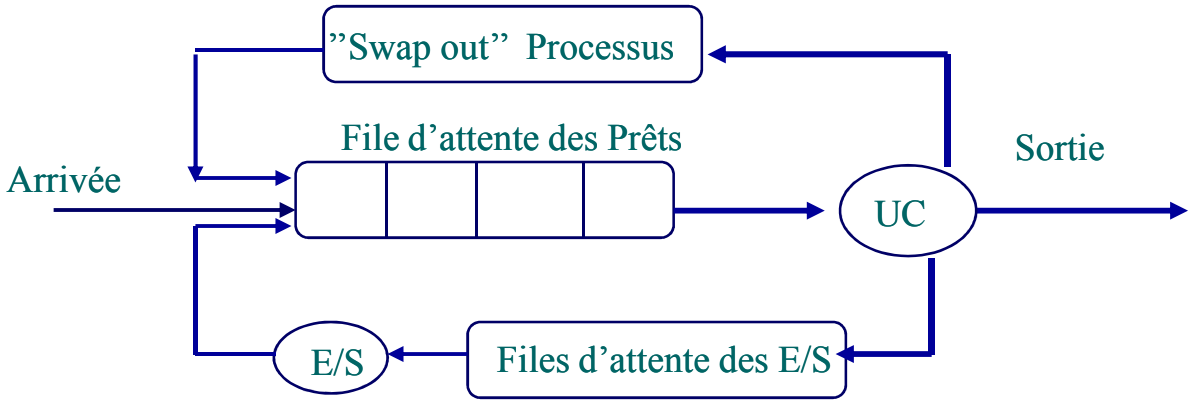
Diagramme de Gantt



- ***Temps de traitement moyen = [(20 - 0) + (9 -2) + (14 -3) + (6-4)]/4 = 9,5***
- Théoriquement, + SRTF offre un minimum de temps d'attente; - difficile de prédire le futur

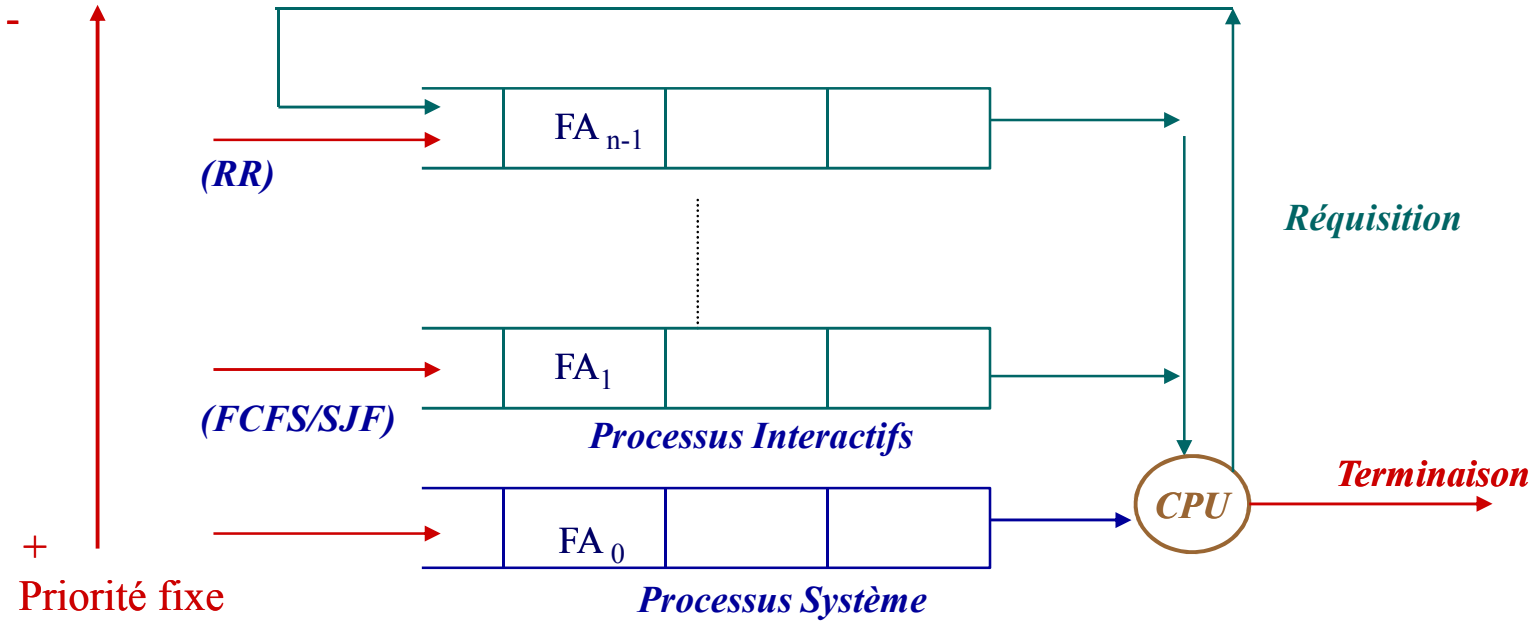
Hiérarchie d'Ordonnancement (1)

- ✓ **L'ensemble des processus prêts est-il souvent en mémoire centrale?**
- ✓ Un processus élu, qui est sur disque, prend beaucoup plus de temps qu'un processus en RAM pour être chargé.
- ✓ Les algorithmes d'ordonnancement complexes permettent de distinguer entre 2 types différents:
  - ⇨ **Ordonnancement à court terme (short term scheduling):** considère seulement les processus prêts en mémoire centrale.
  - ⇨ **Ordonnancement à long terme (long term scheduling):** consiste à utiliser un deuxième algorithme d'ordonnancement pour gérer les "swapping " des processus prêts entre le disque et la RAM



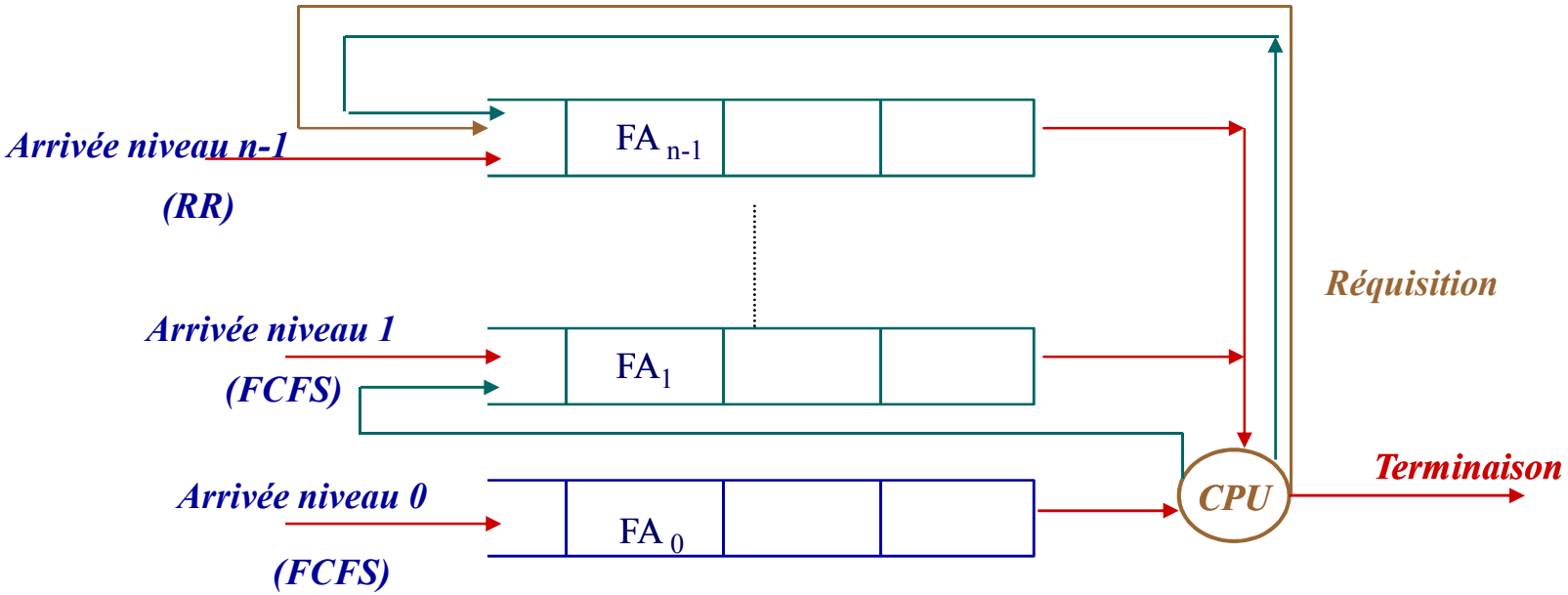
Hiérarchie d'Ordonnancement (2)

- ✓ **Ordonnancement multi-niveaux permet de satisfaire :**
  - ⇨ Favoriser les processus courts
  - ⇨ Favoriser les processus "I/O Bound", qui ne demandent pas trop l 'UC
  - ⇨ Déterminer la nature de chaque processus le plutôt possible et effectuer l'ordonnancement correspondant
- ✓ **Files d'attente sans liens** : un processus se trouvant dans dans FA_i ne peut se trouver dans FA_j ($j \neq i$); il reste dans FA_i jusqu'à ce qu'il se termine



Hiérarchie d'Ordonnancement (3)

- ***Files d'attente avec liens :*** hiérarchiser les FAs



- Un processus dans FA_i ne peut être sélectionné que si toutes les FA_j ($j<i$) sont toutes vides
- Permettre aux processus de se déplacer d'une FA à une autre
  - ⇨ Hiérarchie descendante/ascendante/bidirectionnelle
- Changement dynamique dans le comportement des processus
- Chaque FA a son propre algorithme d'ordonnancement
  - ⇨ *Descendante FA_0 est gérée avec FCFS*
  - ⇨ *Ascendante F_{n-1} est gérée avec FCFS*

Thread Scheduling

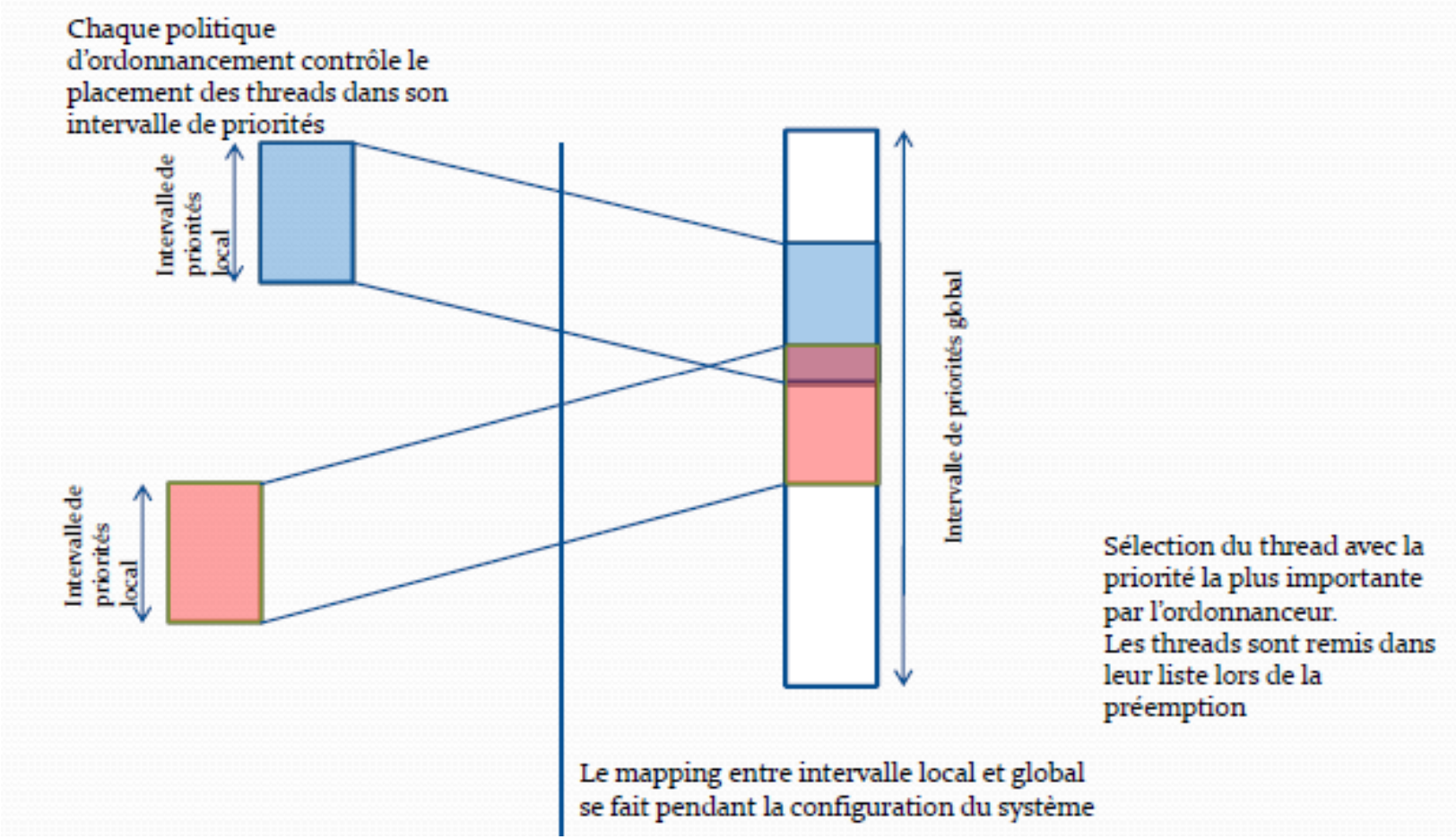
- ✓ **Local Scheduling** - How the threads library decides which user thread to run next within the process

- ✓ **Global Scheduling** - How the kernel decides which kernel thread to run next

Exemples d'ordonnancement

- ✓ Unix: multi-niveaux, plusieurs politiques, qui peuvent changer dans le temps
- ✓ Linux – multi-niveaux avec 3 niveaux les plus importants
  - ⇨ Realtime FIFO
  - ⇨ Realtime round robin
  - ⇨ Timesharing
- ✓ Windows Vista – politique avec priorité à deux dimensions:
  - ⇨ Classe des processus prioritaires
    - ⇘ Real-time, high, above normal, normal, below normal, idle
  - ⇨ Les priorités des threads sont relatives dela la classe des priorités.
    - ⇘ Time-critical, highest, …, idle

Exemples d'ordonnancement POSIX



Annexe

Threads scheduling

Threads scheduling (1/4)

- ✓ Mixed
  - ⇨ Mach:
    - ↳ user-level code to provide scheduling hints to the kernel
  - ⇨ Solaris:
    - ↳ assign each user-level thread to a kernel-level thread (multiple user threads can be in one kernel thread)
    - ↳ creation/switching at the user level
    - ↳ scheduling at the kernel level

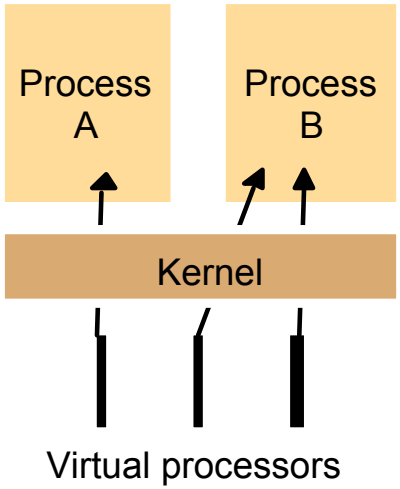
Threads Scheduling (2/4)

- FastThread package
  - hierarchical, event-based scheduling
  - each process has a user-level thread scheduler
  - virtual processors are allocated to processes
    - the # of virtual processors depends on a process's needs
    - physical processors are assigned to virtual processors
    - virtual processors can be dynamically allocated and deallocated to a process according to its needs.
  - Scheduler Activation (SA)
    - event/call from kernel to user-level scheduler
    - represents a **time slice** on a virtual processor (# of SA's < # of virtual processors)
    - user-level scheduler can assign threads to SA's (time slices).

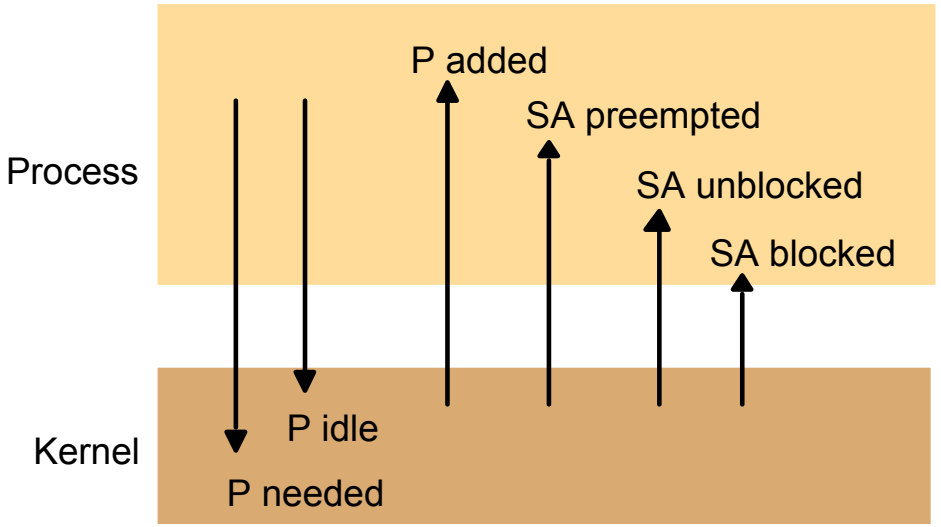
Threads Scheduling (3/4)

- ✓ Events from user-level scheduler to kernel
  - ⇨ **P idle:** virtual processor is idle
  - ⇨ **P needed**: new virtual processor needed
- ✓ Events from kernel to user-level scheduler
  - ⇨ **Virtual processor allocated (P added):**
    - User-level: scheduler can choose a ready thread to run
  - ⇨ **SA blocked (in kernel):**
    - Kernel: sends a new SA
    - User-level: a ready thread is assigned to the new SA to run
  - ⇨ **SA unblocked (in kernel):**
    - user-level: thread is back on the ready queue
    - kernel: allocate new virtual processor to process or preempt another SA
  - ⇨ **SA preempted (in kernel):**
    - user-level: puts the thread back to the ready queue

Threads Scheduling (4/4) --Scheduler activations



A. Assignment of virtual processors to processes



B. Events between user-level scheduler & kernel
Key: P = processor; SA = scheduler activation