

Chapitre 4

Systeme d'Exploitation

Gestion des Processus

Concepts importants

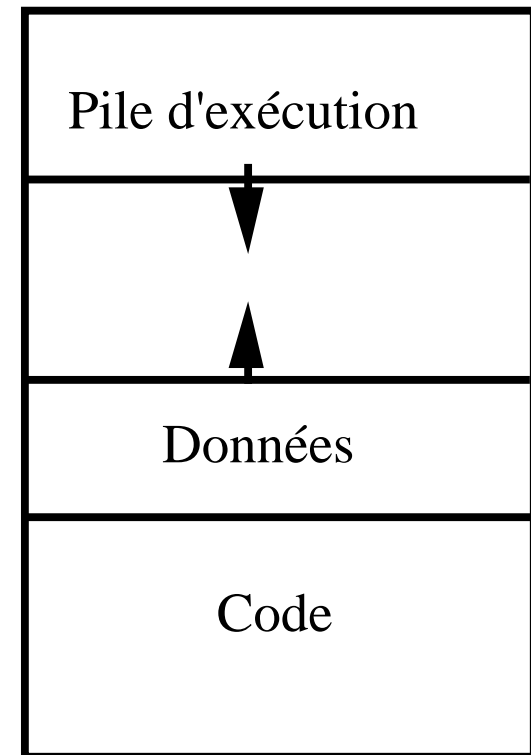
- **Processus**
 - ◆ Création, terminaison, hiérarchie
- **États et transitions d'état des processus**
- **Process Control Block**
- **Commutation de processus**
 - ◆ Sauvegarde, rechargement de PCB
- **Files d'attente de processus et PCB**
- **Ordonnanceurs à court, moyen, long terme**
- **Processus communicants**

Processus et terminologie (aussi appelé job, task, user program)

- Concept de processus: un programme en exécution
 - Possède des ressources de mémoire, périphériques, etc
- Ordonnancement de processus
- Opérations sur les processus
- Processus coopérants
- Processus communicants

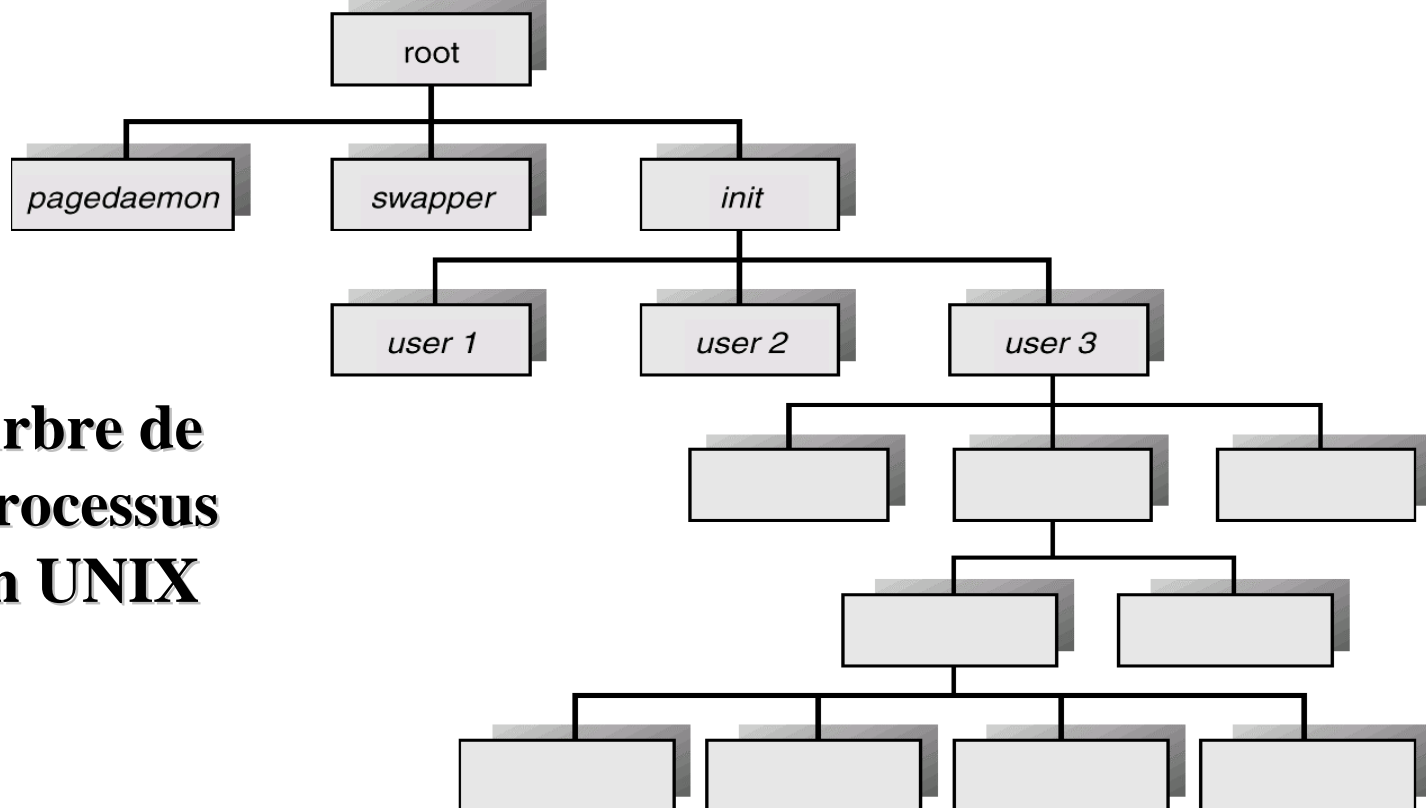
Espace de travail

- Les processus sont composés d'un espace de travail en mémoire formé de 3 segments :



Création de processus

- Les processus peuvent créer d'autres processus, formant une hiérarchie (instruction fork ou semblables)



Arbre de processus en UNIX

Terminaison de processus

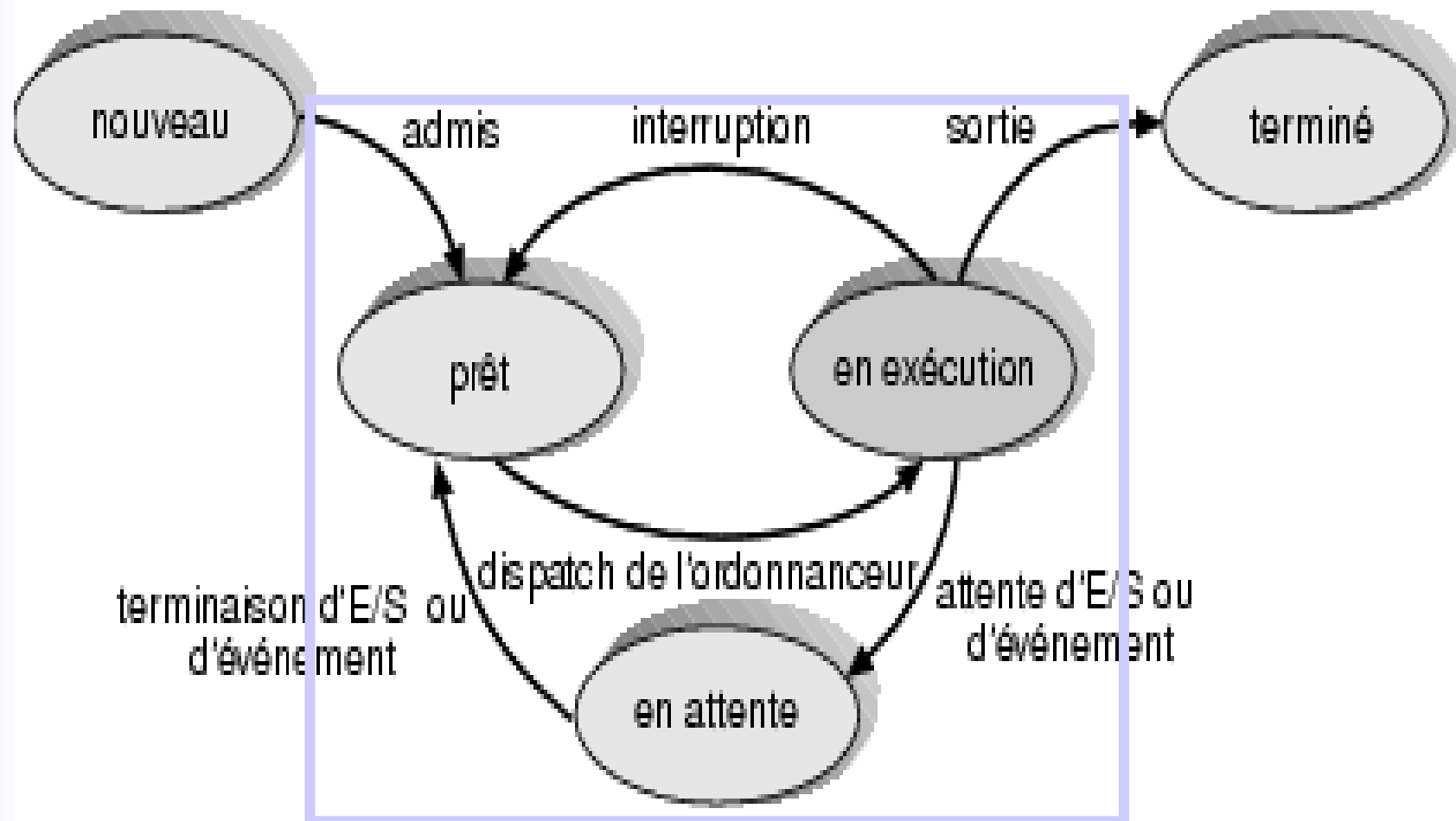
- Un processus exécute sa dernière instruction
 - pourrait passer des données à son parent
 - ses ressources lui sont enlevées
- Le parent termine l'exécution d'un fils (avortement) pour raisons différentes
 - le fils a excédé ses ressources
 - le fils n'est plus requis
- etc.

État de processus

IMPORTANT

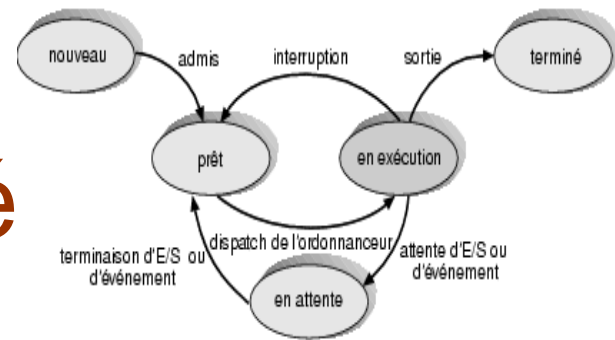
- Au fur et à mesure qu'un processus exécute, il change d'état
 - nouveau: le processus vient d'être créé
 - exécutant-running: le processus est en train d'être exécuté par l'UCT
 - attente-waiting: le processus est en train d'attendre un événement (p.ex. la fin d'une opération d'E/S)
 - prêt-ready: le processus est en attente d'être exécuté par l'UCT
 - terminated: fin d'exécution

Diagramme de transition d'états d'un processus



Ordonnanceur = angl. scheduler

États Nouveau, Terminé



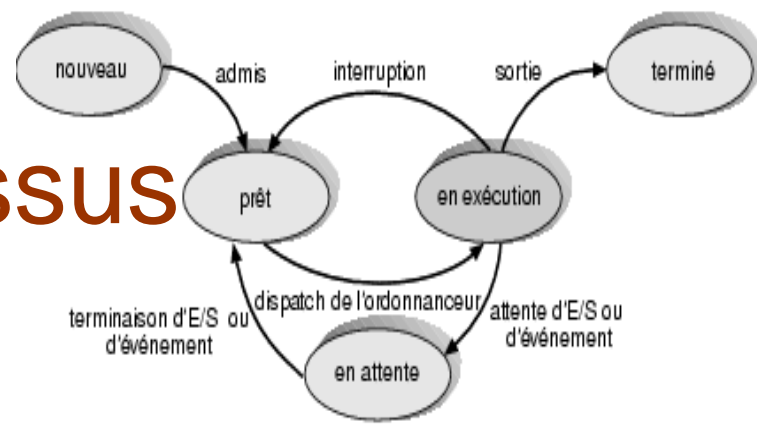
- Nouveau

- Le SE a créé le processus
 - a construit un identificateur pour le processus
 - a construit les tableaux pour gérer le processus
- mais ne s'est pas encore engagé à exécuter le processus (pas encore *admis*)
 - pas encore alloué des ressources
- La file des nouveaux travaux est souvent appelée **spoule travaux** (job spooler)

- Terminé:

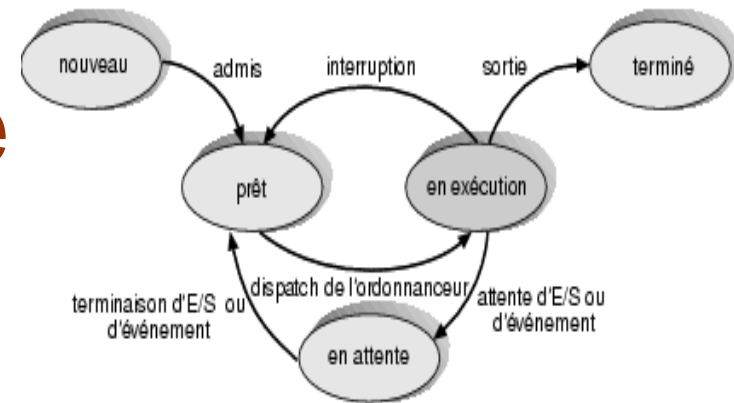
- Le processus n'est plus exécutable, mais ses données sont encore requises par le SE (comptabilité, etc.)

Transitions entre processus



- **Prêt → Exécution**
 - Lorsque l'ordonnanceur UCT choisit un processus pour exécution
- **Exécution → Prêt**
 - Résultat d'une interruption causée par un événement indépendant du processus
 - Il faut traiter cette interruption, donc le processus courant perd l'UCT
 - Cas important: le processus a épuisé son intervalle de temps (minuterie)

Transitions entre



- Exécution → Attente
 - Lorsqu'un processus fait requête d'un service du SE que le SE ne peut offrir immédiatement (interruption causée par le processus lui-même)
 - un accès à une ressource pas encore disponible
 - initie une E/S: doit attendre le résultat
 - a besoin de la réponse d'un autre processus
- Attente → Prêt
 - lorsque l'événement attendu se produit

Sauvegarde d'informations processus

- En multiprogrammation, un processus exécute sur l'UCT de façon intermittente
- Chaque fois qu'un processus reprend l'UCT (transition prêt → exécution) il doit la reprendre dans la même situation où il l'a laissée (même contenu de registres UCT, etc.)
- Donc au moment où un processus sort de l'état exécution il est nécessaire de sauvegarder ses informations essentielles, qu'il faudra récupérer quand il retourne à cet état

PCB = Process Control Block:

*Représente
la situation
actuelle
d'un
processus,
pour le
reprendre
plus tard*

pointeur	état de processus
numéro de processus	
compteur programme	
registres	
limites mémoire	
liste des fichiers ouverts	
⋮	

} Registres UCT

Process Control Block (PCB)

IMPORTANT

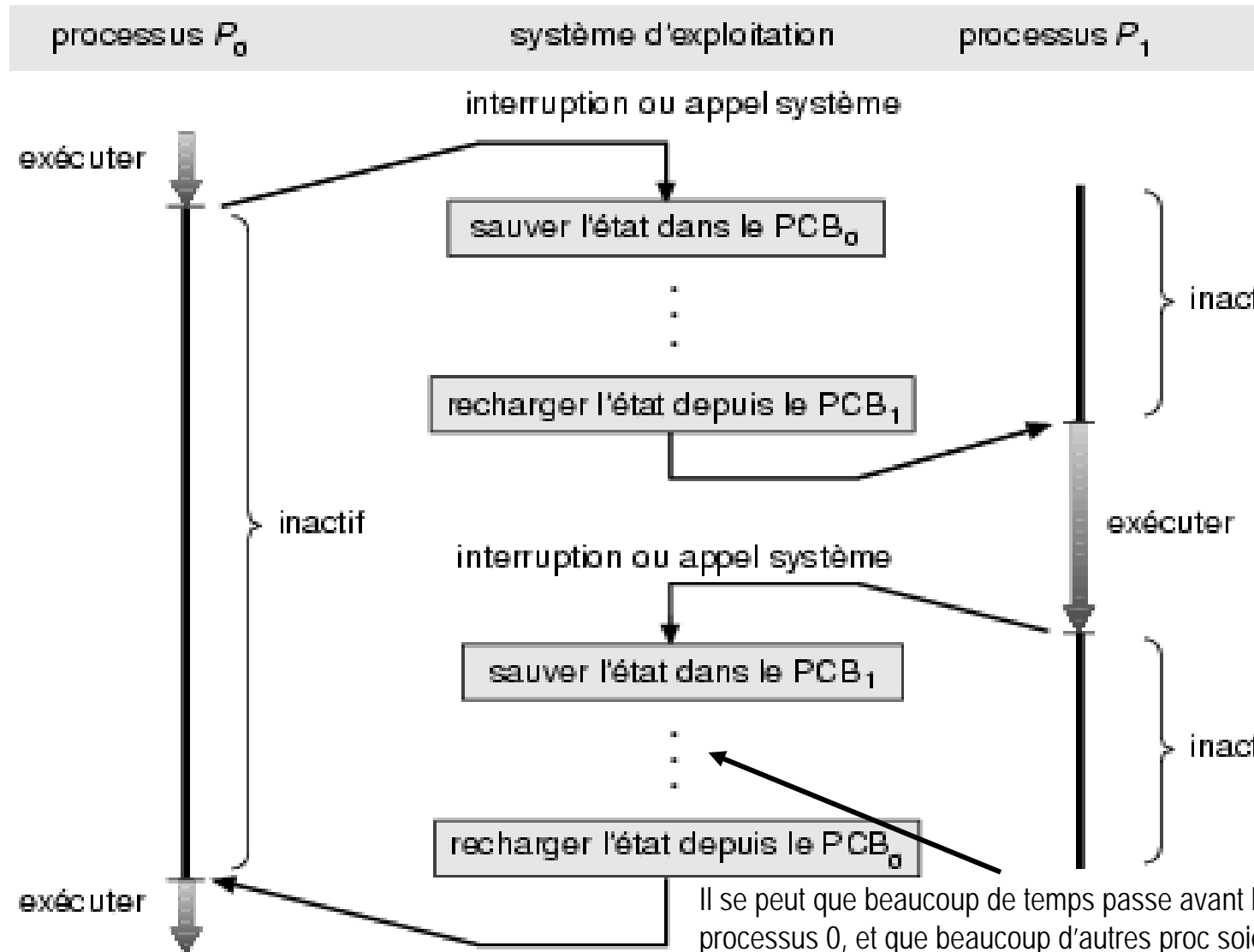
- pointeur: les PCBs sont rangés dans des listes enchaînées (à voir)
- état de processus: ready, running, waiting...
- compteur programme: le processus doit reprendre à l'instruction suivante
- autres registres UCT
- bornes de mémoire
- fichiers qu'il a ouvert
- etc.,

pointeur	état de processus
numéro de processus	
compteur programme	
registres	
limites mémoire	
liste des fichiers ouverts	
:	
:	

Commutation de processus

- Aussi appelé commutation de contexte ou context switching
- Quand l'UCT passe de l'exécution d'un processus 0 à l'exécution d'un proc 1, il faut
 - mettre à jour le PCB de 0
 - sauvegarder le PCB de 0
 - reprendre le PCB de 1, qui avait été sauvegardé avant
 - remettre les registres d'UCT, compteur d'instructions etc. dans la même situation qui est décrite dans le PCB de 1

Commutation de processeur (context switching)



Il se peut que beaucoup de temps passe avant le retour au processus 0, et que beaucoup d'autres proc soient exécutés dans entre temps

Le PCB n'est pas la seule information à sauvegarder

- Il faut aussi sauvegarder l'état des données du programme
- Ceci se fait normalement en gardant l'*image du programme* en mémoire primaire ou secondaire
- Le PCB pointera à cette image

Commutation de processus

Comme l'ordinateur n'a, la plupart du temps, qu'un processeur, il résout ce problème grâce à un pseudo-parallélisme

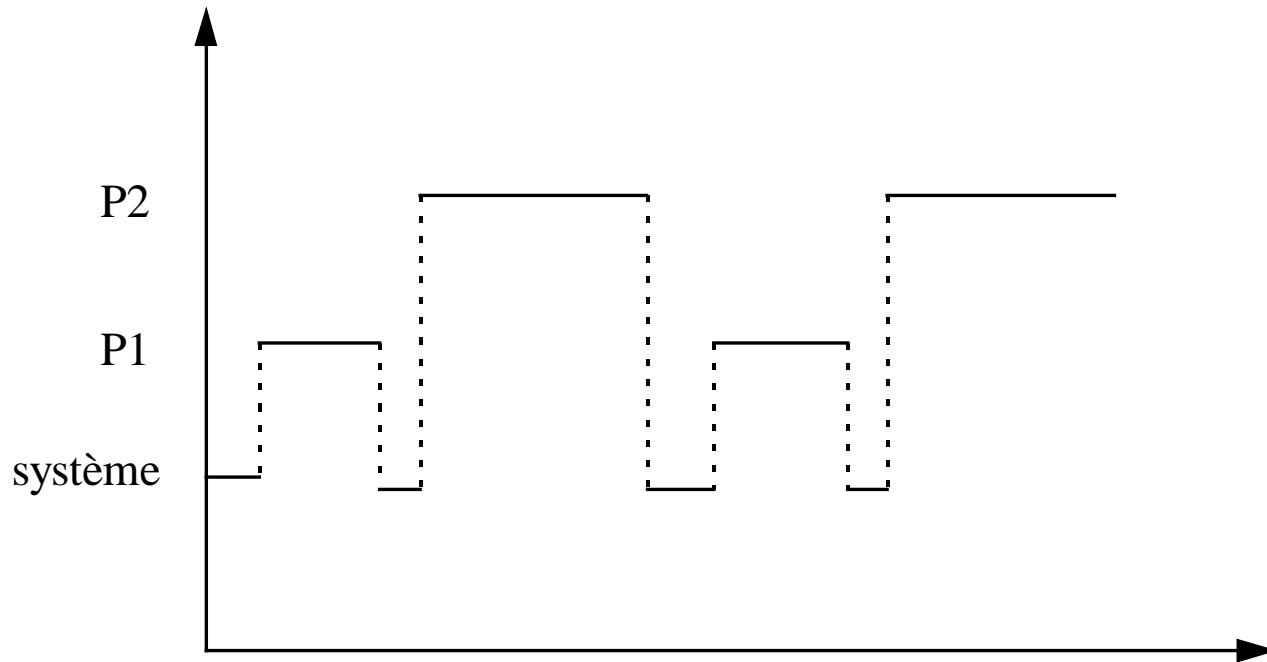


Figure 1 Le multi-tâche

Files d'attente

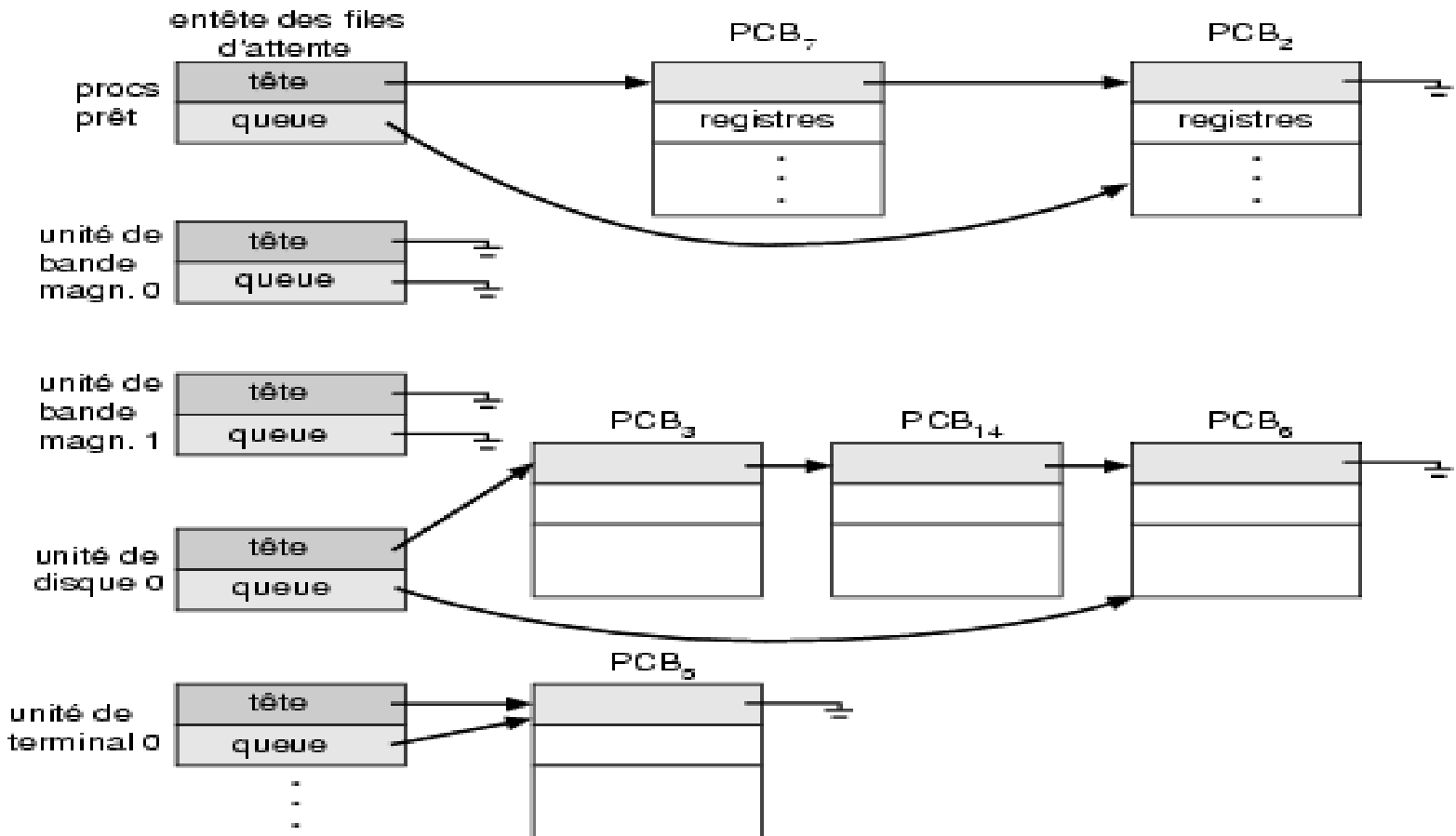
IMPORTANT

- Les ressources d'ordinateur sont souvent limitées par rapport aux processus qui en demandent
- Chaque ressource a sa propre file de processus en attente
- En changeant d'état, les processus se déplacent d'une file à l'autre
 - File prêt: les processus en état prêt=ready
 - Files associés à chaque unité E/S
 - etc.

Files d'attente

Ce sont les PCBs qui sont dans les files d'attente

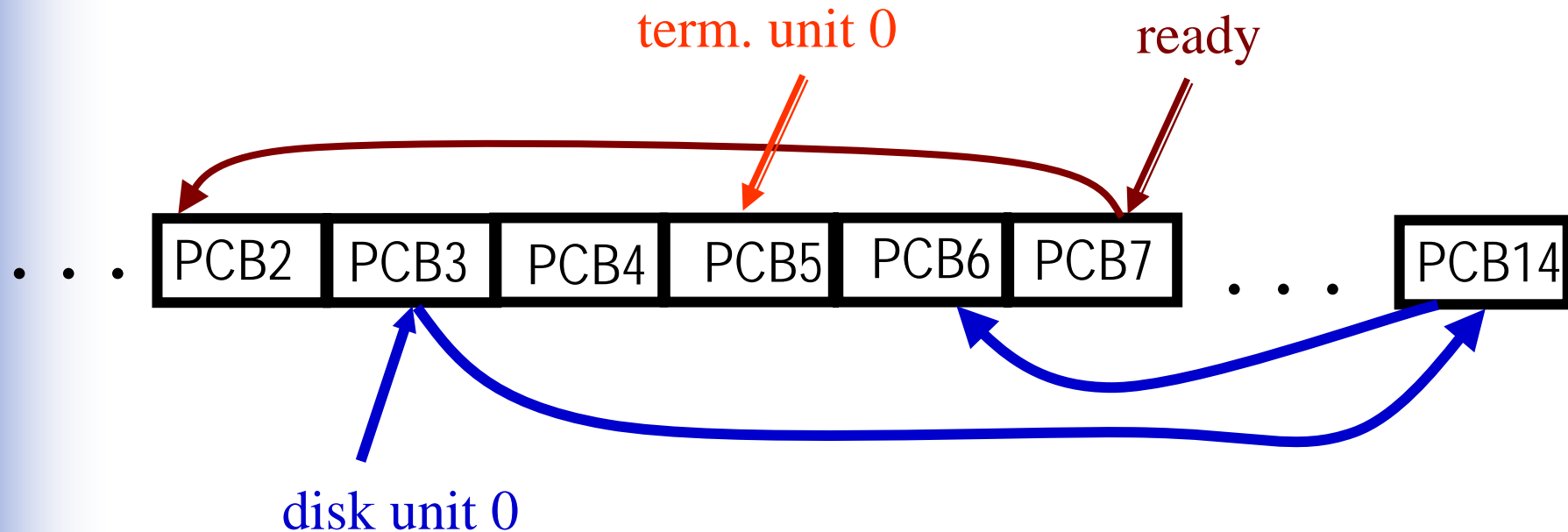
file prêt



Nous ferons l'hypothèse que le premier processus dans une file est celui qui utilise la ressource: ici, proc7 exécute, proc3 utilise disque 0, etc. 20

Les PCBs

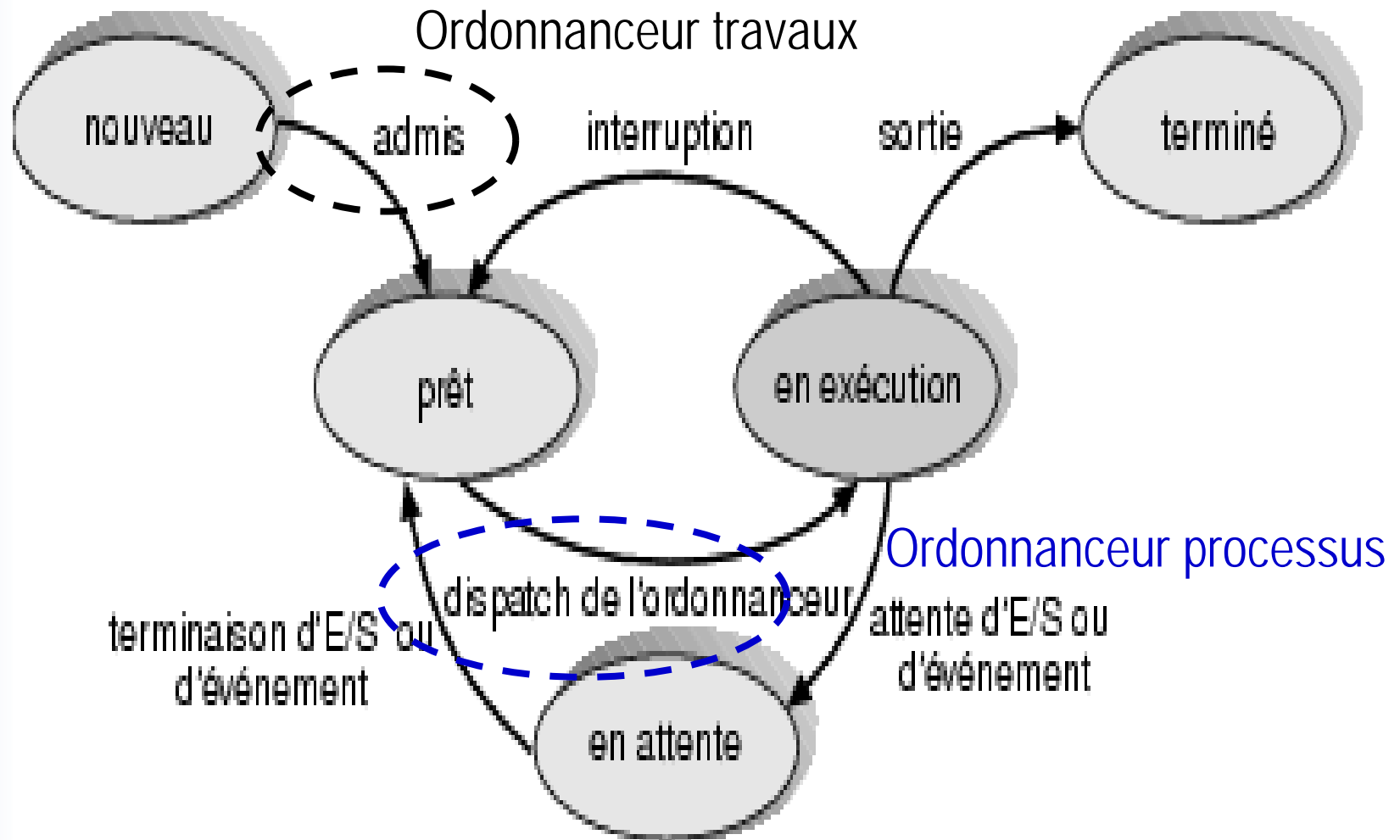
**Les PCBs ne sont pas déplacés en mémoire pour être mis dans les différentes files:
ce sont les pointeurs qui changent.**



Ordonnanceurs (schedulers)

- Programmes qui gèrent l'utilisation de ressources de l'ordinateur
- Trois types d'ordonnanceurs :
 - À court terme = **ordonnanceur processus**: sélectionne quel processus doit exécuter la transition **prêt** → **exécution**
 - À long terme = **ordonnanceur travaux**: sélectionne quels processus peuvent exécuter la transition **nouveau** → **prêt** (événement *admitted*) (de spoule travaux à file prêt)
 - À moyen terme: nous verrons

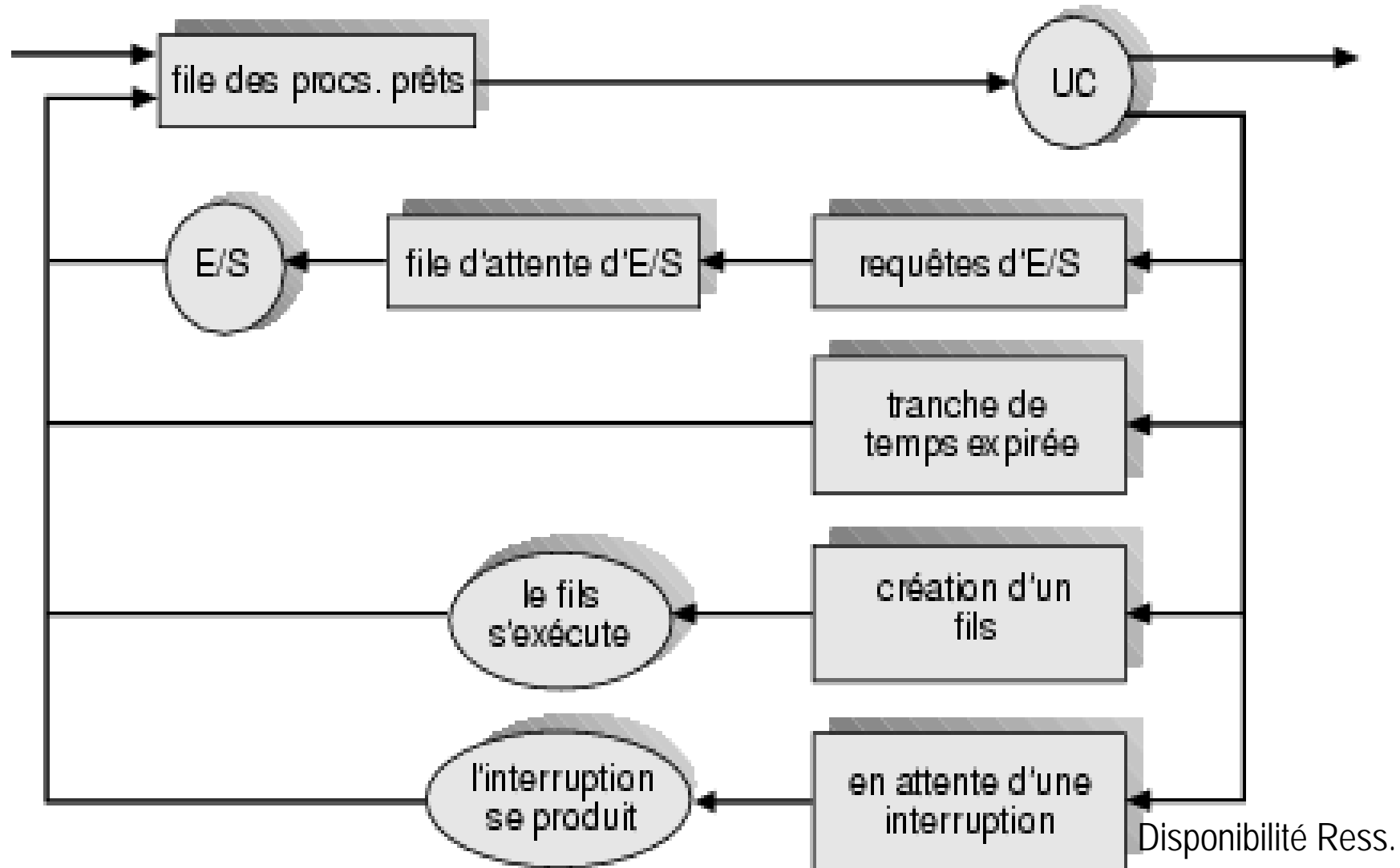
Ordonnanceur travaux = long terme et ordonnanceur processus = court



Ordonnanceurs

- L'ordonnanceur à court terme est exécuté très souvent (millisecondes)
 - doit être très efficace
- L'ordonnanceur à long terme doit être exécuté beaucoup plus rarement: il contrôle le niveau de multiprogrammation
 - doit établir une balance entre travaux liés à l'UCT et ceux liés à l'E/S
 - de façon que les ressources de l'ordinateur soient bien utilisées

Ordonnancement de processus

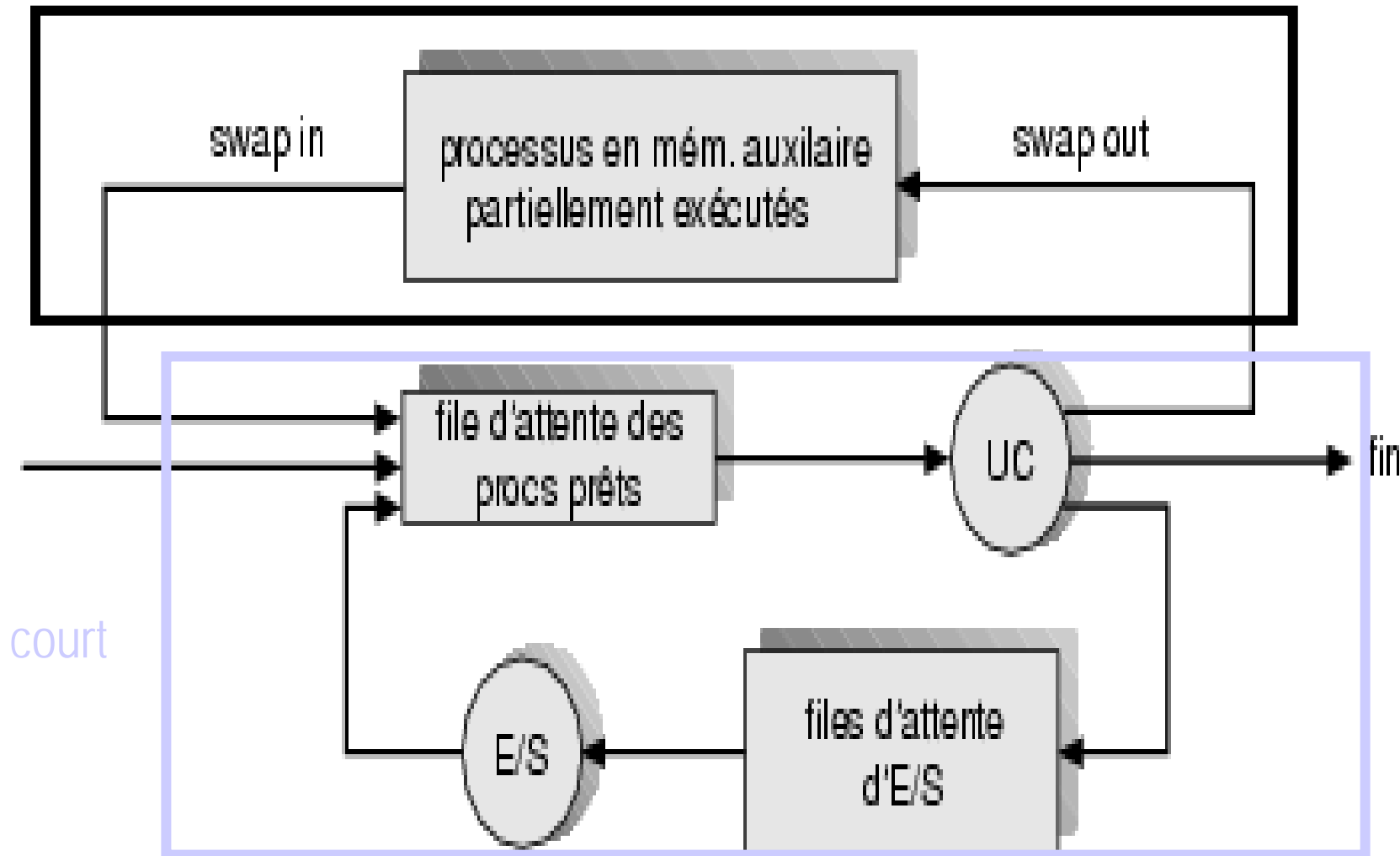


Ordonnanceur à moyen terme

- Le manque de ressources peut parfois forcer le SE à *suspendre* des processus
 - ils seront plus en concurrence avec les autres pour des ressources
 - ils seront repris plus tard quand les ressources deviendront disponibles
- Ces processus sont enlevés de mémoire centrale et mis en mémoire secondaire, pour être repris plus tard
 - ‘swap out’, ‘swap in’ , va-et-vient

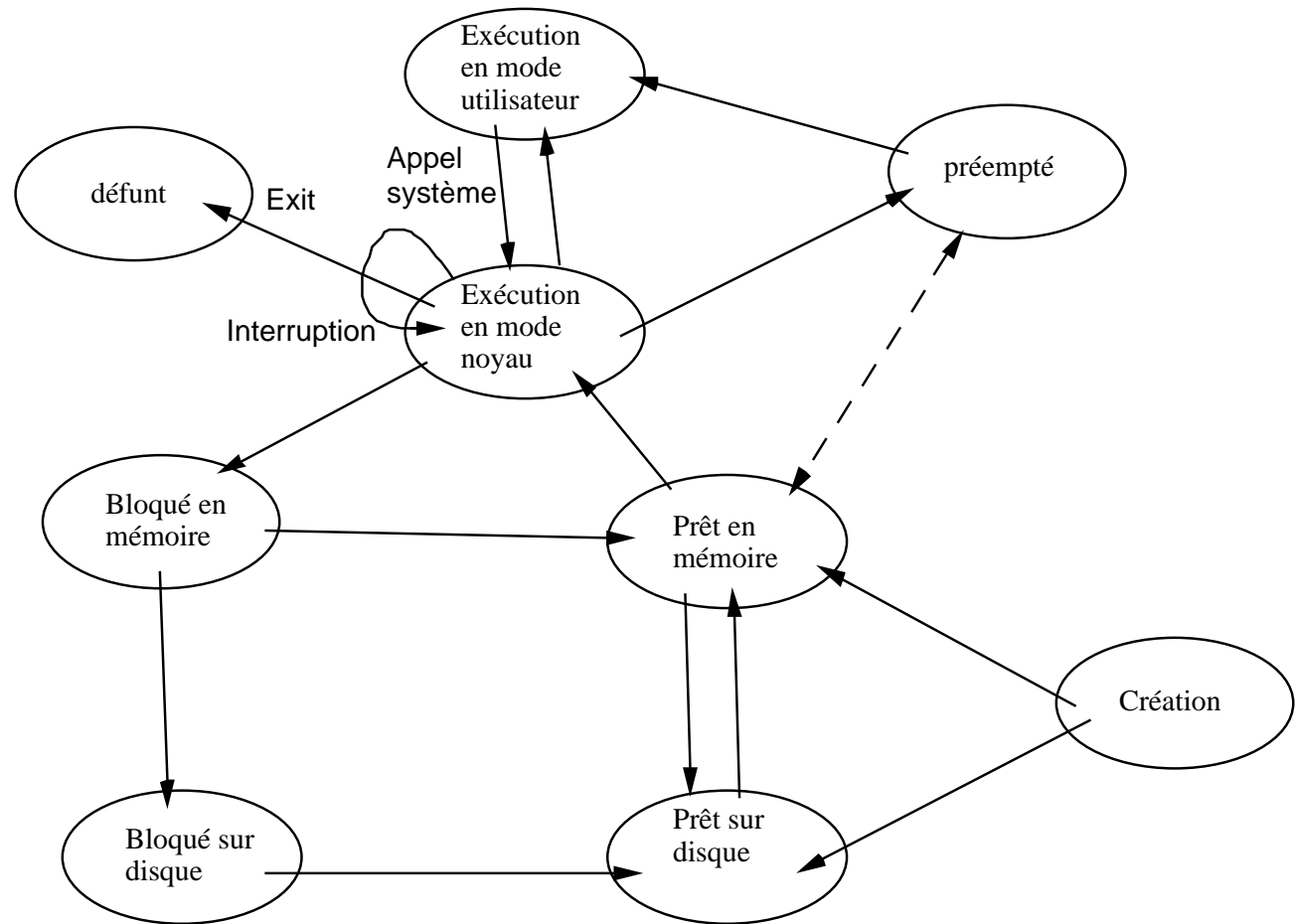
Ordonnanceurs à court et moyen terme

moyen



États de processus dans UNIX SVR4 (Stallings)

Un exemple de
diagramme de
transitions
d'états pour un
SE réel



Kernel, user mode =
monitor, user mode

Processus coopérants

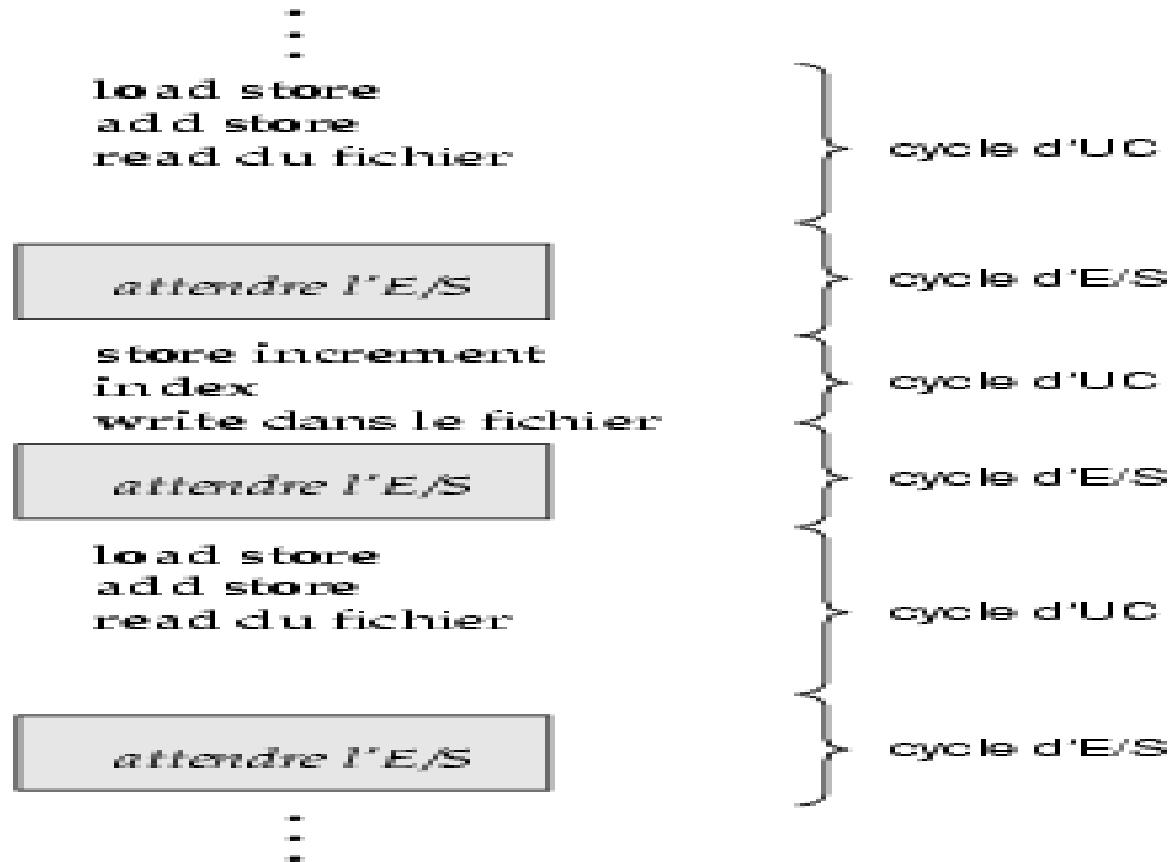
- Les processus coopérants peuvent affecter mutuellement leur exécution
- Avantages de la coopération entre processus:
 - partage de l'information
 - vitesse en faisant des tâches en parallèle
 - modularité
 - la nature du problème pourrait le demander

Ordonnancement Processus

Concepts de base

- La multiprogrammation est conçue pour obtenir une utilisation maximale des ressources, surtout l'UCT
- L'ordonnanceur UCT est la partie du SE qui décide quel processus dans la file ready/prêt obtient l'UCT quand elle devient libre
 - doit viser à une utilisation optimale de l'UCT
- L'UCT est la ressource la plus précieuse dans un ordinateur, donc nous parlons d'elle
 - Cependant, les principes que nous verrons s'appliquent aussi à l'ordonnancement des autres ressources (unités E/S, etc).

Les cycles d'un processus



- Cycles (bursts) d'UCT et E/S: l'exécution d'un processus consiste de séquences d'exécution sur UCT et d'attentes E/S

Quand invoquer l'ordonnanceur

- L'ordonnanceur doit prendre sa décision chaque fois que le processus exécutant est interrompu, c-à.-d.
 - un processus se présente en tant que nouveau ou se termine ou
 - un processus exécutant devient bloqué en attente
 - un processus change d'exécutant/running à prêt/ready
 - un processus change de attente à prêt/read
 - en conclusion, tout événement dans un système cause une interruption de l'UCT et l'intervention de l'ordonnanceur, qui devra prendre une décision concernant quel proc ou fil aura l'UCT après
- Préemption: on a préemption dans les derniers deux cas si on enlève l'UCT à un processus qui l'avait et peut continuer à s'en servir
- Dans les 1ers deux cas, il n'y a pas de préemption

Critères d'ordonnancement

- Il y aura normalement plusieurs processus dans la file prêt
- Quand l'UCT devient disponible, lequel choisir?
- L'idée générale est d'effectuer le choix dans l'intérêt de l'efficacité d'utilisation de la machine
- Mais cette dernière peut être jugée selon différents critères...

Critères d'ordonnancement

- Utilisation UCT: pourcentage d'utilisation
- Débit = Throughput: nombre de processus qui complètent dans l'unité de temps
- Temps de rotation = turnaround: le temps pris par le proc de son arrivée à sa terminaison.
- Temps d'attente: attente dans la file prêt (somme de tout le temps passé en file prêt)
- Temps de réponse (pour les systèmes interactifs): le temps entre une demande et la réponse

Critères d'ordonnancement:

maximiser/minimiser

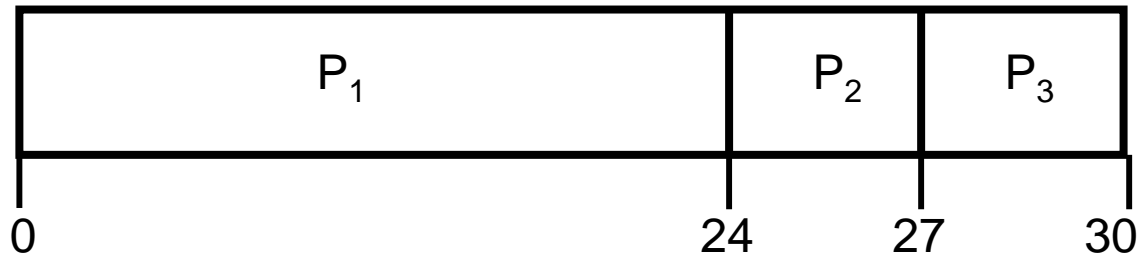
- Utilisation UCT: pourcentage d'utilisation
 - ceci est à maximiser
- Débit = Throughput: nombre de processus qui complètent dans l'unité de temps
 - ceci est à maximiser
- Temps de rotation (turnaround): temps terminaison moins temps arrivée
 - à minimiser
- Temps d'attente: attente dans la file prêt
 - à minimiser
- Temps de réponse (pour les systèmes interactifs): le temps entre une demande et la réponse
 - à minimiser

Premier arrive, premier servi (First come, first serve, FCFS)

Exemple:

<u>Processus</u>	<u>Temps de cycle</u>
P1	24
P2	3
P3	3

Si les processus arrivent au temps 0 dans l'ordre: P1 , P2 , P3
Le diagramme Gantt est:

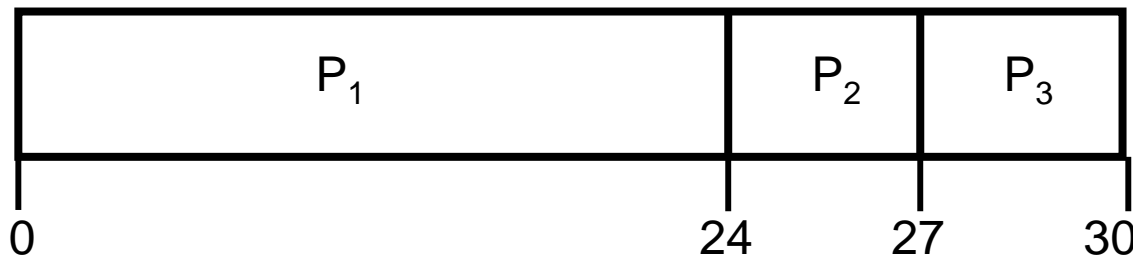


Temps d'attente pour P1= 0; P2= 24; P3= 27

Temps attente moyen: $(0 + 24 + 27)/3 = 17$

Premier arrive, premier servi

- Utilisation UCT = 100%
- Débit = $3/30 = 0,1$
 - 3 processus complétés en 30 unités de temps
- Temps de rotation moyen: $(24+27+30)/3 = 27$



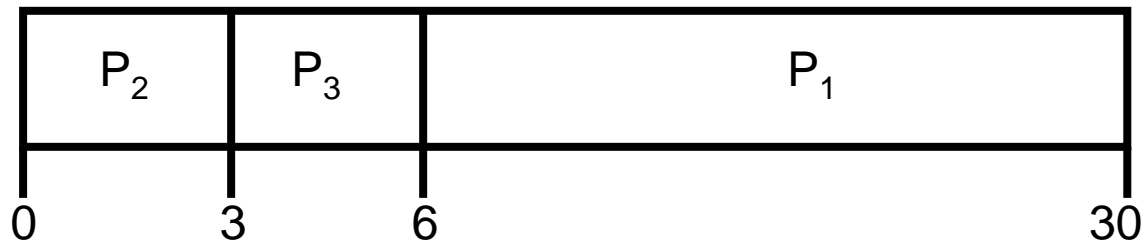
Tenir compte du temps d'arrivée!

- Dans le cas où les processus arrivent à moment différents, il faut soustraire les temps d'arrivée
- Exercice: répéter les calculs si:
 - P0 arrive à temps 0
 - P1 arrive à temps 2
 - P3 arrive à temps 5

FCFS Scheduling (Cont.)

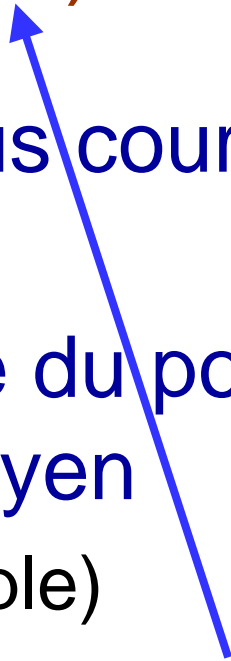
Si les mêmes processus arrivent à 0 mais dans l'ordre
 P_2, P_3, P_1 .

Le diagramme de Gantt est:



- Temps d'attente pour $P_1 = 6$ $P_2 = 0$ $P_3 = 3$
- Temps moyen d'attente: $(6 + 0 + 3)/3 = 3$
- Beaucoup mieux!
- Donc pour cette technique, le temps d'attente moyen peut varier grandement

Plus Court d'abord = Shortest Job First (SJF)

- Le processus le plus court part le premier
 - Optimal en principe du point de vue du temps d'attente moyen
 - (v. le dernier exemple)
 - Mais comment savons-nous
- 

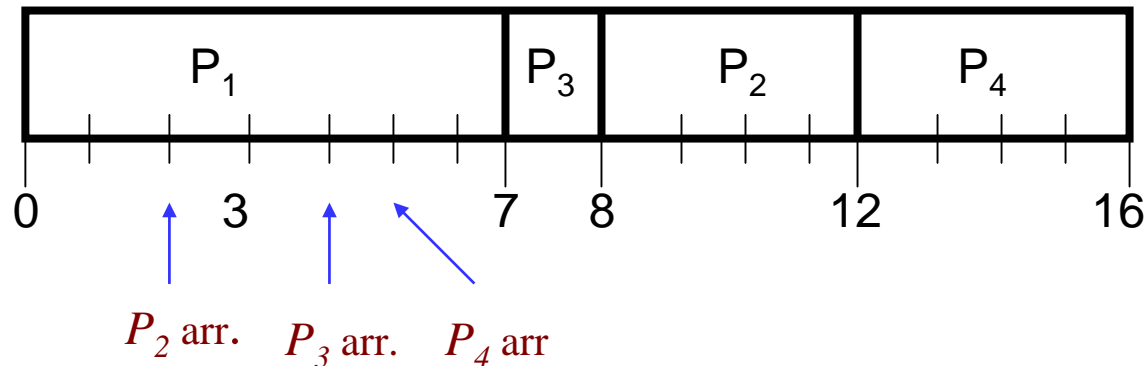
SJF avec préemption (réquisition) ou non

- Avec préemption: si un processus qui dure moins que le *restant* du processus courant se présente plus tard, l'UCT est donnée à ce nouveau processus
 - SRTF: shortest remaining-time first
- Sans préemption: on permet au processus courant de terminer son cycle
 - Observation: SRTF est plus logique car de toute façon le processus exécutant sera interrompu par l'arrivée du nouveau processus
 - Il est retourné à l'état prêt

Exemple de SJF sans préemption

<u>Processus</u>	<u>Arrivée</u>	<u>Cycle</u>
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (sans préemption)

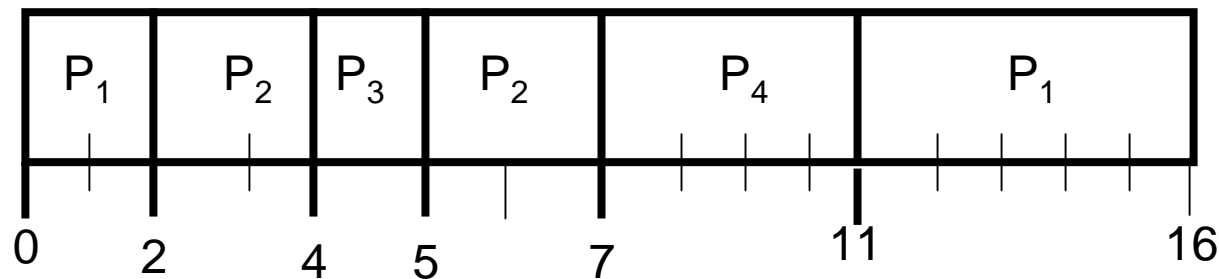


- Temps d'attente moyen = $(0 + 6 + 3 + 7)/4 = 4$

Exemple de SJF avec préemption SRTF

<u>Processus</u>	<u>Arrivée</u>	<u>Cycle</u>
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (préemptive)



P_2 arr. P_3 arr. P_4 arr

- Temps moyen d'attente = $(9 + 1 + 0 + 2)/4 = 3$
 - P_1 attend de 2 à 11, P_2 de 4 à 5, P_4 de 5 à 7

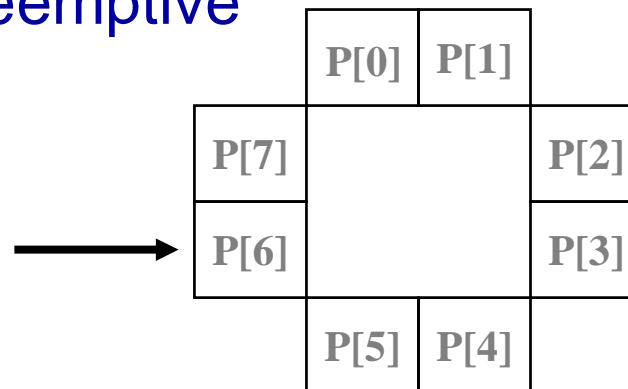
Le plus court d'abord SJF: critique

- Difficulté d'estimer la longueur à l'avance
- Les processus longs souffriront de *famine* lorsqu'il y a un apport constant de processus courts
- La préemption est nécessaire pour environnements à temps partagé
 - Un processus long peut monopoliser l'UCT s'il est le 1er à entrer dans le système et il ne fait pas d'E/S
- Il y a assignation implicite de priorités: préférences aux travaux plus courts

Tourniquet = Round-Robin (RR)

Le plus utilisé en pratique

- Chaque processus est alloué un quantum de temps (p.ex. 10-100 millisecs.) pour s'exécuter
 - (*tranche de temps*)
- S'il s'exécute pour un quantum entier sans autres interruptions, il est interrompu par la minuterie et l'UCT est donnée à un autre processus
- Le processus interrompu redevient prêt (à la fin de la file)
- Méthode préemptive

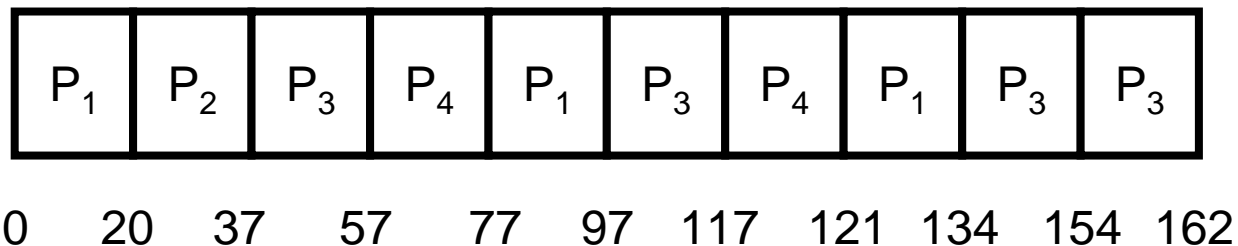


La file prêt est un cercle (dont RR)

Exemple: Tourniquet

Quantum = 20

<u>Processus</u>	<u>Cycle</u>
P_1	53
P_2	17
P_3	68
P_4	24



- Normalement,
 - temps de rotation (turnaround) plus élevé que SJF
 - mais temps attente moyen meilleur – contrôlez!

Priorités

- Affectation d'une priorité à chaque processus (p.ex. un nombre entier)
 - souvent les petits chiffres dénotent des hautes priorités
 - 0 la plus haute
- L'UCT est donnée au processus prêt avec la plus haute priorité
 - avec ou sans préemption
 - il y a une file *prêt* pour chaque priorité

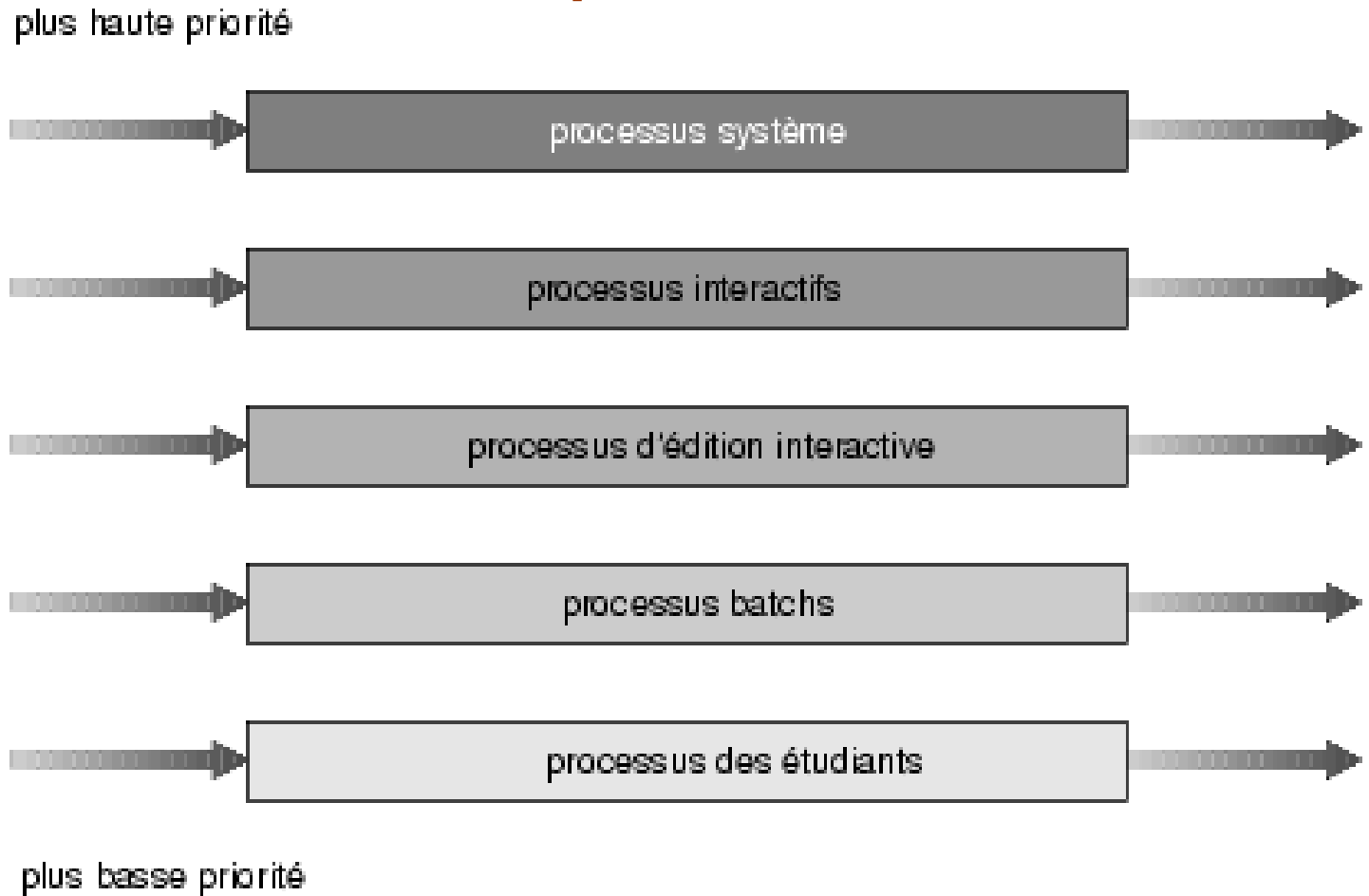
Problème possible avec les priorités

- Famine: les processus moins prioritaires n'arrivent jamais à exécuter
- Solution: vieillissement:
 - modifier la priorité d'un processus en fonction de son âge et de son historique d'exécution
 - le processus change de file d'attente
- Plus en général, la modification dynamique des priorités est une politique souvent utilisée

Files à plusieurs niveaux (multiples)

- La file *prêt* est séparée en plusieurs files, p.ex.
 - travaux 'd'arrière-plan' (background - batch)
 - travaux 'de premier plan' (foreground - interactive)
- Chaque file a son propre algorithme d'ordonnancement, p.ex.
 - FCFS pour arrière-plan
 - tourniquet pour premier plan
- Comment ordonnancer entre files?
 - Priorité fixe à chaque file → famine possible, ou
 - Chaque file reçoit un certain pourcentage de temps UCT, p.ex.
 - 80% pour arrière-plan
 - 20% pour premier plan

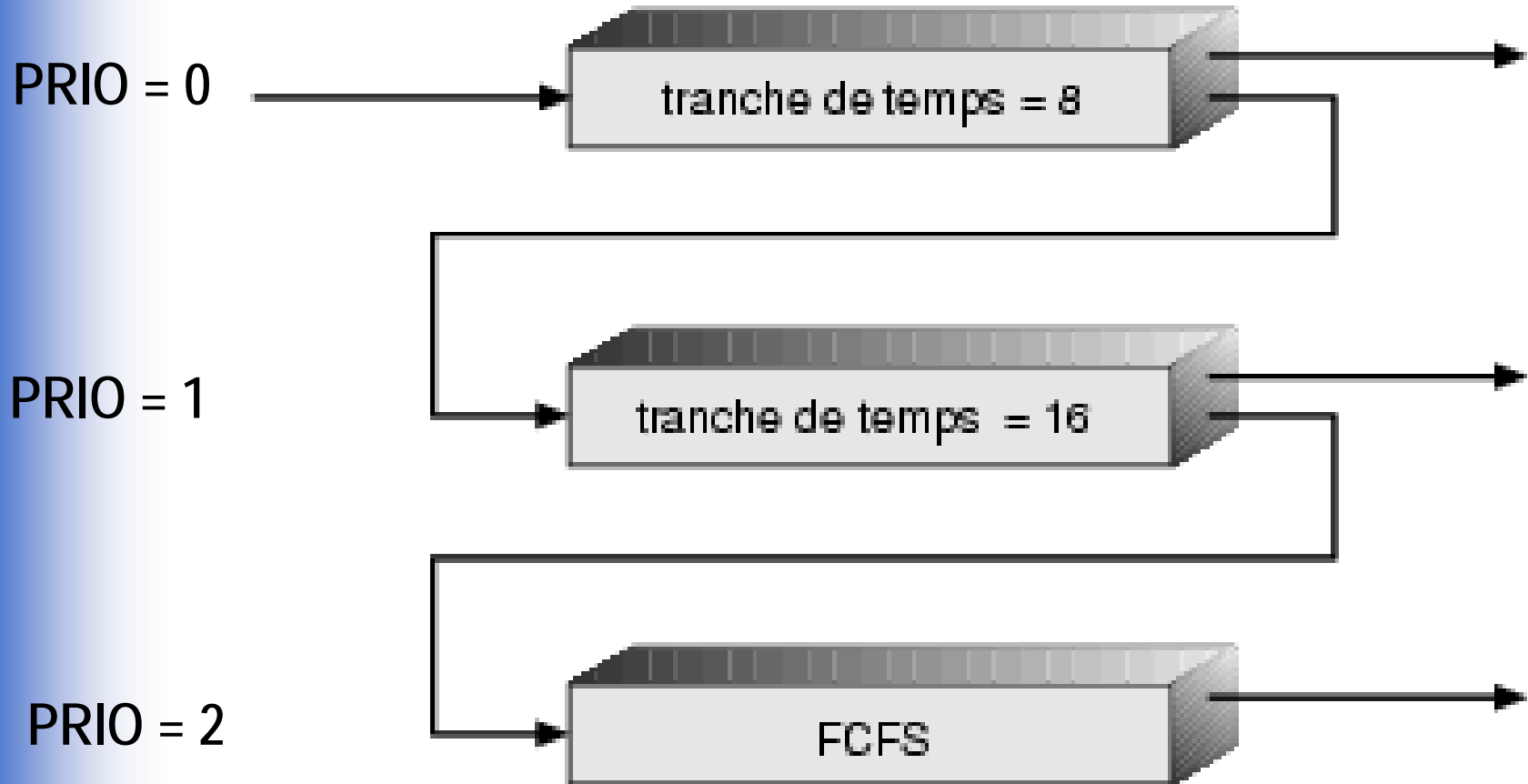
Ordonnancement avec files multiples



Files multiples et à retour

- Un processus peut passer d'une file à l'autre, p.ex. quand il a passé trop de temps dans une file
- À déterminer:
 - nombre de files
 - algorithmes d'ordonnancement pour chaque file
 - algorithmes pour décider quand un proc doit passer d'une file à l'autre
 - algorithme pour déterminer, pour un proc qui devient prêt, sur quelle file il doit être mis

Files multiples et à retour



Exemple de files multiples à retour

- Trois files:
 - Q0: tourniquet, quantum 8 msec
 - Q1: tourniquet, quantum 16 msec
 - Q2: FCFS
- Ordonnancement:
 - Un nouveau processus entre dans Q0, il reçoit 8 msec d'UCT
 - S'il ne finit pas dans les 8 msec, il est mis dans Q1, il reçoit 16 msec additionnels
 - S'il ne finit pas encore, il est interrompu et mis dans Q2
 - Si plus tard il commence à demander des quantum plus petits, il pourrait retourner à Q0 ou Q1

En pratique...

- Les méthodes que nous avons vu sont toutes utilisées en pratique (sauf plus court servi *pur* qui est impossible)
- Les SE sophistiqués fournissent au gérant du système une librairie de méthodes, qu'il peut choisir et combiner au besoin après avoir observé le comportement du système
- Pour chaque méthode, plusieurs params sont disponibles: p.ex. durée du quantum, coefficients, etc.

Aussi...

- Notre étude des méthodes d'ordonnancement est théorique, ne considère pas en détail tous les problèmes qui se présentent dans l'ordonnancement UCT
- P.ex. les ordonnanceurs UCT ne peuvent pas donner l'UCT à un processus pour tout le temps dont il a besoin
 - Car en pratique, l'UCT sera souvent interrompue par quelque événement externe avant la fin de son cycle
- Cependant les mêmes principes d'ordonnancement s'appliquent à unités qui ne peuvent pas être interrompues, comme une imprimante, une unité disque, etc.
- Dans le cas de ces unités, on pourrait avoir aussi des infos complètes concernant le temps de cycle prévu, etc.
- Aussi, cette étude ne considère pas du tout les temps d'exécution de l'ordonnanceur