



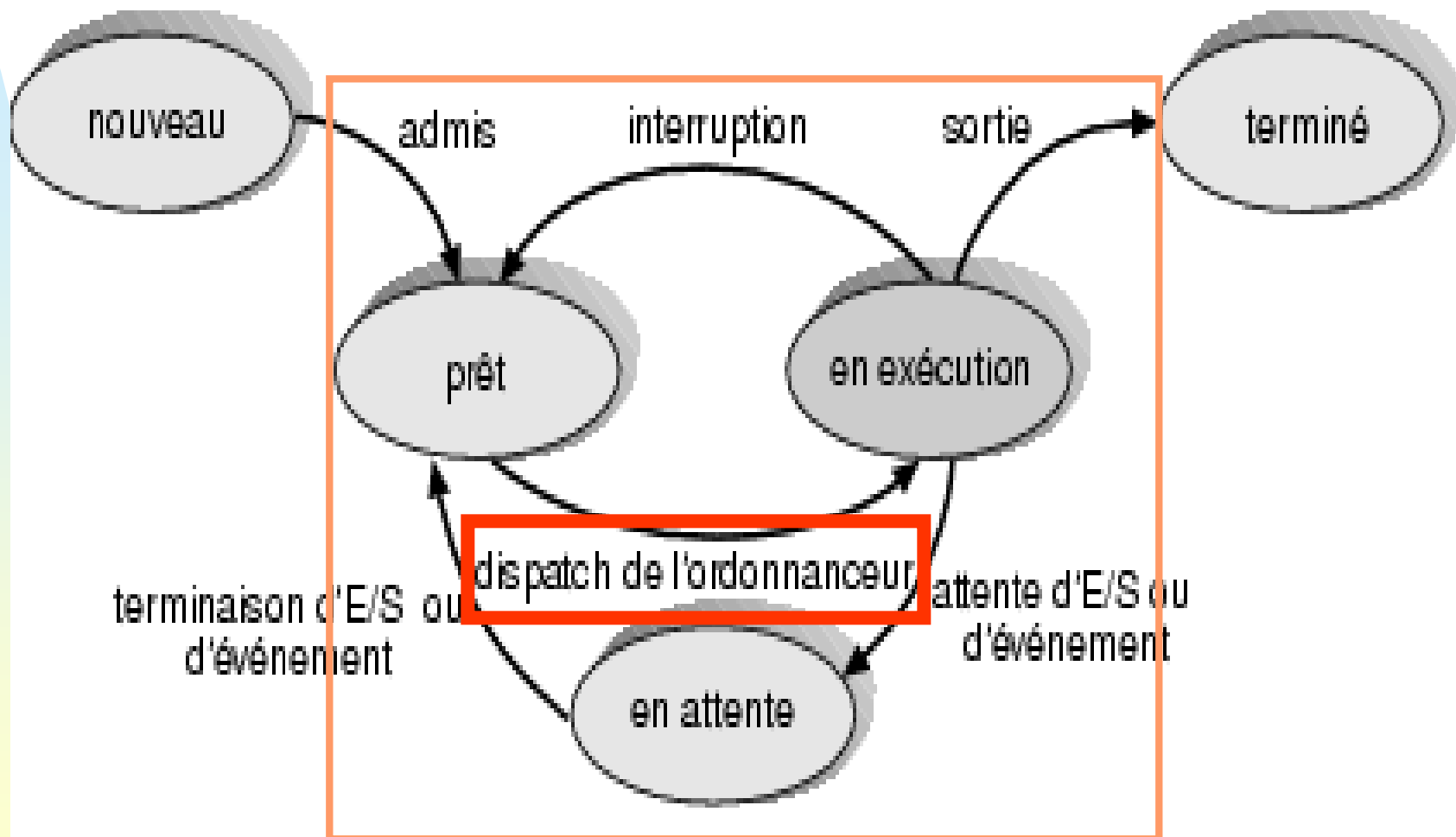
Module 4 - Ordonnancement Processus

Lecture: Chapitre 5

Aperçu du module

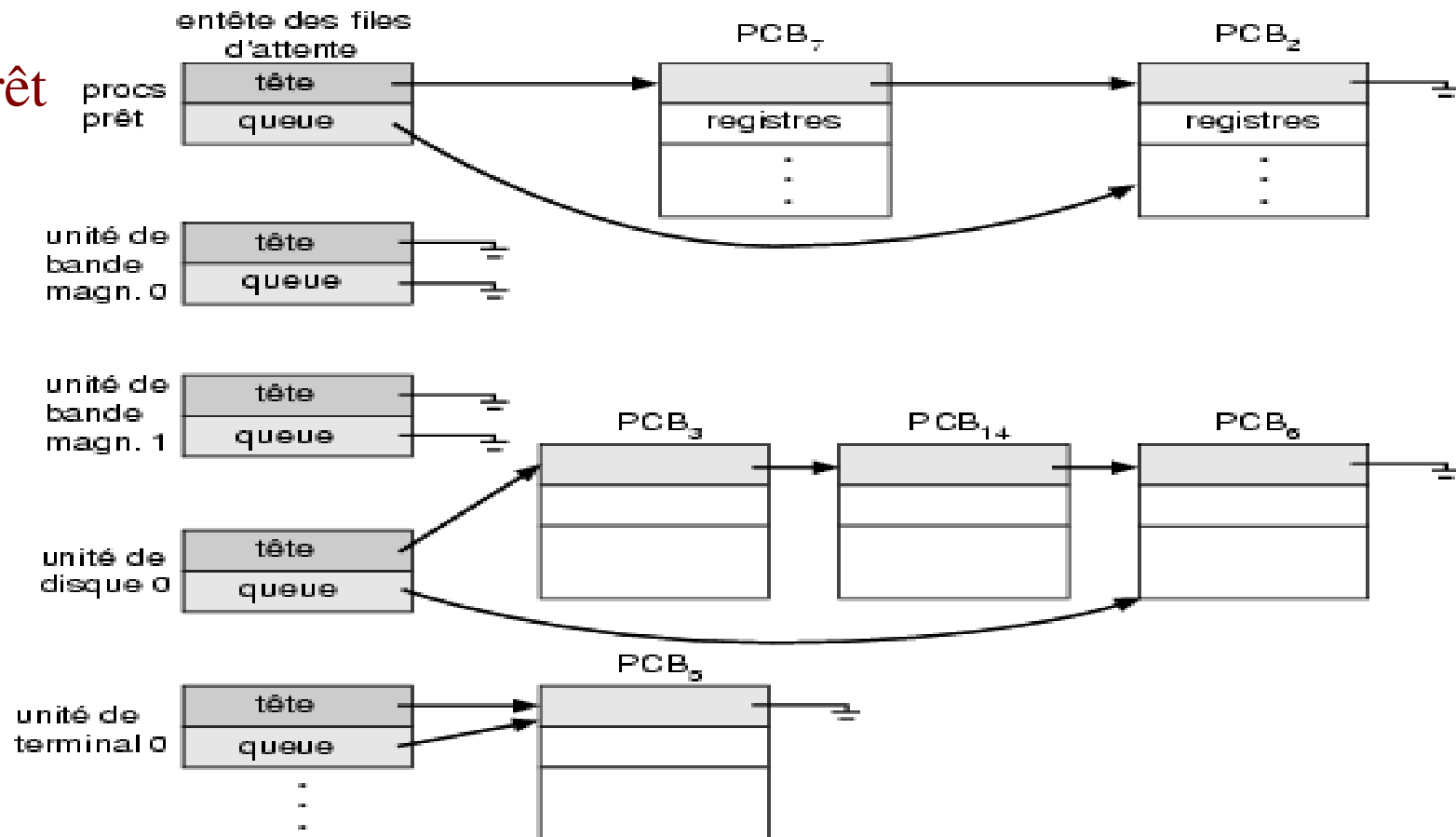
- **Concepts de base**
- **Critères d'ordonnancement**
- **Algorithmes d'ordonnancement**
- **Ordonnancement de multiprocesseurs**
- **Évaluation d'algorithmes**

Diagramme de transition d'états d'un processus



Files d'attente de processus pour ordonnancement

file prêt

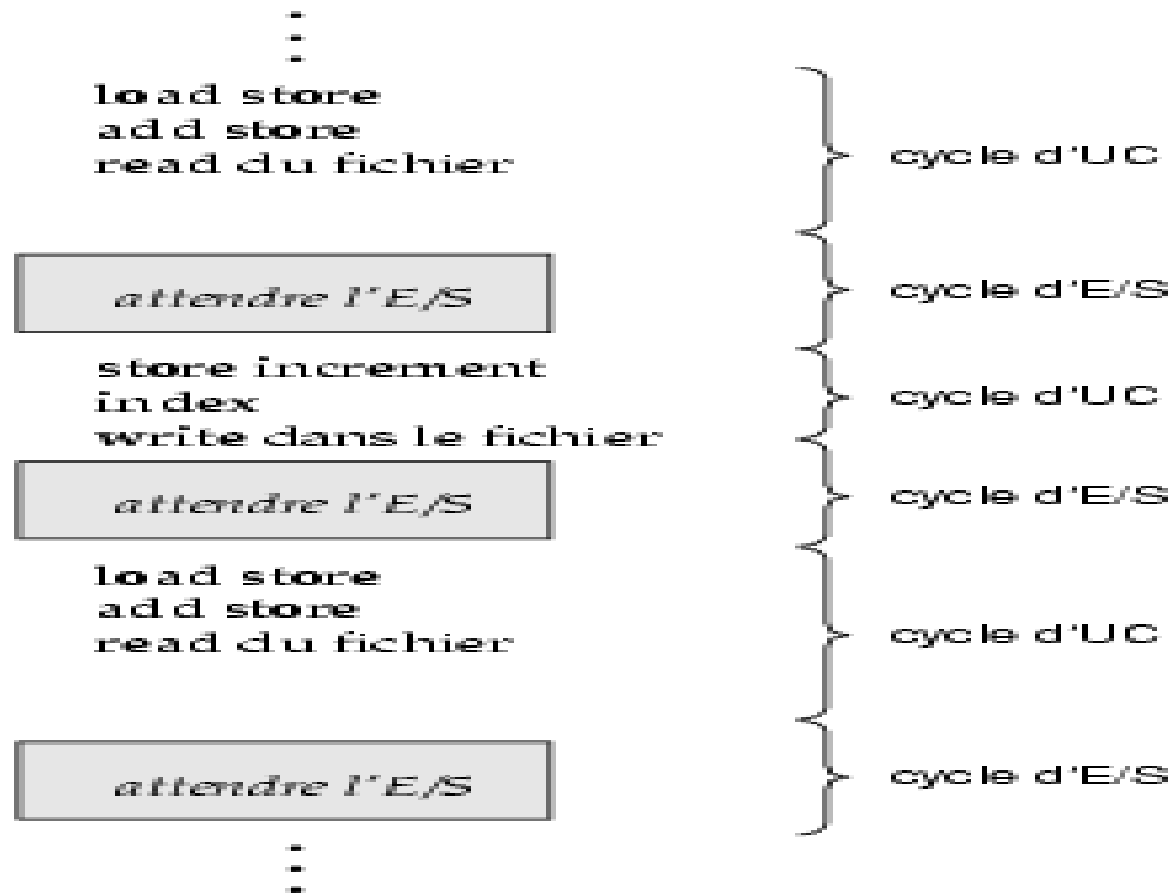


Nous ferons l'hypothèse que le premier processus dans une file est celui qui utilise la ressource: ici, proc7 exécute

Concepts de base

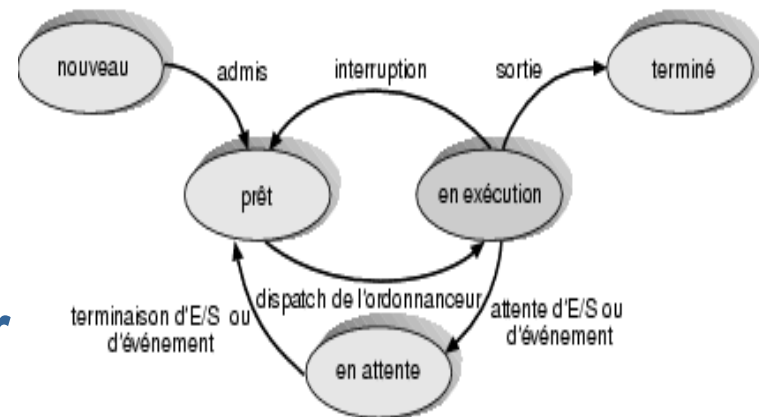
- **La multiprogrammation est conçue pour obtenir une utilisation maximale des ressources, surtout de l'UCT**
- **L'ordonnanceur UCT est la partie du SE qui décide quel processus dans la file ready/prêt obtient l'UCT quand elle devient libre**
 - ◆ Objectif: utilisation optimale de l'UCT
- **L'UCT est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

Les cycles d'un processus



- **Cycles/activités (bursts) de l'UCT et E/S: l'exécution d'un processus consiste de séquences d'exécution sur l'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 est **créé (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 processus ou fil aura l'UCT par la suite
- **Préférentiel**: on a préférence 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 préférence**
- Plusieurs problèmes à résoudre dans le cas préférentiel

Dispatcheur

- **Le module passe le contrôle de l'UCT au processus choisi par l'ordonnanceur à court terme; cela implique:**
 - ◆ changer de contexte
 - ◆ changer au mode usager
 - ◆ réamorcer le processus choisi
- **Temps de réponse de requête de processus (dispatcher latency)**
 - ◆ Le temps nécessaire au dispatcher d'arrêter un processus et de démarrer un autre
 - ◆ Souvent négligeable, il faut supposer qu'il soit petit par rapport à la longueur d'un cycle

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 un choix pour optimiser l'utilisation de la machine
- Mais cette dernière peut être jugée selon différents critères...

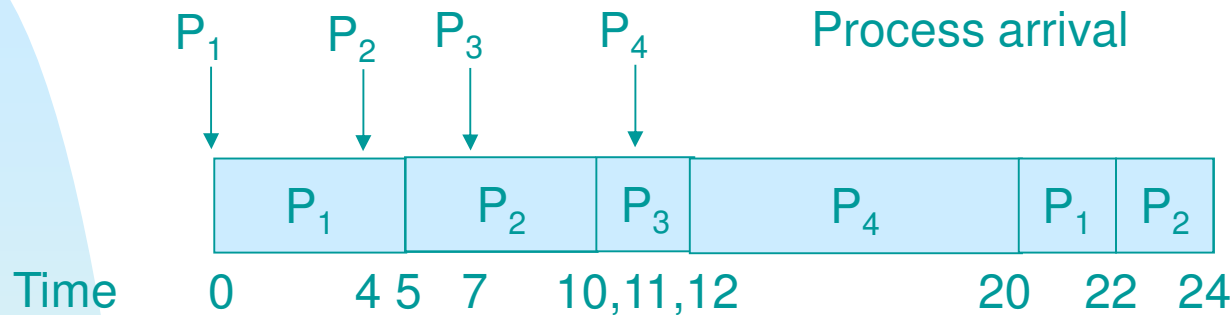
Critères d'ordonnancement

- **Raison principale pour l'ordonnancement**
 - ◆ Pourcentage d'utilisation: garder l'UCT et les modules E/S occupés aussi longtemps que possible
- **Systèmes à temps partagés?**
 - ◆ Temps de réponse (pour les systèmes interactifs): le temps entre une demande et la réponse
- **Serveurs?**
 - ◆ Débit (Throughput): nombre de processus qui achèvent leur exécution par unité de temps
- **Systèmes de traitement par lots (batch)?**
 - ◆ Temps de rotation (turnaround): le temps entre la soumission et la complétion d'un processus.
- **Systèmes chargés?**
 - ◆ Temps d'attente: le temps que passe un processus dans la file prêt

Critères d'ordonnement: maximiser/minimiser

- **À maximiser**
 - ◆ Utilisation de l'UCT
 - ◆ Débit
- **À minimiser**
 - ◆ Temps de réponse
 - ◆ Temps de rotation
 - ◆ Temps d'attente

Exemple de mesure des critères d'ordonnancement



En haut : temps de requete donc quand le processus est demander au CPU.
Ensuite les rectangles sont quand il est vraiment dans le CPU

- **Utilisation de l'UCT:**
 - ◆ 100%
- **Temps de réponse (P_3, P_2):** Temps avant la requete et quil est vraiment commencer d'executer dans le processus
 - ◆ $P_3: 3 = (10 - 7)$
 - ◆ $P_2: 1 = (5 - 4)$
- **Débit :** Nbr de process en un nbr de temps
 - ◆ $4/24$
- **Temps de rotation (P_3, P_2):** Temps avant la fin du programme et la requete au CPU
 - ◆ $P_3: 5 = (12 - 7)$
 - ◆ $P_2: 20 = (24 - 4)$
- **Temps d'attente (P_2):** Temps que le programme est en attente (Sois entre la requete et le debut du processus + temps entre interruption et recommencement de l'execution)
 - ◆ $P_2: 13 = (5 - 4) + (22 - 10)$



**Examinons plusieurs méthodes
d'ordonnancement et leurs comportements par
rapport aux critères utilisés**

nous étudierons des cas spécifiques

***l'étude du cas général demanderait recours à techniques probabilistes ou de
simulation***

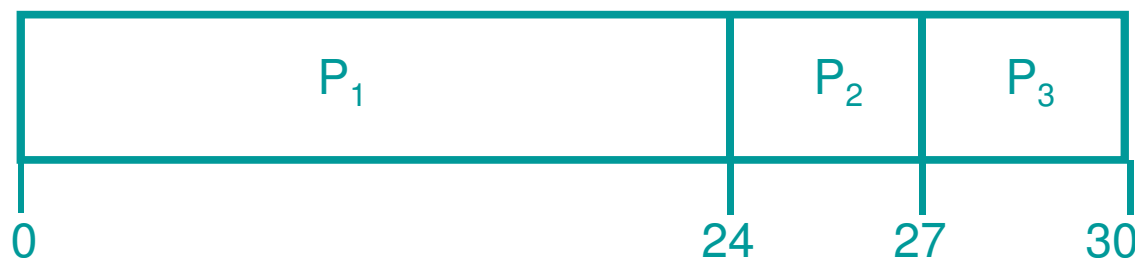
Premier arrivé, premier servi (FCFS)

- Notez, aucune préemption

Exemple: <u>Processus</u>	<u>Temps de cycle</u>	<u>Temps d'arrivée</u> = Temps de requete
P1	24	0 (premier)
P2	3	0 (second)
P3	3	0 (troisième)

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 arrivé, premier servi

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



Ordonnancement FCFS (suite)

Si les mêmes processus arrivaient 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
- *Exercice: calculer aussi le temps moyen de rotation, débit, etc.*

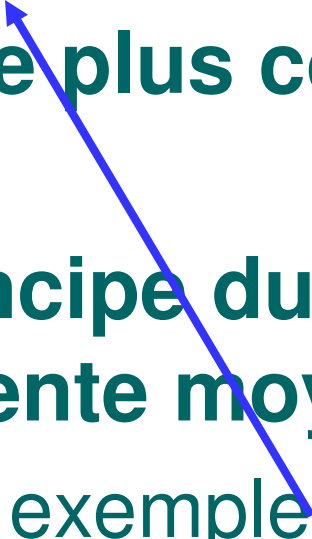
Tenir compte du temps d'arrivée!

- **Dans le cas où les processus arrivent à des moments différents, il faut soustraire les temps d'arrivées**
- **Exercice: répéter les calculs si:**
 - ◆ P2 arrive à temps 0
 - ◆ P1 arrive à temps 2
 - ◆ P3 arrive à temps 5

Effet d'accumulation (convoy effect) dans le FCFS

- Considérons un processus tributaire de l'UCT et plusieurs tributaires de l'E/S (situation assez normale)
- Les processus tributaires de l'E/S attendent l'UCT: les E/S sont sous-utilisées (*)
- Le processus tributaire de l'UCT demande une E/S: les autres processus exécutent rapidement leur cycle d'UCT et retournent sur l'attente E/S: l'UCT est sous-utilisée
- Le processus tributaire de l'UCT fini son E/S, puis les autres processus aussi : retour à la situation (*)
- Une solution: interrompre de temps en temps les processus tributaires de l'UCT pour permettre aux autres processus d'opérer (préférentiel)

Plus Court Job 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**
 - ◆ (voir le dernier exemple)
 - **Mais comment savons-nous**
- 

SJF avec préemption ou non

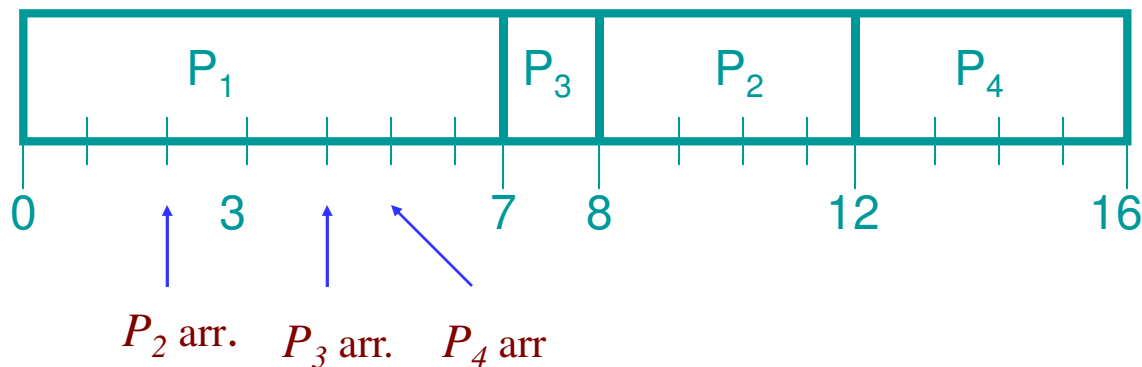
- **Avec préemption: si un processus qui dure moins que le *restant* du processus courant se présente plus tard, l'UCT est donnée à ce nouveau processus**
 - ◆ SRTF: shortest remaining-time first
- **Sans préemption: on permet au processus courant de terminer son cycle**
 - ◆ Observation: SRTF est plus 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

Pas seulement le temps d'exécution est important mais aussi le temps de la requête comme la P_3 est plus court que P_2 et arrive après sauf que P_1 n'est pas terminée donc P_3 passe avant P_2

■ SJF (sans préemption)

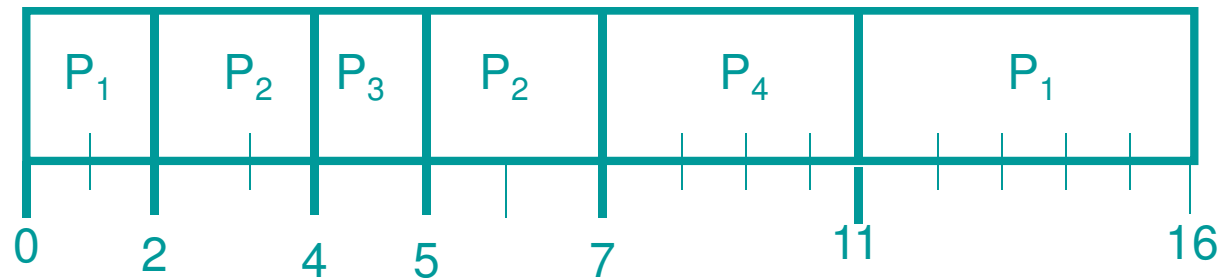


■ Temps d'attente moyen = $(0 + 6 + 3 + 7)/4 = 4$

Exemple de SJF avec 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 (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

Comment déterminer la longueur des cycles à l'avance?

- **Quelques méthodes proposent de déterminer le comportement futur d'un processus sur la base de son passé**
 - ◆ ex. moyenne exponentielle

Estimation de la durée du prochain cycle

- T_i : la durée du $i^{\text{ème}}$ cycle de l'UCT pour ce processus
- S_i : la valeur *estimée* du $i^{\text{ème}}$ cycle de l'UCT pour ce processus. Un choix simple est:
 - ◆ $S_{n+1} = (1/n) \sum_{i=1}^n T_i$ (une simple moyenne)
- Nous pouvons éviter de recalculer la somme en récrivant:
 - ◆ $S_{n+1} = (1/n) T_n + ((n-1)/n) S_n$
- Ceci donne un poids identique à chaque cycle

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**

Priorités

- **Affectation d'une priorité à chaque processus (par ex. un nombre entier)**
 - ◆ souvent les petits chiffres dénotent des hautes priorités (dans UNIX)
 - ☞ 0 la plus haute
 - ◆ Windows fait l'inverse – donne une plus haute priorité aux plus grands chiffres
- **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
- **Priorités sont explicites**
 - ◆ Pour raisons politiques ou techniques
- **Priorités implicites**
 - ◆ Voir SJF - critiques

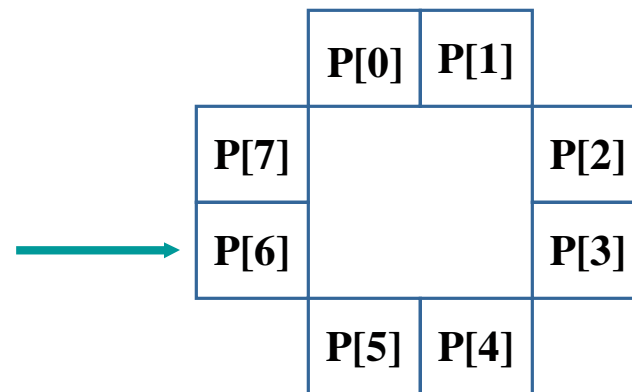
Problème possible avec les priorités

- **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
- **Généralement, la modification dynamique des priorités est une politique souvent utilisée** (files à rétroaction ou retour)
- **Que faire avec les processus de même priorités?**
 - ◆ FCFS
 - ◆ Ajoutons la préemption -> le Tourniquet

Tourniquet = Round-Robin (RR)

Le plus utilisé en pratique

- Chaque processus est alloué un intervalle de temps de l'UCT (ex. 10 à 100 millisecs.) pour s'exécuter
 - ◆ (terminologie du livre: *tranche de temps*)
- Après que ce temps s'est écoulé, le processus est interrompu, mis à la fin de la queue *prêt* et l'UCT est donnée au processus en tête de la queue
- Méthode préemptive



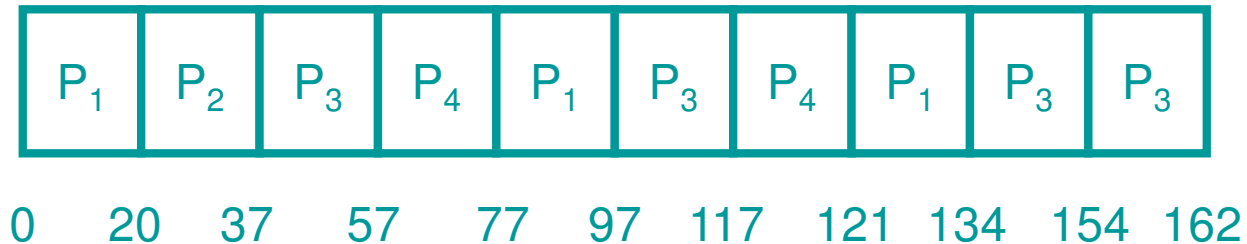
La file prêt est un cercle (RR)

Performance du tourniquet

- S'il y a n processus dans la file *prêt* et la tranche de temps est q , alors chaque processus reçoit $1/n$ du temps d'UCT en unités de durée maximale q
- Si q est grand \Rightarrow FCFS
- Si q est petit... à voir

Exemple: Tourniquet Quantum = 20

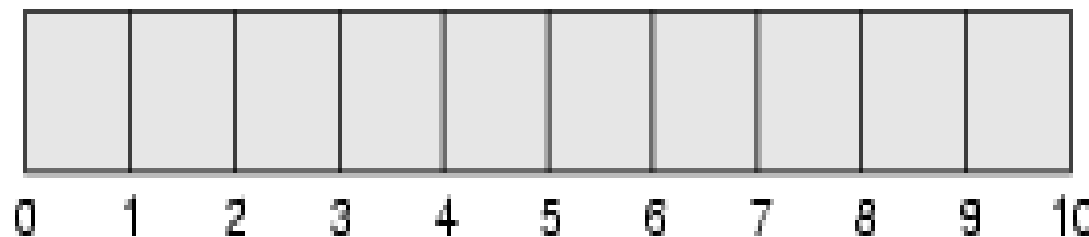
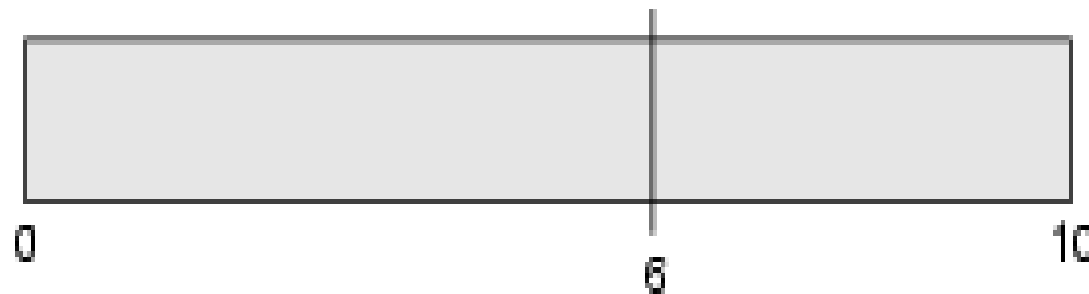
<u>Processus</u>	<u>Cycle</u>
P_1	53
P_2	17
P_3	68
P_4	24



- **Normalement,**
 - ◆ temps de rotation (turnaround) plus élevé que SJF
 - ◆ mais temps d'attente moyen meilleur

Un petit intervalle augmente les commutations de contexte (temps de SE)

temps du processus = 10



tranches de
temps

12

6

1

intermittence
changement
de contexte

0

1

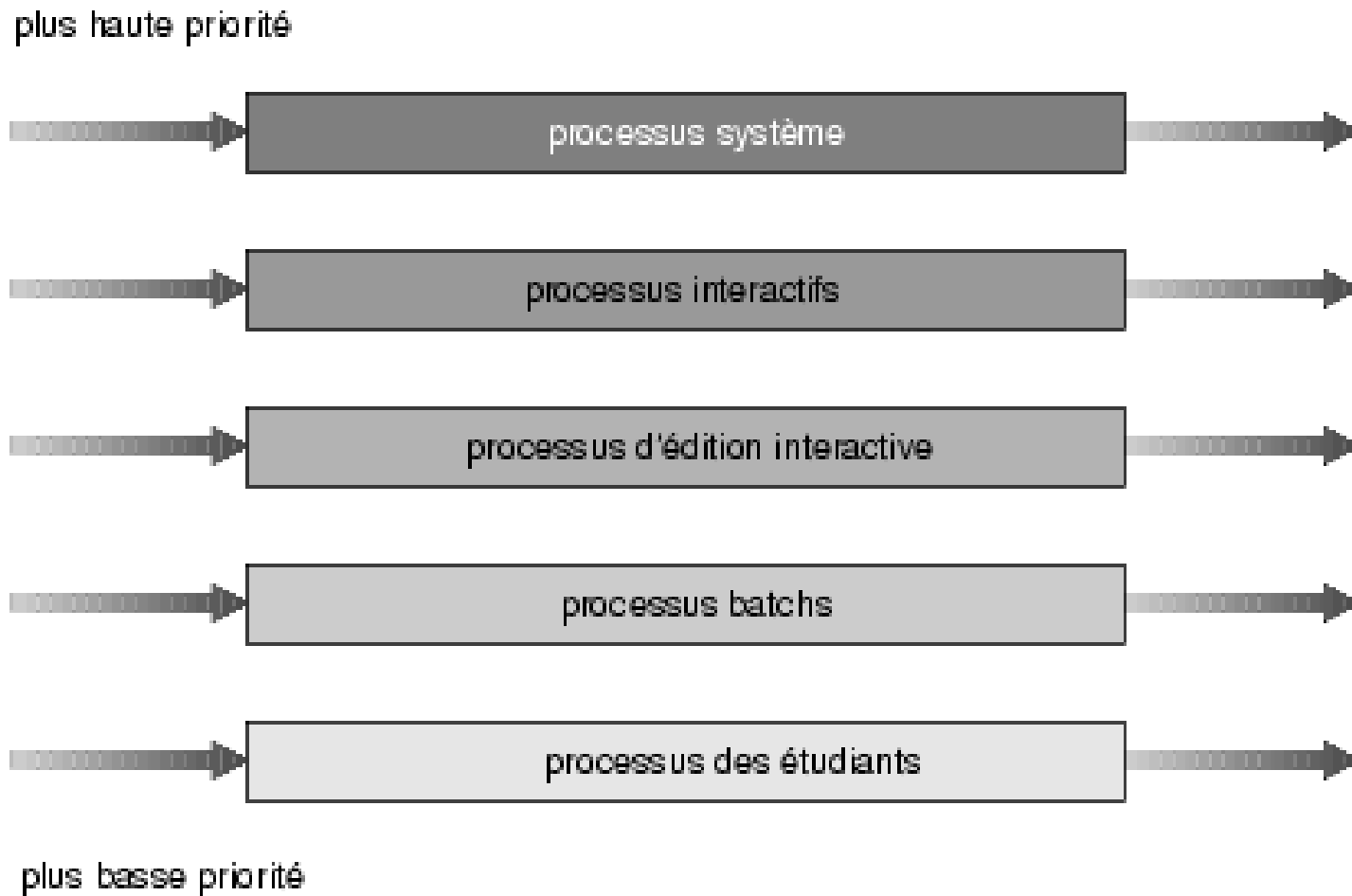
9

Pas d'intermittence

Queues/files à plusieurs niveaux (multiples)

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

Ordonnancement avec Queues multiples



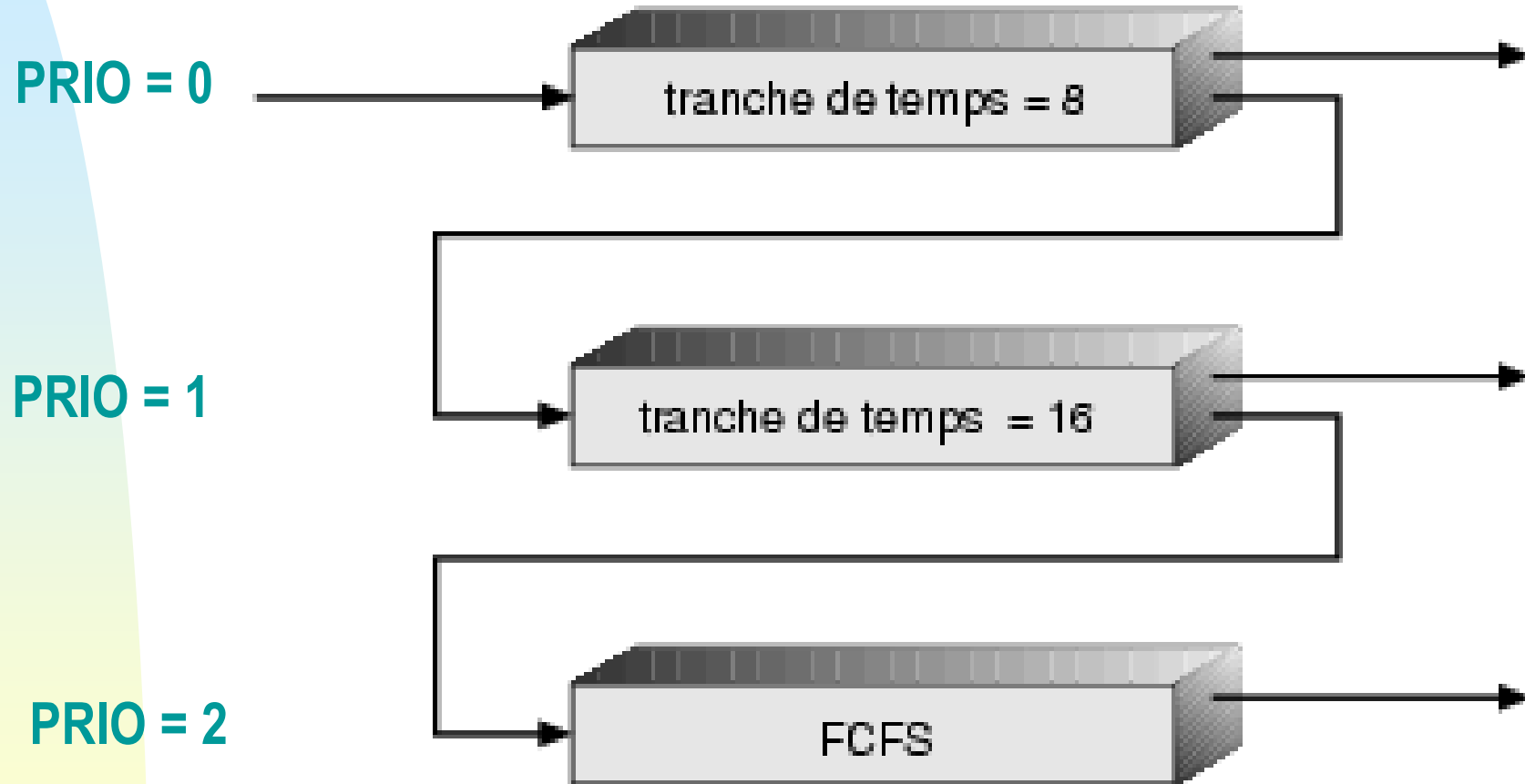
Queues multiples à rétroaction

- **Usage de queues à niveaux multiples**
- **Un processus peut passer d'une queue à une autre**
- **S'il utilise trop de temps d'UCT, il va dans une queue de moindre priorité**
- **Lorsqu'il est dans un état de famine d'utilisation d'UCT, il se déplace vers une queue de plus haute priorité, permet aussi d'établir son âge**

Queues multiples à rétroaction

- **Un organisateur de queues multiples à rétroaction est défini par les paramètres suivants:**
 - ◆ nombre de files
 - ◆ algorithmes d'ordonnancement pour chaque file
 - ◆ algorithmes pour décider quand promouvoir un processus
 - ◆ algorithmes pour décider quand rétrograder un processus
 - ◆ algorithme pour déterminer quelle queue utilisée lorsqu'un processus qui est *prêt* a besoin de services

Files multiples à rétroaction (trois files)



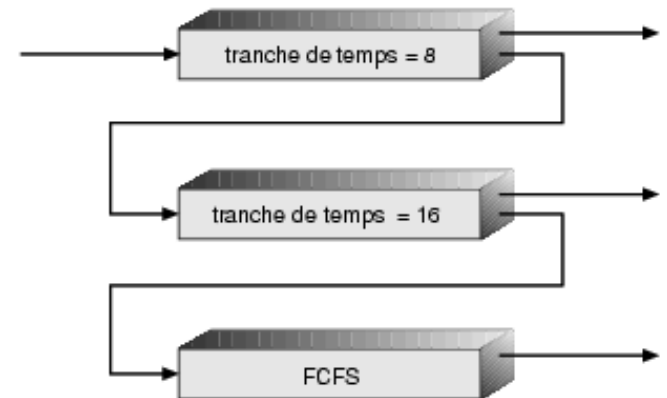
Exemple de queues multiples à rétroaction

- **Trois files:**

- ◆ Q0: tourniquet, tranche de 8 msec
- ◆ Q1: tourniquet, tranche de 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 n'a pas encore fini, 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 paramètres sont disponibles, ex. durée de l'intervalle de temps, coefficients, etc.

Aussi...

- **Notre étude des méthodes d'ordonnancement est théorique et ne considère pas en détail tous les problèmes qui se présentent dans l'ordonnancement UCT**
- **Par ex. les ordonnanceurs UCT ne peuvent pas donner l'UCT à un processus durant tout le temps dont il a besoin**
 - ◆ En pratique, l'UCT sera souvent interrompue par des événements externes avant la fin de son cycle
- **Cependant les mêmes principes d'ordonnancement s'appliquent aux 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.**
- **Cette étude aussi ne considère pas du tout les temps d'exécution de l'ordonnanceur**

Résumé des algorithmes d'ordonnancement

- **Premier arrivé, premier servi (FCFS)**
 - ◆ simple, court temps de système (over Head), de faibles qualités
- **Plus court d'abords (SJF)**
 - ◆ Doit savoir les temps de traitements (pas pratique)
 - ◆ Doit prédire en utilisant la moyenne exponentielle du passé
- **Ordonnancement avec priorité**
 - ◆ C'est une classe d'algorithmes
- **Tourniquet**
 - ◆ FCFS avec préemption
- **Queues à plusieurs niveaux (Multilevel Queues)**
 - ◆ Possibilité d'utiliser différents algorithmes dans chaque queue
- **Queues multiples à rétroaction (Multilevel Feedback Queues)**
 - ◆ Combine plusieurs concepts et techniques

Survol des sujets avancés de l'ordonnancement

- **L'ordonnancement avec plusieurs UCTs identiques**
- **Modèle d'évaluation**

Ordonnancement avec plusieurs UCTs identiques: *homogénéité*

- Une seule liste *prêt* pour toutes les UCTs (division du travail = load sharing)
 - ☞ une liste séparée pour chaque UCT ne permettrait pas cela
 - ◆ méthodes symétriques: chaque UCT peut exécuter l'ordonnancement et la répartition
 - ◆ méthodes asymétriques: ces fonctions sont réservées à une seule UCT

Solaris 2:

- **Priorités et préemption**
- **Queues à multiniveaux à rétroaction avec changement de priorité**
- **Différentes tranches par niveau de priorité (plus grands pour les priorités plus élevées)**
- **Priorité élevée pour les processus interactifs, plus petite pour les processus tributaires de l'UCT**
- **La plus haute priorité aux processus en temps réel**
- **Tourniquet pour les fils de priorités égales**

Méthode d'évaluation et comparaison d'algorithmes (section plutôt à lire)

- **Modélisation déterministe**
- **Modèles de queues d'attente (queuing theory)**
- **Simulation**

Modélisation déterministe

- **Essentiellement, ce que nous avons déjà fait en étudiant le comportement de plusieurs algorithmes sur plusieurs exemples**

Utilisation de la théorie des files (queuing th.)

- **Méthode analytique basée sur la théorie des probabilités**
- **Modèle simplifié: notamment, les temps du SE sont ignorés**
- **Cependant, elle permet d'obtenir des estimations**

Théorie des files: la formule de Little

- **Un résultat important:**

$$n = \lambda \times W$$

- ◆ n : longueur moyenne de la queue d'attente
- ◆ λ : débit d'arrivée de travaux dans la queue
- ◆ W : temps d'attente moyen dans la queue

- **Exemple.**

- ◆ λ si les travaux arrivent 3 par sec.
- ◆ W et il restent dans la file 2 secs
- ◆ n la longueur moyenne de la file sera???

- **Exercice: Résoudre aussi pour λ et W**

- **Observer qu'afin que n soit stable, $\lambda \times W$ doit être stable**

- ◆ Un débit d'arrivée plus rapide doit impliquer un temps de service mineur, et vice-versa
 - ☞ Si n doit rester à 6 et que λ monte à 4, quel doit être W ?

Simulation

- Construire un modèle (*simplifié...*) de la séquence d'événements dans le SE
- Attribuer une durée de temps à chaque événement
- Supposer une certaine séquence d'événements extérieurs (par ex. l'arrivée de travaux, etc.)
- Exécuter le modèle pour cette séquence afin d'obtenir des statistiques

Points importants dans ce chapitre

- **Queues d'attente pour l'UCT**
- **Critères d'ordonnancement**
- **Algorithmes d'ordonnancement**
 - ◆ FCFS: simple, non optimal
 - ◆ SJF: optimal, implémentation difficile
 - ☞ Procédé de moyenne exponentielle
 - ◆ Priorités
 - ◆ Tourniquet: sélection de la tranche de temps
- **Évaluation des méthodes, théorie des files,**
 - ◆ formule de Little