



Gestion de processus

Ordonnancement de l'exécution

Violeta Felea

violeta dot felea at femto-st dot fr

Département des Enseignements
Informatiques – UFR ST

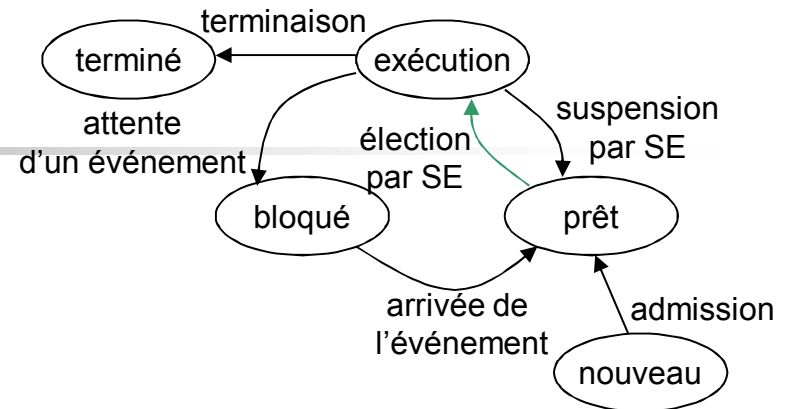




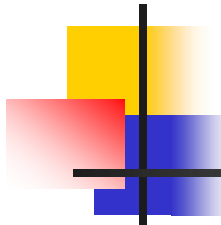
Ordonnancement de l'exécution

- Définition
- Contexte : multiprogrammation
- Cycle d'exécution
- Ordonnancement
 - préemptif
 - non préemptif

Sauvegarde des informations du processus



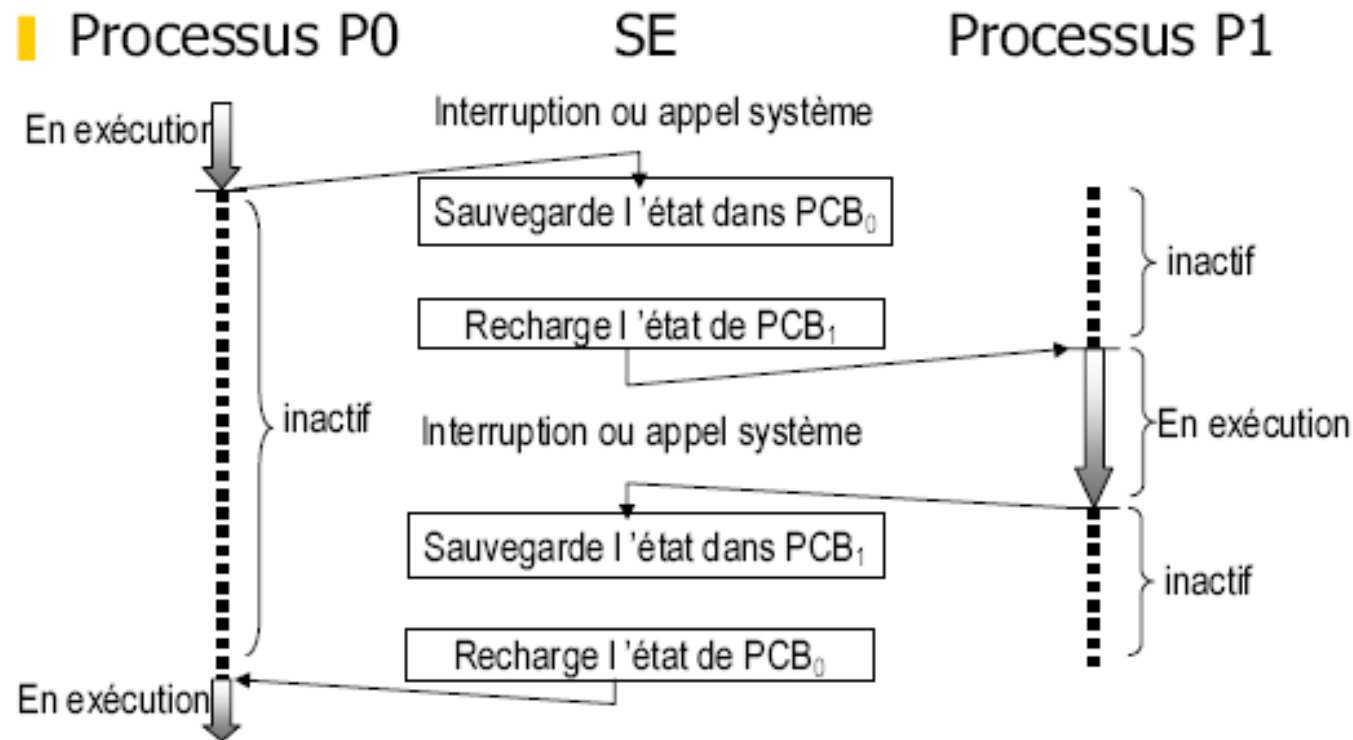
- en multiprogrammation, un processus détient l'UC de façon intermittente
- reprise cohérente
 - ⇒ sauvegarde des informations essentielles
 - PCB
 - l'état des données du programme : *image du programme* en mémoire principale ou secondaire (le PCB pointera à cette image)



PCB

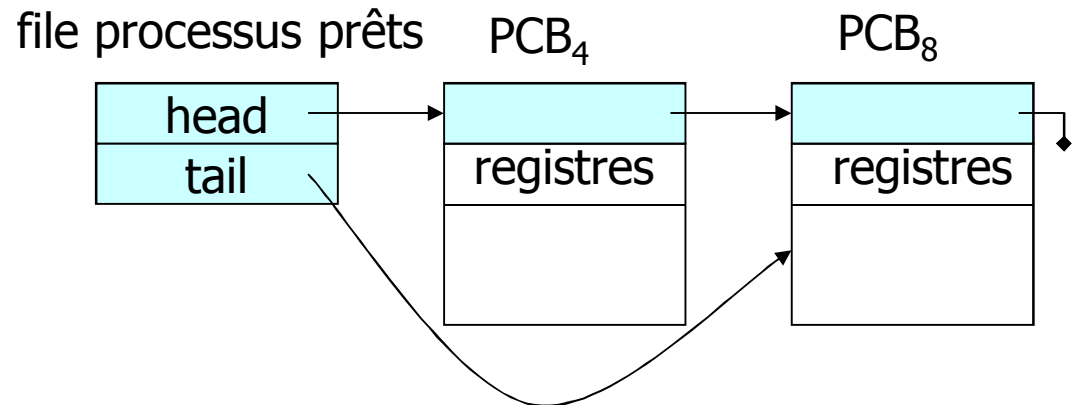
- Process Control Block (bloc de contrôle de processus)
 - état du processus
 - compteur d'instructions
 - registres CPU
 - information d'ordonnancement
 - information sur la gestion mémoire (table des pages/segments)
 - information de comptabilisation
 - état des E/S
- sert à sauvegarder et restaurer le contexte mémoire et processus lors d'une commutation de contexte

Commutation de l'UC entre processus (context switching)

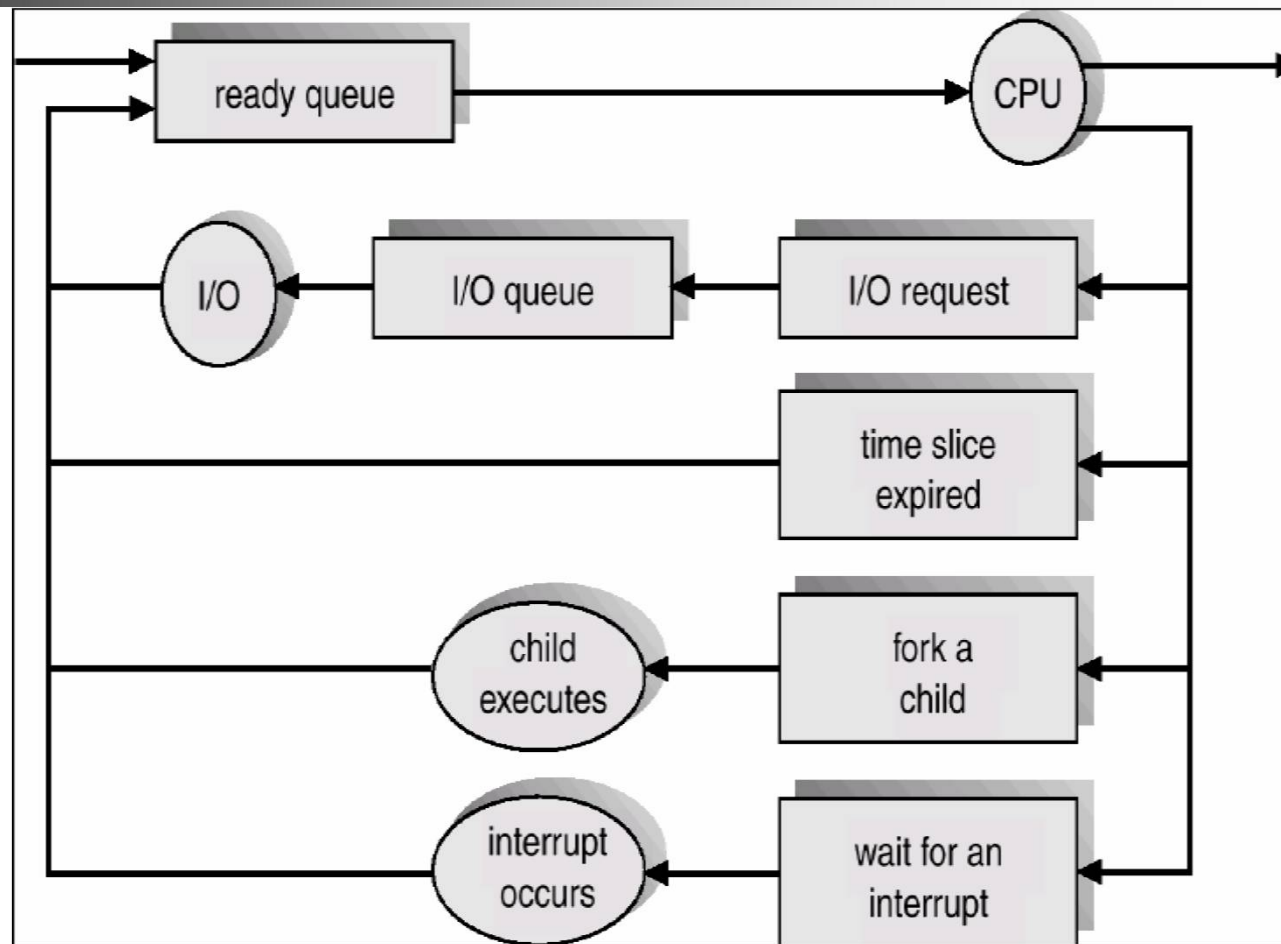


Ordonnancement des processus

- objectif : avoir à chaque moment un processus en exécution afin de maximiser l'utilisation de l'UC
- files d'attente
 - pointeurs vers les PCBs



Files d'attente (2)



ordonnancement de l'exécution
et allocation des ressources

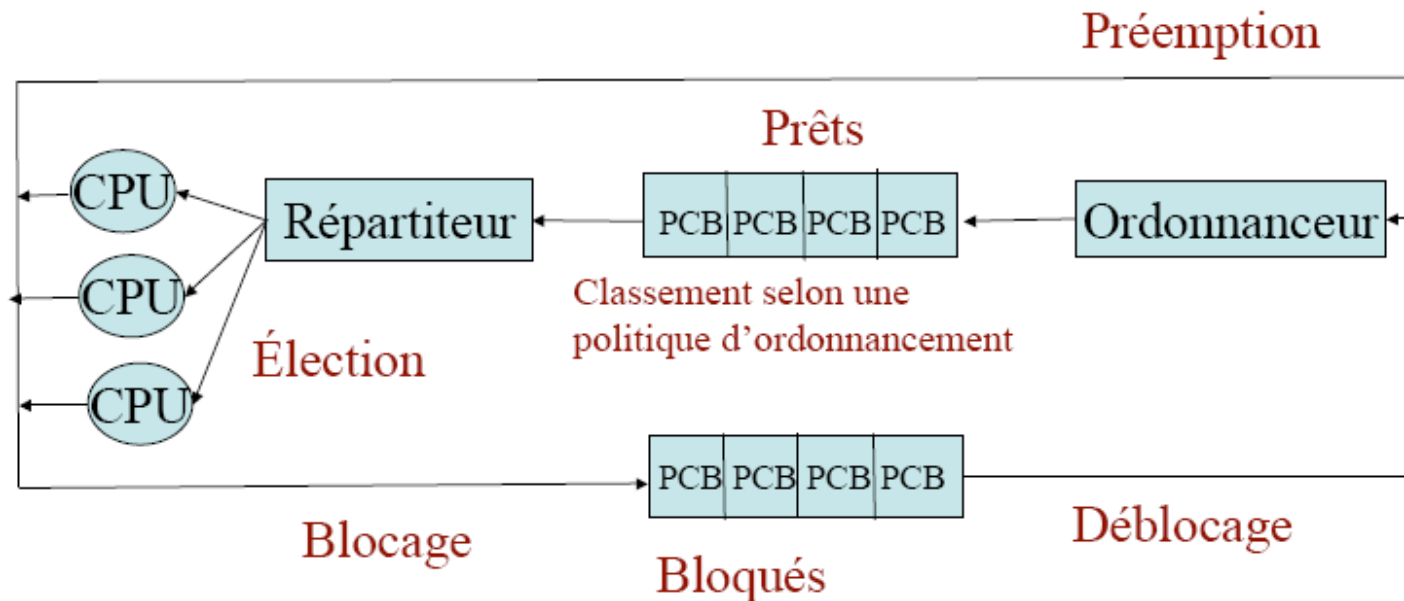
Silberschatz



Ordonnancement des processus

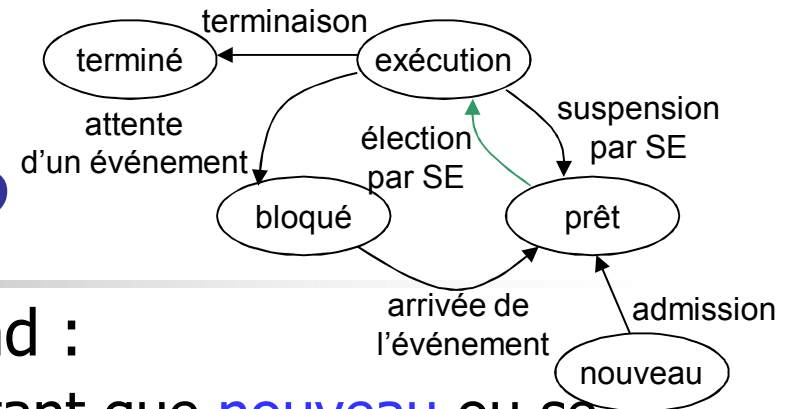
- objectif : avoir à chaque moment un processus en exécution afin de maximiser l'utilisation de l'UC
- files d'attente
 - pointeurs vers les PCBs
- solution : quand l'UC devient disponible, le SE choisit un processus parmi ceux de la file d'attente des processus prêts pour être exécuté ⇒ ***ordonnanceur de l'UC***

Ordonnanceur et répartiteur

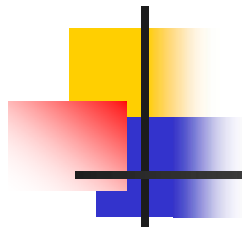


Delacroix

Quand invoquer l'ordonnanceur ?



- l'ordonnanceur intervient quand :
 - un processus se présente en tant que **nouveau** ou se **termine** ou
 - un processus **exécutant** devient **bloqué**
 - un processus change d'**exécutant** à **prêt**
 - un processus change de **bloqué** à **prêt**
- ⇒ tout événement dans un système cause une interruption de l'UC et l'intervention de l'ordonnanceur, qui devra prendre une décision concernant quel processus aura l'UC après
- **préemption** : si on enlève l'UC à un processus qui l'avait et peut continuer à s'en servir
 - dans les premiers deux cas, il n'y a **pas de préemption**
ordonnancement de l'exécution
et allocation des ressources



Critères d'ordonnancement

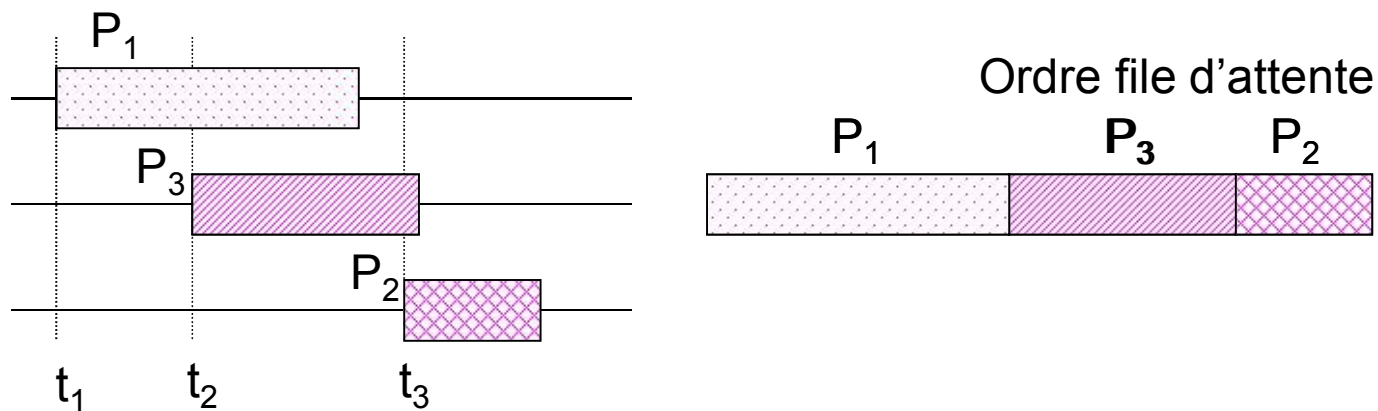
- situation :
 - plusieurs processus prêts (file *prêt*)
 - quand l'UC devient disponible lequel des processus choisir pour exécution ?

Critères

- l'*utilisation de l'unité centrale* : à maximiser
- le *débit (throughput)* : à maximiser
- le *délai de rotation (turnaround)* : à minimiser
- le *temps de réponse* : à minimiser
- le *temps d'attente* : à minimiser

Algorithmes d'ordonnancement (1)

- premier arrivé, premier servi (FCFS)



Ordre d'exécution

FCFS – exemple (1)

Exemple:

Processus

Temps de cycle

P_1

26

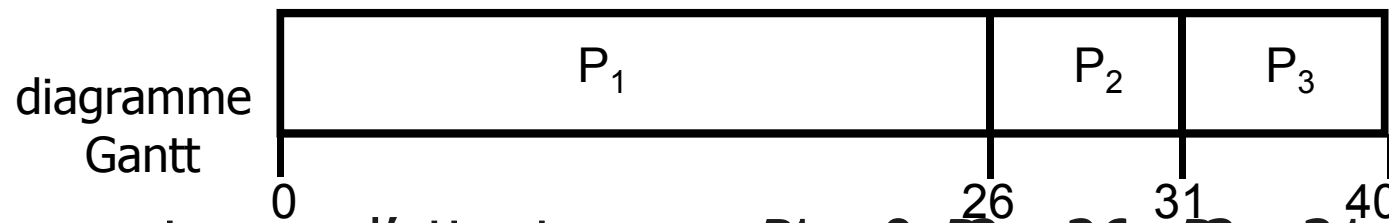
P_2

5

P_3

9

- les processus arrivent au temps 0 dans l'ordre P_1 , P_2 , P_3

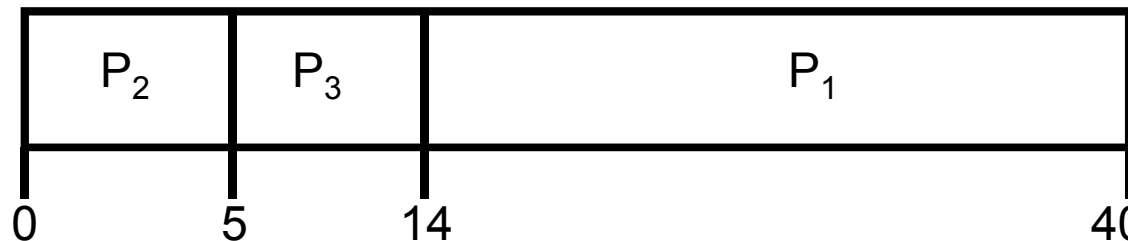


- temps d'attente pour $P_1=0$ $P_2=26$ $P_3=31$
- temps moyen d'attente : $(0 + 26 + 31)/3 = 19$
- utilisation UC : 100%
- débit : $3/40 = 0,075$
 - 3 processus complétés en 40 unités de temps
- temps moyen de rotation : $(26+31+40)/3 \cong 32,3$

ordonnancement de l'exécution
et allocation des ressources

FCFS – exemple (2)

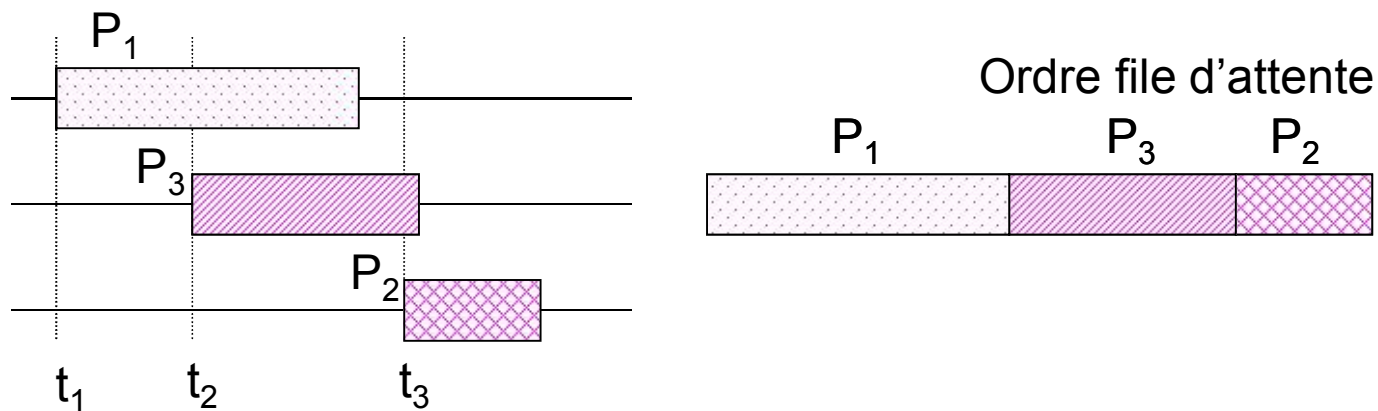
- les processus arrivent au temps 0 dans l'ordre P_2 , P_3 , P_1



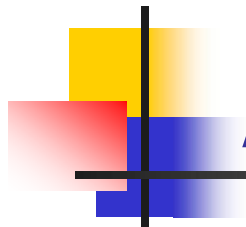
- temps d'attente pour $P_1 = 14$ $P_2 = 0$ $P_3 = 5$
- temps moyen d'attente : $(14 + 0 + 5)/3 \cong 6,3$
 - beaucoup mieux !
 - donc pour cette technique, le temps d'attente moyen peut varier grandement
- utilisation UC : 100%, débit : $3/40=0,075$
 - mêmes
- temps moyen de rotation : $(5+14+40)/3 \cong 19,6$

Algorithmes d'ordonnancement (2)

- travail le plus court d'abord (SJF)



Ordre d'exécution

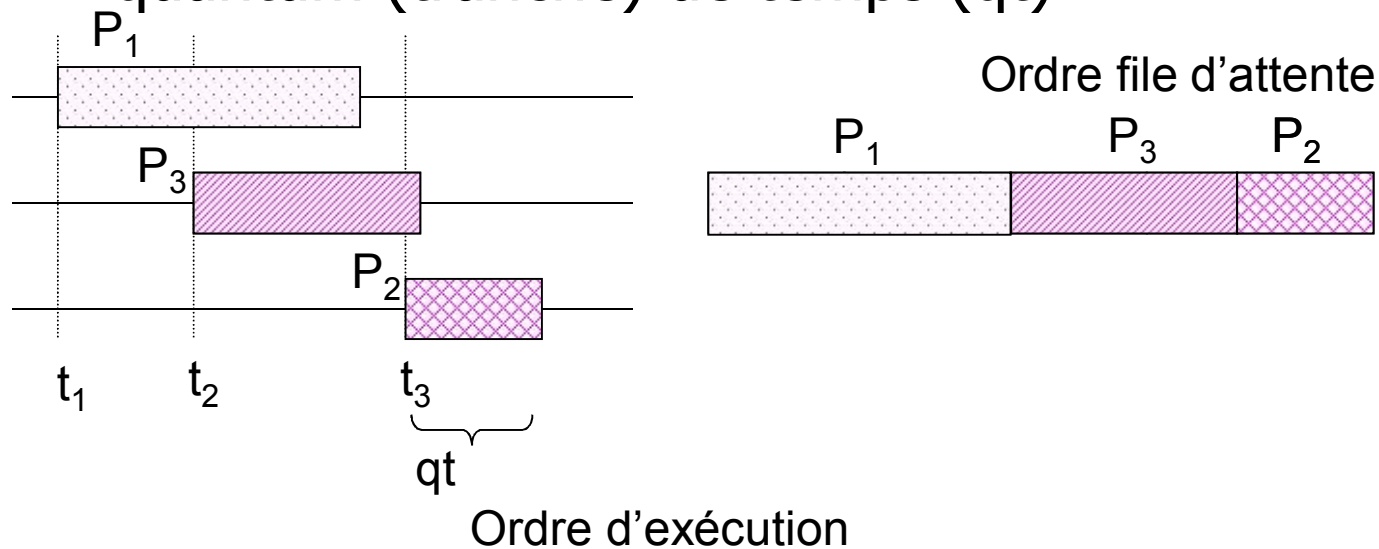


Algorithme SJF - critique

- **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 des environnements à temps partagé
 - un processus long peut monopoliser l'UC s'il est le 1er à entrer dans le système et il ne fait pas d'E/S

Algorithmes d'ordonnancement (3)

- ordonnancement à tourniquet (circulaire ou Round Robin)
 - quantum (tranche) de temps (qt)



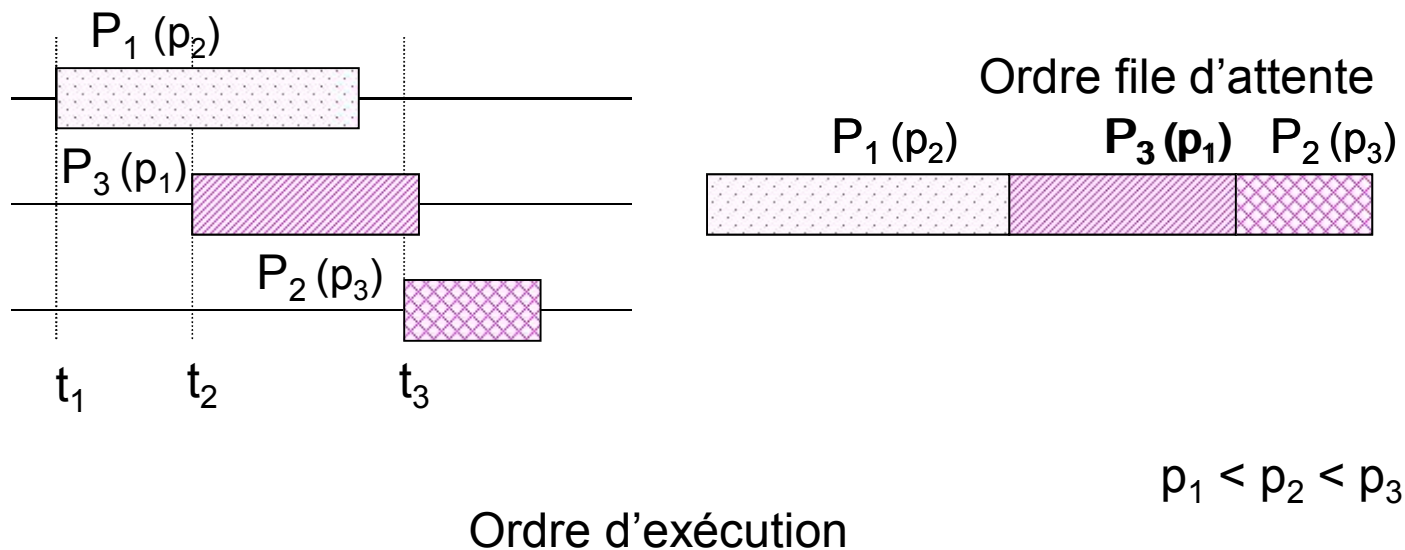


Priorité

- 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'UC est donnée au processus prêt avec la plus haute priorité
 - avec ou sans préemption
 - il y a une **file prêt** pour chaque priorité

Algorithmes d'ordonnancement (4)

- ordonnancement avec priorité



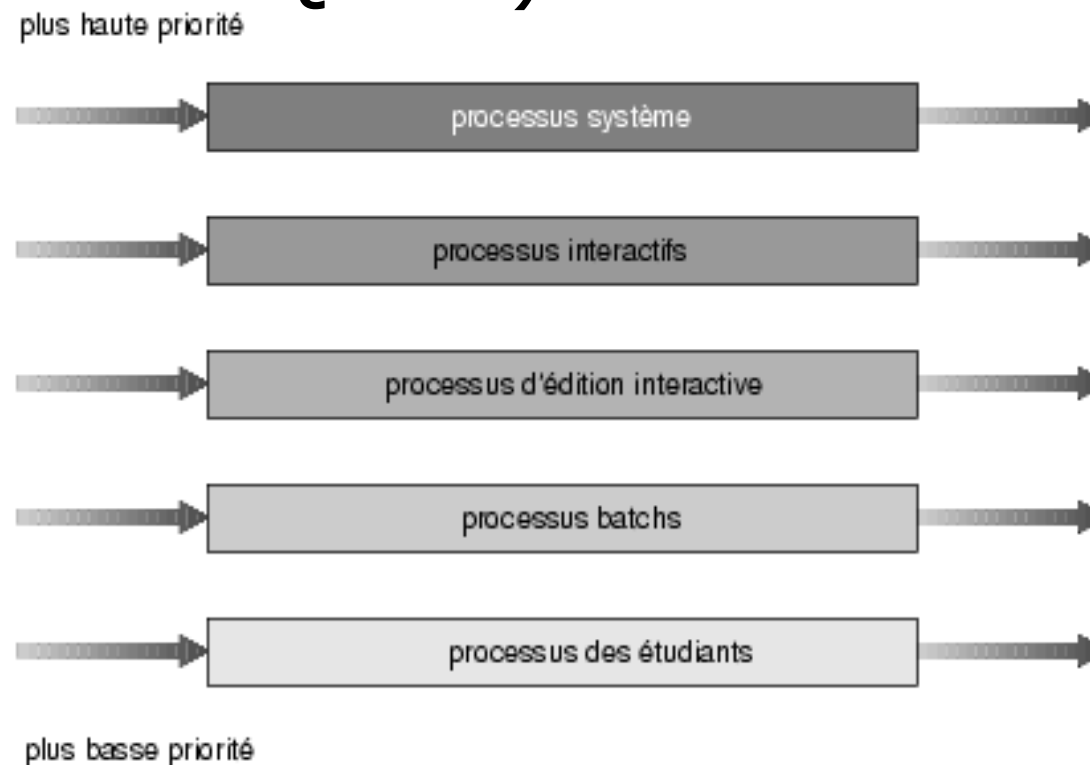


Problème possible avec les priorités

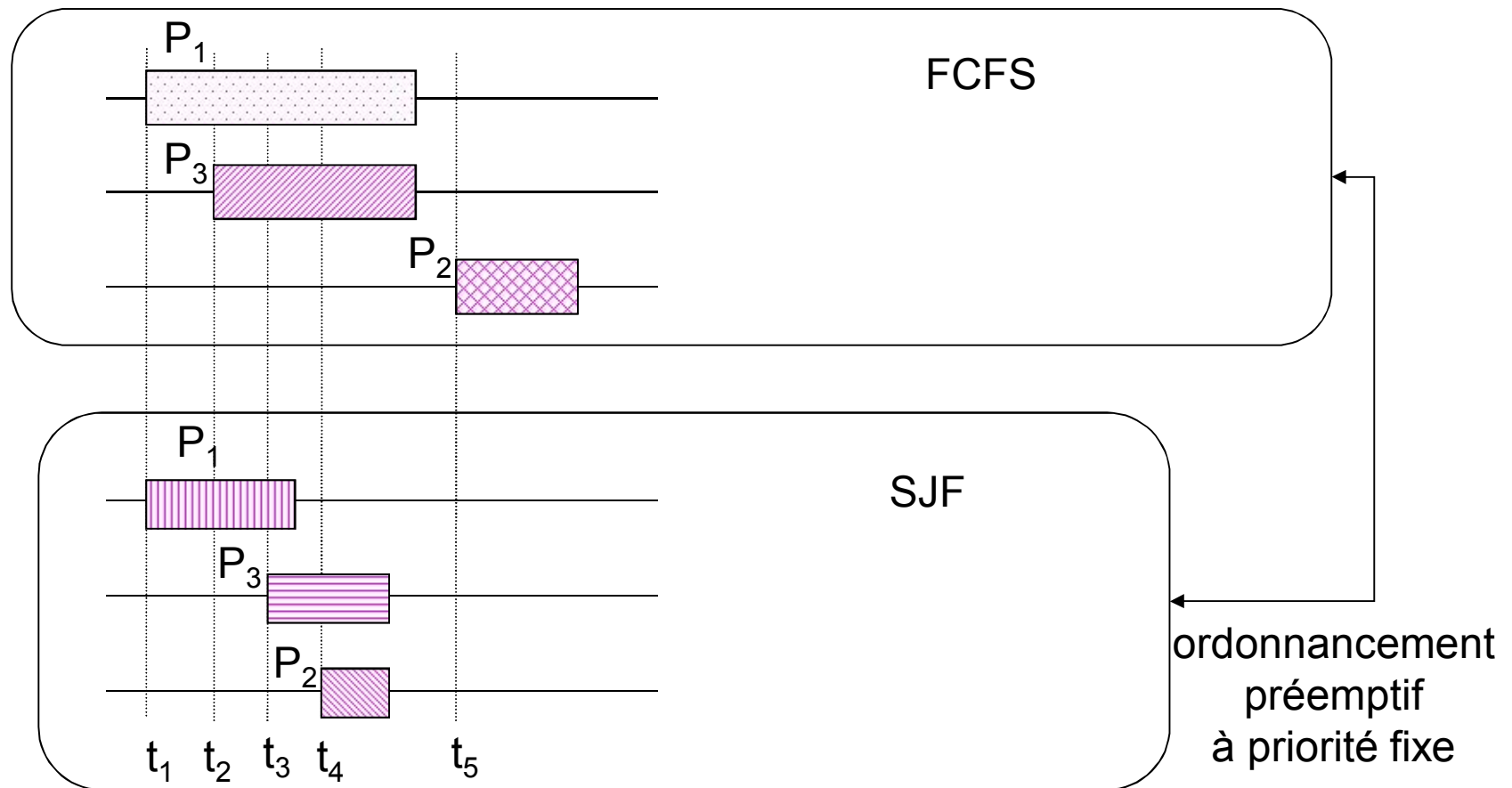
- ***famine*** : les processus moins prioritaires n'arrivent jamais à s'exécuter
- **Solutions** :
 - modifier la priorité d'un processus en fonction de son âge et de son historique d'exécution \Rightarrow ***vieillesse***
 - assignation dynamique des priorités
 - grouper les processus en catégories de priorités

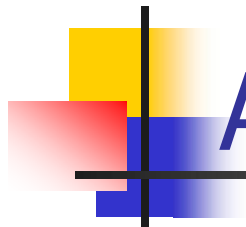
Algorithmes d'ordonnancement (5)

- ordonnancement à files multiniveaux (Multilevel Queue)



Ordonnancement à files multiniveaux





Algorithmes d'ordonnancement (6)

- ordonnancement avec files d'attente multiniveaux et à retour (Multilevel Feedback Queue)
 - ordonnancement avec files multiniveaux
 - les processus peuvent passer d'une file à une autre
 - déplacer un processus vers une file d'attente de priorité plus/moins élevée
 - empêche la famine
 - paramètres : le nombre de files, l'algorithme d'ordonnancement pour chaque file, méthode de déplacement d'un processus



En pratique

- les méthodes d'ordonnancement sont toutes utilisées en pratique (sauf plus court servi *pur* qui est impossible)
- les SE sophistiqués fournissent au gérant du système une bibliothèque de méthodes, qu'il peut choisir et combiner au besoin après avoir observé le comportement du système
- pour chaque méthode, plusieurs paramètres sont disponibles : p.ex. durée du quantum, coefficients, etc.



Gestion de processus

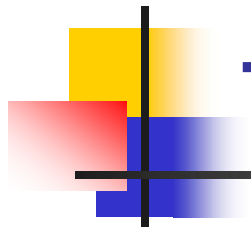
Allocation des ressources

Violeta Felea

violeta dot felea at femto-st dot fr

Département des Enseignements
Informatiques – UFR ST





Types de ressources

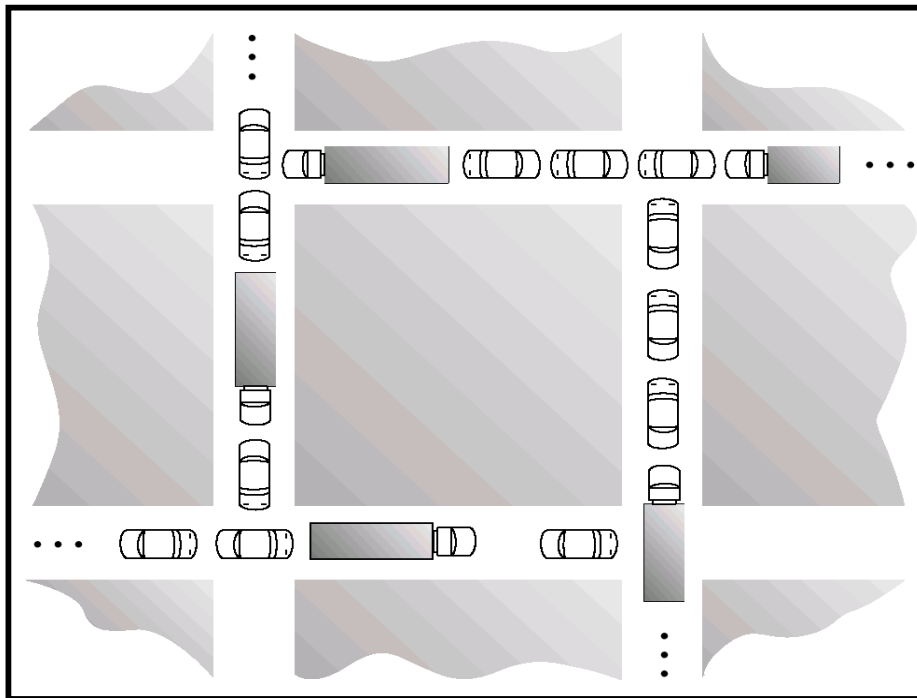
- ***ressource préemptible***
 - peut être retirée sans risque au processus qui la détient
 - exemple : la mémoire
- ***ressource non préemptible***
 - exemple : l'imprimante



Interblocage

- un ensemble de processus est en ***interblocage*** si chaque processus attend un événement que seul un autre processus de l'ensemble peut l'engendrer
 - la plupart du temps, événement = libération d'une ressource détenue par un autre processus

Exemples



Sémaphores

P_0

P_1

$P(A)$

$P(B)$

$P(B)$

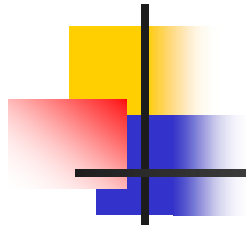
$P(A)$

A et B initialisés à 1



Interblocages – caractéristiques

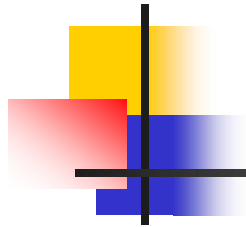
- Conditions nécessaires
 - exclusion mutuelle
 - détention et attente
 - pas de préemption
 - attente circulaire



Interblocages – solutions

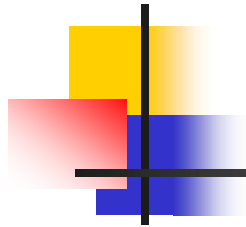
Stratégies pour traiter les interblocages

- les ignorer (exemple : Linux)
- détection et correction
- prévention
- empêchement



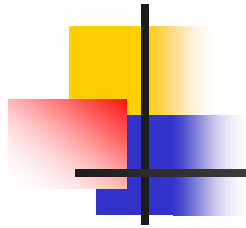
Détection des interblocages

- cas d'un seul type de ressources
 - modélisation sous la forme d'un graphe
 - des demandes d'allocation de ressources
 - des détentions de ressources
 - détection de cycle
- cas de plusieurs types de ressources
 - construire
 - matrice d'allocations courantes
 - matrice de demandes
 - matrice de ressources disponibles



Correction d'interblocages

- au moyen de la préemption des ressources
- au moyen de rollback (points de reprise - checkpoints)
- au moyen de suppression (terminaison) de processus



Prévention des interblocages

- exclusion mutuelle
- détention et attente
- pas de préemption
- attente circulaire



Elimination de la condition « occupation et attente »

- un processus demande toutes ses ressources en même temps
- le système lui alloue sur une base tout ou rien
- **Inconvénient** : beaucoup de processus ne connaissent pas à l'avance les ressources dont ils ont besoin



Elimination de la condition « pas de préemption »

- un processus qui attend pour une ressource doit libérer toutes les ressources qu'il détient
- **Inconvénients**
 - coûteux
 - famine
 - l'état de certaines ressources ne peut pas être sauvegardé



Elimination de la condition « attente circulaire »

- allocation des ressources dans un ordre bien défini
- $R = \{r_1, \dots, r_n\}$ l'ensemble des types de ressources
- fonction $f: R \rightarrow \mathbb{N}$
- **règle** : un processus qui détient une ressource r_i peut demander une ressource r_j ssi $f(r_i) < f(r_j)$ (ordonner les ressources)
= empêche l'apparition de cycles dans le graphe des ressources
- **Inconvénients**
 - nombres élevés
 - ajout de nouvelles ressources

Empêchement des interblocages

- **état sûr**

- séquence $\langle P_0, P_1, \dots, P_n \rangle$ sûre
 - si $\forall i$, les ressources que P_i pourrait encore demander peuvent être satisfaites par celles disponibles et celles des processus P_j , $j < i$

- Exemple : 12 ressources disponibles

cas 1 : $\langle P_1, P_0, P_2 \rangle$ séquence sûre ? cas 2 : $\langle P_1, P_0, P_2 \rangle$ séquence sûre ?

Processus	Besoin max	allouées
P_0	10	5
P_1	4	2
P_2	9	3

Processus	Besoin max	allouées
P_0	10	5
P_1	4	2
P_2	9	2

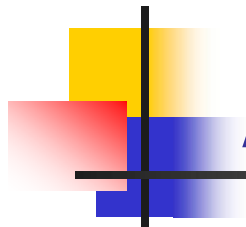
ordonnancement de l'exécution
et allocation des ressources



Empêchement

- algorithme permettant une allocation ssi celle-ci laisse le système dans un état sûr
 - un interblocage est un état non sûr
 - un état non sûr n'implique pas un interblocage
 - un processus demandant une ressource peut attendre même si elle est disponible
- algorithme du banquier (Dijkstra, 1965)

ordonnancement de l'exécution
et allocation des ressources



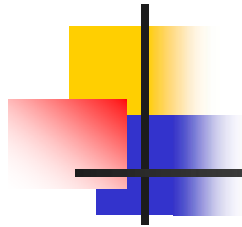
Algorithme du banquier (1)

- Supposition : un seul type de ressources
- Soit
 - dem_i le vecteur des demandes de P_i
 - $disponible$ le nombre de ressources disponibles
 - $allouées_i$ le nombre de ressources allouées à P_i
 - max_i le maximum de ressources requises par P_i
 - $besoin_i$ les besoins restants de P_i ($max_i - allouées_i$)



Algorithme du banquier (2)

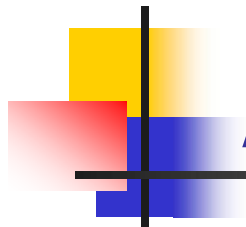
- lors d'une demande d'allocation du processus P_i
 - si $dem_i > besoin_i$ alors erreur
 - si $dem_i > disponible$ alors attente
 - le système simule l'allocation
 - $disponible = disponible - dem_i$
 - $allouées_i = allouées_i + dem_i$
 - $besoin_i = besoin_i - dem_i$
 - si l'état résultant est sûr alors l'allocation est effectuée



Algorithme du banquier (3)

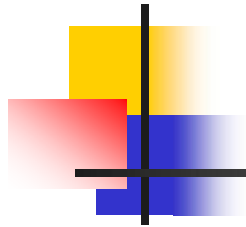
Algorithme pour vérifier qu'un état est sûr

- $travail = disponible$ et $\forall P_i : fini[i] = faux$
- pour $\forall i$, tel que $fini[i] = faux$ et
$$besoin_i \leq travail$$
 - $travail = travail + allouées_i$
 - $fini[i] = vrai$
- si $\forall i, fini[i] = vrai$ alors l'état est sûr



Algorithme du banquier (4)

- Inconvénients
 - nombre fixe de ressources
 - nombre fixe de demandeurs
 - besoin d'informations supplémentaires (par ex. maximum de ressources nécessaires)



Bilan du cours

- Ordonnancement de l'exécution
 - algorithmes (FCFS, SJF, RR, priorité)
- Allocation des ressources
 - interblocage
 - stratégies de traitement
 - les ignorer
 - détection et correction
 - prévention
 - empêchement