

Gestion des processus dans un OS

M1105 - Systèmes d'exploitation

Sebastien.Kramm@univ-rouen.fr

IUT de Rouen, dépt. Réseaux & Télécoms

Version du 15 octobre 2018



Introduction

- ▶ Un OS exécute des "programmes" : terme ambigu
- ▶ Définition : Programme = fichier exécutable binaire, stocké sur unité de stockage (HDD).
Son exécution va provoquer la création d'un **processus**.
- ▶ Multiplicité :
 - ▶ un programme en cours d'exécution peut être implémenté via plusieurs processus qui communiquent entre eux.
 - ▶ Le même programme peut-être plusieurs fois en mémoire : plusieurs processus peuvent exécuter le même programme

Définition

- ▶ processus = programme en cours d'exécution
- ▶ processus : activité qui possède :
 - ▶ un programme ("recette"),
 - ▶ des données ("ingrédients"),
 - ▶ un état courant (un point précis dans la recette)



Application et services ?

Application

- ▶ Une **application** est un programme utilisateur (traitement de texte, jeux, navigateur, etc.), généralement associé au bureau (desktop).
- ▶ Peut-avoir un seul processus associé ou plusieurs, qui peuvent communiquer entre eux, via des mécanismes d'IPC : *Inter Process Communication*.

Service

- ▶ Un **service** est un processus qui s'exécute en arrière plan et n'a pas d'interactions avec l'utilisateur.
- ▶ Géré par l'OS (démarrage, arrêt), correspond à des "services" dont les applications ont besoin.
- ▶ Windows : les services sont en général exécutés via des instances du processus `svchost.exe` (*Service Host*).
- ▶ Linux : on appelle les services des *daemons*, sont souvent identifiés comme tels par la lettre "d" en fin de leur nom (exemples : `crond`, `lighttpd`, ...) (voir liste)



Windows

- ▶ Le programme "Explorateur de tâches" (`taskmgr.exe`) montre applications et services, ainsi que les processus associés.
- ▶ Accès : CTRL-ALT-SUPPR (ou taper son nom dans un shell).

Nom de l'image	Nom d'u...	P...	Mémoire ...	Description
ccvchost.exe *32	krammseb	00	1 820 K	Symantec Service Fra...
csrss.exe		00	1 592 K	
dwm.exe	krammseb	00	1 288 K	Gestionnaire de fenêtr...
explorer.exe	krammseb	00	11 560 K	Explorateur Windows
java.exe *32	krammseb	00	77 008 K	Java(TM) Platform SE ...
rdpclip.exe	krammseb	00	1 476 K	Analysateur de clip RDP
taskhost.exe	krammseb	00	1 924 K	Processus hôte pour T...
taskmgr.exe	krammseb	00	2 076 K	Gestionnaire des tâche...
winlogon.exe		00	1 668 K	

Nom	PID	Description	Statut	Groupe
AeLookupSvc	920	Expérience d'application	En co...	netsvcs
ALG		Service de la passerell...	Arrêté	N/D
AppIDSvc		Identité de l'application	Arrêté	LocalServiceAndN...
AppInfo	920	Informations d'applica...	En co...	netsvcs
AppMgmt		Gestion d'applications	Arrêté	netsvcs
aspnet_state		Service d'état ASP.NET	Arrêté	N/D
AudioEndpointBuilder		Générateur de points ...	Arrêté	LocalSystemNetwo...
AudioSrv		Audio Windows	En co...	LocalServiceNetwo...
BFE	800	Moteur de filtrage de ...	Arrêté	LocalServiceNoNet...
BITS		Service de transfert l...	Arrêté	netsvcs
Browser		Explorateur d'ordinat...	Arrêté	netsvcs
CertPropSvc	920	Propagation du certificat	En co...	netsvcs

- ▶ Il existe des outils tiers plus évolués, montrant la généalogie des processus. (voir SysInternals ProcessExplorer)



- ▶ Outil GUI : selon la distribution
- ▶ Vue statique à un instant donné (*snapshot*) : commande `ps`
- ▶ vue dynamique : commande `top` et `htop` (version améliorée) :

```
File Edit View Search Terminal Help
1 [ | 1.3% ] 5 [ |
2 [ | 0.7% ] 6 [ |
3 [ | 0.0% ] 7 [ |
4 [ | 2.6% ] 8 [ |
Mem[|||||]2309/7890MB Tasks: 132, 267 thr; 1 running
Swp[ ] 0/0MB Load average: 0.06 0.10 0.10
Uptime: 3 days, 09:26:30

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
24213 sk 20 0 2100M 652M 95540 S 3.3 8.3 1h08:42 /usr/lib/firefox/firefox
31254 sk 20 0 30624 2780 1448 R 0.7 0.0 0:00.53 htop
1745 root 20 0 1030M 361M 318M S 0.7 4.6 55:31.03 /usr/bin/X -core :0 -seat se
30498 sk 20 0 676M 30616 24864 S 0.7 0.5 0:02.81 gnome-system-monitor
2494 sk 20 0 646M 25604 13984 S 0.7 0.3 0:24.49 /usr/lib/gnome-terminal/gnom
24231 sk 20 0 2100M 652M 95540 S 0.7 8.3 0:12.76 /usr/lib/firefox/firefox
2386 sk 20 0 483M 30516 13912 S 0.0 0.5 32:01.78 compiz
24239 sk 20 0 2100M 652M 95540 S 0.0 8.3 0:45.00 /usr/lib/firefox/firefox
24229 sk 20 0 2100M 652M 95540 S 0.0 8.3 0:12.65 /usr/lib/firefox/firefox
2672 sk 20 0 617M 15284 11132 S 0.0 0.2 2:02.96 /usr/lib/gnome-applets/multt
24267 sk 20 0 2100M 652M 95540 S 0.0 8.3 0:19.55 /usr/lib/firefox/firefox
24286 sk 20 0 2100M 652M 95540 S 0.0 8.3 0:23.69 /usr/lib/firefox/firefox
2396 sk 20 0 806M 80256 20712 S 0.0 1.1 2:31.27 gnome-panel
2252 sk 20 0 362M 18576 2892 S 0.0 0.2 2:51.22 /usr/bin/ibus-daemon --daemon
```

- ▶ Un processus est caractérisé par
 - ▶ identifiant : valeur numérique entière unique : PID
 - ▶ état : en cours d'exécution, prêt (en attente d'exécution), bloqué (en attente d'I/O)
 - ▶ répertoire de travail
 - ▶ adresses et taille du bloc de mémoire alloué par l'OS
 - ▶ priorité d'exécution
 - ▶ liste des fichiers ouverts
 - ▶ ...
- ▶ Ces informations sont mémorisées par l'OS dans un *Process control block* (PCB)

Pointer	Process state
	Process number
	Program counter
	Registers
	Memory limits
	List of open files
	...

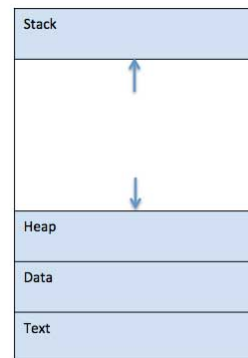
Différences entre systèmes

- ▶ Linux : Tous les processus ont un parent, définissant une **arborescence** de processus depuis le processus initial (*init*), lancé au boot.
- ▶ Windows : il est possible de créer un processus sans parent.

Allocation mémoire

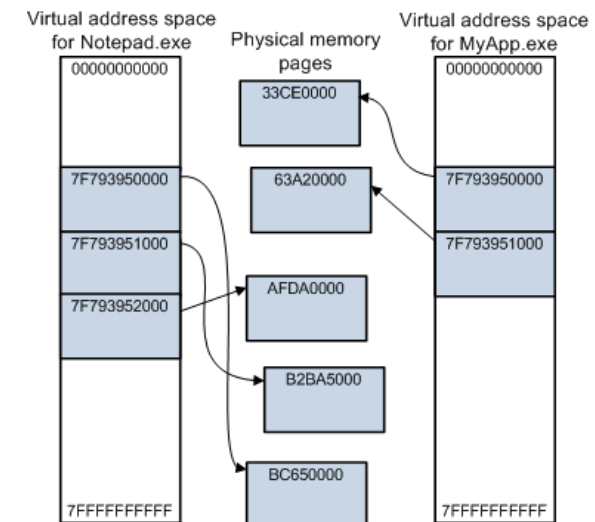
- ▶ Le chargement d'un exécutable en mémoire consiste à allouer un segment de mémoire pour chaque section :

- ▶ `text` : segment de code, contient les instructions du programme à exécuter.
- ▶ `data` : segment de données : contient les variables globales ou statiques du programme.
- ▶ `heap` ("tas") : segment de données : contient les variables obtenues via une allocation dynamique de mémoire.
- ▶ `stack` : segment de pile : stockage des appels de fonctions avec les paramètres et les variables locales.



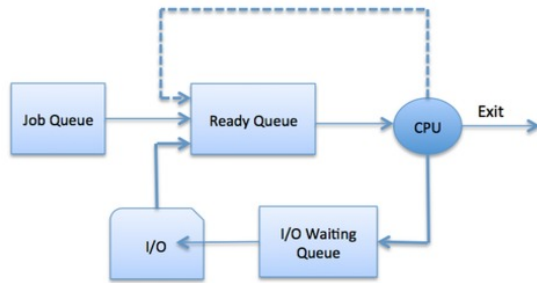
Adresses virtuelles et physiques

- ▶ Ces différents segments sont **virtuels** : un mécanisme de pagination donne l'adresse réelle.

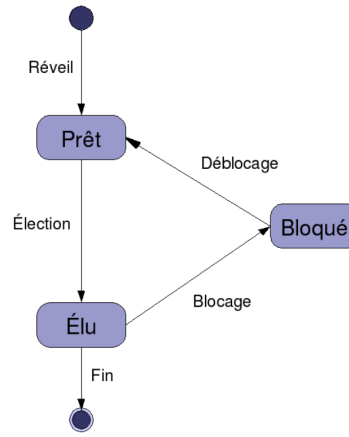


États d'un processus

- ▶ Un processus est toujours dans l'un des ces 3 états :
 1. en cours d'exécution,
 2. prêt et en pause (le CPU exécute un autre),
 3. bloqué : en attente d'un événement externe :
Attente du résultat d'un appel système (opération d'E/S)
- ▶ L'OS maintient des listes dynamiques des processus dans les cas 2 et 3.



source : tutorialspoint.com

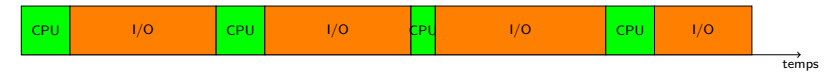


source : F. Letombe

Types de processus

- ▶ Tous les processus ne se comportent pas de la même manière (ratio calcul / attente I/O)

- ▶ Programme GUI interactif
→ faible besoins CPU mais d'attente des I/O



- ▶ Programme de calcul intensif (Ex : encodage de vidéo)
→ faibles besoins I/O mais beaucoup de temps CPU



- ▶ L'OS peut en déduire la durée prévisible des "burst" CPU, qu'on assimilera au temps nécessaire pour terminer le processus.

Communication avec les processus

- ▶ Différents moyens permettent d'interagir avec les processus (voir *Inter-Process Communication*).
- ▶ Le moyen le plus simple : envoi d'un **signal** par l'OS au processus.
- ▶ Le programme associé au processus peut :
 - ▶ avoir prévu la réception de ce signal, et entamer une action spécifique lors de sa réception.
 - ▶ ne **pas** avoir prévu la réception de ce signal : le processus se termine.
- ⇒ technique utilisée par l'OS lorsque l'utilisateur veut terminer un processus récalcitrant.
- ▶ Linux : envoi du signal SIGTERM (*Terminate*) depuis le shell :
 - ▶ kill <PID>
 - ▶ killall <nom-process>
- ▶ POSIX définit une trentaine de signaux différents.

Ordonnancement de tâches

Définition

Un ordonnanceur de tâches est un composant logiciel du noyau qui attribue du temps-CPU à chaque processus dans l'état "prêt", de façon à ce qu'il puisse s'exécuter en totalité.

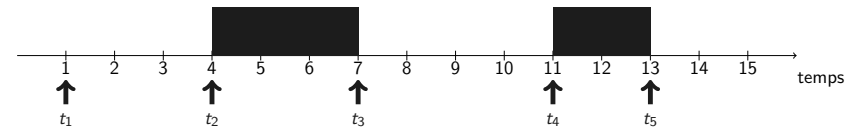
- ▶ Objectifs :
 - ▶ Maximiser l'utilisation du processeur
 - ▶ Présenter un temps de réponse acceptable
 - ▶ Respecter l'équité entre les processus selon le critère d'ordonnancement utilisé.
- ▶ Deux types d'algorithmes : Prémptifs & Non-prémptifs
- ▶ Problématiques :
 - ▶ Commutation de tâches coûteuse
 - ▶ Choix du quantum de temps

Critères de performance

- ▶ Pour comparer la performance des différents algorithmes, on a besoin de critères, qui seront soit à minimiser soit à maximiser.
- ▶ On définit les indicateurs suivants, qu'on cherchera à minimiser :
 - ▶ *Waiting time* (Temps d'attente) t_w : durée qu'un processus passe à attendre.
 - ▶ *Response time* (Temps de réponse) t_r : temps qu'il faut au système pour **commencer** à traiter le processus.
 - ▶ *Turnaround time* (Temps de rotation) t_t : temps terminaison moins temps arrivée.
- ▶ Pour caractériser la performance d'un algorithme, on calculera la moyenne de ces valeurs pour plusieurs processus.
- ▶ Pertinence : le temps de rotation est lié à la durée d'exécution intrinsèque du processus → critère peu objectif.

Critères de performance : exemple

- ▶ Exemple : soit un processus qui arrive à un instant t_1 , dont l'exécution commence à un instant t_2 , qui est interrompu à un instant t_3 . L'exécution reprend ensuite à un instant t_4 pour se terminer à l'instant t_5 .
- ▶ On peut représenter son exécution par le graphique suivant (unité de temps arbitraire) :



- ▶ Résultats pour ce processus :
 - ▶ *Response time* $t_r = t_2 - t_1 = 3$
 - ▶ *Waiting time* $t_w = t_r + (t_4 - t_3) = 3 + (11 - 7) = 7$
 - ▶ *Turnaround time* $t_t = t_5 - t_1 = 13 - 1 = 12$

Algorithmes

On peut classifier les algorithmes d'ordonnement :

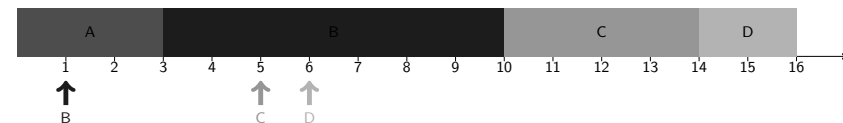
- ▶ Non-préemptifs : un processus en exécution continue jusqu'à ce qu'il se termine ou qu'il passe à l'état "bloqué" (attente d'I/O)
 - ▶ FCFS : *First Come, Firsts Serve* (premier arrivé, premier servi)
 - ▶ FJS : *Shortest-Job-First* (priorité au plus court)
- ▶ Préemptifs : un processus en exécution peut être interrompu par l'OS pour en exécuter un autre
 - ▶ SRT : *Shortest Remaining Time*
 - ▶ RR : *Round Robin* (Tourniquet)
- ▶ Algorithmes à priorité : on catégorise les processus par un niveau de priorité, déterminé en fonction de plusieurs critères.

First Come, Firsts Serve (FCFS)

- ▶ Les processus qui sont prêts sont exécutés dans l'ordre de leur arrivée
- ▶ Exemple : soit les 4 processus suivants :

Processus	Date d'arrivée	Temps de traitement
A	0	3
B	1	7
C	5	4
D	6	2

- ▶ Exécution :



- ▶ Performance :
 - ▶ Réponse et attente : $t_w = t_r = A : 0 + B : 2 + C : 5 + D : 8 = 15$
Moyenne sur 4 processus = 3,75
 - ▶ Turnaround : $t_t = A : (3-0) + B : (10-1) + C : (14-5) + D : (16-6) = 3 + 9 + 9 + 10 = 31$
Moyenne sur 4 processus = 7,75

Shortest-Job-First (SJF)

- ▶ Implique une connaissance a priori sur la durée d'exécution.
- ▶ Principe : au moment de la sélection du prochain processus, on choisit celui avec la durée d'exécution la plus faible.
- ▶ Exemple : avec les mêmes processus que précédemment :

Processus	Date d'arrivée	Temps de traitement
A	0	3
B	1	7
C	5	4
D	6	2

- ▶ Exécution :



- ▶ Performance :

- ▶ Réponse et attente : $t_w = t_r = A : 0 + B : 2 + C : 7 + D : 4 = 13$
Moyenne sur 4 processus = 3,24
- ▶ Turnaround : $t_t = A : (3-0) + B : (10-1) + C : (16-5) + D : (12-6)$
 $= 3 + 9 + 11 + 4 = 27$
Moyenne sur 4 processus = 6,75

Shortest Remaining Time (SRT) - 1

- ▶ Version préemptive du Shortest-Job-First

Principe

Chaque fois qu'un nouveau processus est introduit dans la file, on compare sa durée à la durée restante du processus en cours d'exécution.

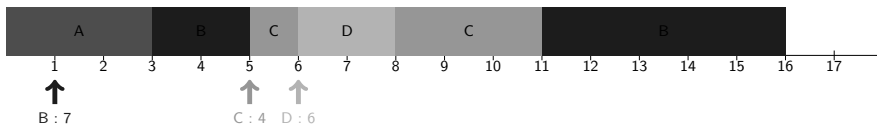
- ▶ si le temps de traitement du nouveau processus est inférieur, le processus en cours d'exécution est arrêté ;
- ▶ sinon, on continue avec le processus en cours.
- ▶ Favorise les processus courts, les processus longs peuvent être victimes de famine.

Shortest Remaining Time (SRT) - 2

- ▶ Exemple, avec les mêmes processus que précédemment :

Processus	Date d'arrivée	Temps de traitement
A	0	3
B	1	7
C	5	4
D	6	2

- ▶ Exécution :

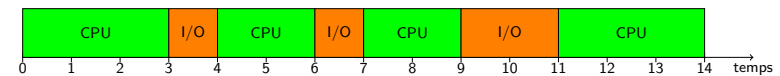


- ▶ Performance :

- ▶ Réponse : $t_r = A : 0 + B : 2 + C : 0 + D : 0 = 2$
Moyenne sur 4 processus = 0,5
- ▶ Attente : $t_w = A : 0 + B : (2+6) + C : 2 + D : 0 = 10$
Moyenne sur 4 processus = 2,5
- ▶ Turnaround : $t_t = A : (3-0) + B : (16-1) + C : (11-5) + D : (8-6)$
 $= 3 + 15 + 6 + 2 = 26$
Moyenne sur 4 processus = 6,5

Problème

- ▶ Problème des deux algorithmes précédents : ils impliquent une connaissance de la durée d'exécution des processus... qu'on a en général pas !
- ▶ Solution : estimation statistique des durées des "burst" CPU à partir de la moyenne des exécutions précédentes.
- ▶ Exemple : on observe le comportement d'un process :



⇒ Le prochain burst CPU peut être estimé à $(3+2+2+3)/4 = 2,5$

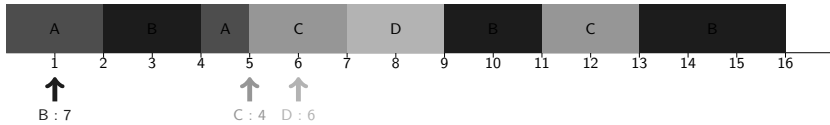
Round Robin (RR) - 1

Principe

- ▶ chaque processus se voit attribuer un quantum de temps CPU ;
- ▶ un *Timer* provoque une interruption à intervalles régulier, et on bascule d'un processus à un autre ;
- ▶ un processus prêt arrivant est placé en fin de file.

▶ Version de base : sans priorités

▶ Exemple avec un quantum de 2 unités de temps :

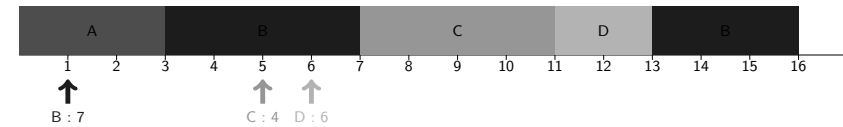


▶ Performance :

- ▶ Réponse : $t_r = A : 0 + B : 1 + C : 0 + D : 1 = 2$
Moyenne sur 4 processus = 0,5
- ▶ Attente : $t_w = A : 2 + B : 8 + C : 4 + D : 1 = 15$
Moyenne sur 4 processus = 3,75
- ▶ Turnaround : $t_t = A : (5-0) + B : (16-1) + C : (13-5) + D : (9-6) = 5 + 15 + 8 + 3 = 31$
Moyenne sur 4 processus = 7,75

Round Robin (RR) - 2

▶ Exemple avec un quantum de 4 :



▶ Performance :

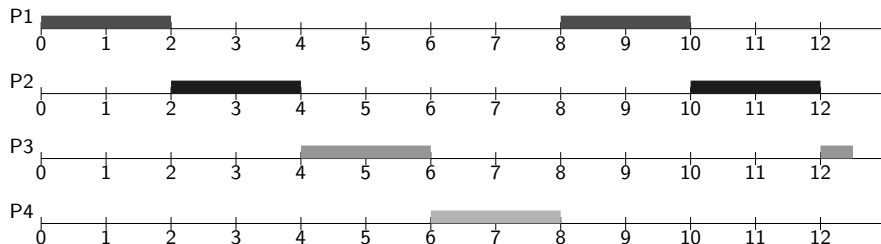
- ▶ Réponse : $t_r = A : 0 + B : 2 + C : 2 + D : 5 = 9$
Moyenne sur 4 processus = 2,25
- ▶ Attente : $t_w = A : 0 + B : 8 + C : 2 + D : 5 = 15$
Moyenne sur 4 processus = 3,75
- ▶ Turnaround : $t_t = A : (3-0) + B : (16-1) + C : (11-5) + D : (13-6) = 3 + 15 + 6 + 7 = 26$
Moyenne sur 4 processus = 6,5

Round Robin : cas générique

Avec n processus

- ▶ Chacun d'eux se voit attribuer $1/n$ du temps CPU
- ▶ Chacun d'eux attend $(n-1)/n$ % du temps
- ▶ Le temps de réponse est plus rapide

▶ Exemple, avec $n = 4$ et $q = 2$:



Comparaison des performances

	Algo	Réponse t_r	Attente t_w	Turnaround t_t
Non-préemptifs	FCFS	3,8	3,8	7,8
	SJF	3,2	3,2	6,8
Préemptifs	SRT	0,5	2,5	6,5
	RR (q=2)	0,5	3,8	7,8
	RR (q=4)	2,3	3,8	6,5

- ▶ Cette analyse n'est valable que pour cet exemple précis.
- ▶ Pour généraliser, il faut utiliser des techniques mathématiques probabilistes (Théorie des "files d'attentes").
- ▶ Pour le *Round Robin*, a priori, il semble qu'un quantum faible améliore les performances, **sauf que...**

Commutation de contexte

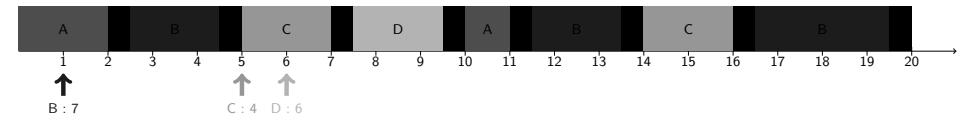
- ▶ Lors d'un changement de contexte d'un processus p1 à un processus p2, il faut
 1. sauvegarder les valeurs des registres CPU de p1
 2. charger dans les registres CPU les valeurs de p2
 3. basculer (éventuellement) le CPU du mode protégé au mode utilisateur
- ▶ Il faudra également sauvegarder et restaurer l'état des ressources utilisées par le processus (fichiers ouverts, connexion réseaux, connexions à des E/S périphériques, etc.)

A retenir

Ces opérations sont couteuses en temps, et génèrent une perte de rendement du temps CPU.

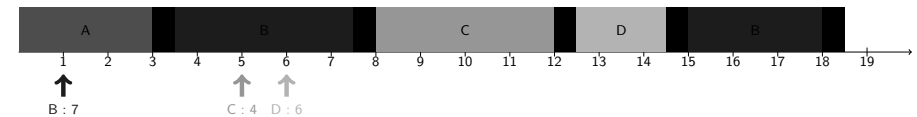
Commutation de contexte : exemple

- ▶ Hypothèse : la commutation de contexte a une durée de 0.5
- ▶ Exemple avec un quantum de 2 :



⇒ Rendement = temps total / temps utile = 16 / 20 = 80 %

- ▶ En augmentant le quantum à 4 :



⇒ Rendement = temps total / temps utile = 16/18,5 = 86 %

Règle générale pour le choix du quantum

- ▶ trop petit : baisse du rendement, trop de temps passé dans la commutation de contexte
- ▶ trop élevé : baisse de la réactivité du système (augmentation du temps d'attente)

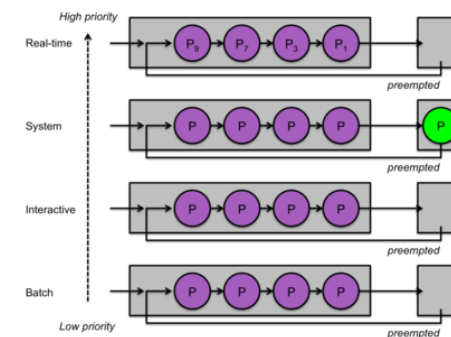
Algorithmes à priorités : principes

- ▶ Chaque processus se voit assigné un niveau de priorité
- ▶ L'ordonnanceur va distribuer le temps CPU en priorité aux processus ayant la priorité la plus élevée
- ▶ La priorité peut-être fixée de façon :
 - ▶ **interne** : en fonction de paramètres internes : historique du processus, mémoire utilisée, etc.
 - ▶ **externe** : fixée par l'utilisateur et/ou l'administrateur
- ▶ La priorité peut-être :
 - ▶ **statique** : ne change pas pendant toute la durée de vie du processus
Mais problème : les tâches les moins prioritaires risquent d'être "en famine" (=jamais exécutées).
 - ▶ **dynamique** : évolue dans le temps.
⇒ plus est processus est "ancien" dans la file d'attente, plus son niveau de priorité augmente.

Algorithmes évolués : files multiples

Dans un OS moderne, il n'y a pas qu'une seule file d'attente : on classe les processus selon un degré de priorité dans différentes files qui sont traitées en parallèle, chacune utilisant l'algorithme le plus adapté.

- ▶ Linux/Unix : 0 = priorité la **plus** élevée
- ▶ Windows : 0 = priorité la **moins** élevée



source : Paul Krzyzanowski