I22 — Architecture des ordinateurs et système d'exploitation

J. Razik

Dpt. Informatique — UTLN

2020–2021

Les systèmes d'exploitation

Gestion des utilisateurs

Types de système :

- Mono-utilisateur : un seul utilisateur peut utiliser le système en même temps,
- Multi-utilisateurs : plusieurs utilisateurs peuvent utiliser le système en même temps.

Un système d'exploitation multi-utilisateur doit proposer les services suivants :

- Service de connexion/déconnexion
  - Vérification de validation de l'accès (login/passwd)
  - Initialisation de l'environnement et des ressources des utilisateurs
- Service d'administration des utilisateurs
  - Pour les utilisateurs : gestion de leurs ressources, modification de paramètres (passwd, . . .)
  - Pour l'administrateur : gestion globale des ressources, ajout d'utilisateurs, politique d'allocation des ressources, sauvegarde, sécurité, etc.

Gestion des fichiers

Le système d'exploitation doit donc intégrer un système de gestion des fichiers qui va gérer :

- L'espace physique pour le stockage,
  - Le système assure l'allocation et la restitution des blocs (liste des blocs libres) pour gérer l'espace libre,
  - Gère l'allocation des blocs physiques sur le support,
- L'accès physique aux fichiers,
- Les liens entre la structuration logique et l'allocation physique,
  - Gère les répertoires et mémorise la structure de données du système de gestion de fichiers.
- Les protections associées aux fichiers, habituellement sous forme de :
  - listes de droits d'accès (droits `rwx` sous unix),
  - liste de contrôle d'accès (Access Control List — ACL).

Gestion des fichiers

L'OS doit assurer le transferts logique entre "système de fichier" et "processus utilisateurs". Il y a nécessité des deux mécanismes suivants :

- Un mécanisme de gestion des méthodes d'accès
  - Fonctions spécifiques à la structure de fichiers (open, close, read, write, ...)
  - Organisations traditionnellement implantées :
    - ★ Les fichiers séquentiels : accès exclusivement séquentiel ;
    - ★ Les fichiers séquentiels indexés : accès direct via une table d'index et des clés ;
    - ★ Les fichiers directs : accès direct par le numéro de l'enregistrement.
- Un mécanisme de gestion des tampons
  - Gère les tampons nécessaires aux entrées/sorties physiques,
  - Résident dans une zone mémoire du système pour rendre les entrées/sorties performantes,
  - Optimisation par heuristiques pour l'association tampons-blocs physiques.

Gestion des tâches

Rôle du système d'exploitation : gérer les programmes utilisateurs afin qu'ils puissent se réaliser correctement et si possible de manière « efficace ».
Gestion des processus à exécuter :

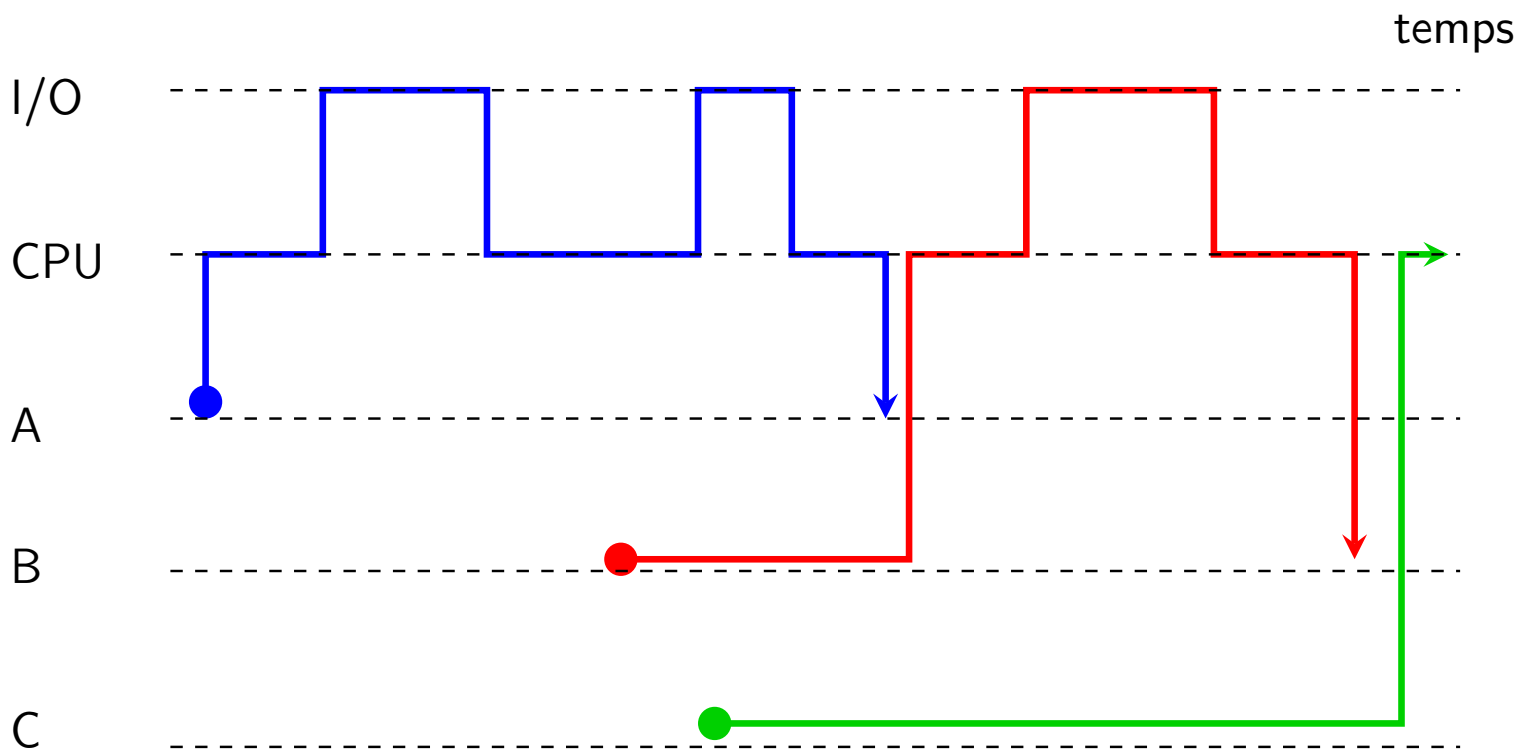
- Mono-tâche : une seule tâche peut être exécutée en même temps. Les autres tâches doivent attendre la fin de la tâche courante pour pouvoir commencer.
- Multi-tâches : plusieurs tâches peuvent s'exécuter en parallèle, sans bloquer le lancement d'une nouvelle tâche supplémentaire.

Autre distinction :

- Systèmes mono-programmés
  - Premiers systèmes d'exploitation développés,
  - Un seul processus réside en mémoire et utilise toutes les ressources,
  - Ordinateur utilisé séquentiellement par les différents processus,
- Systèmes multi-programmés
  - Plusieurs processus résident en mémoire,
  - Partagent l'unité centrale,
  - Cèdent leur tour durant les entrées/sorties.

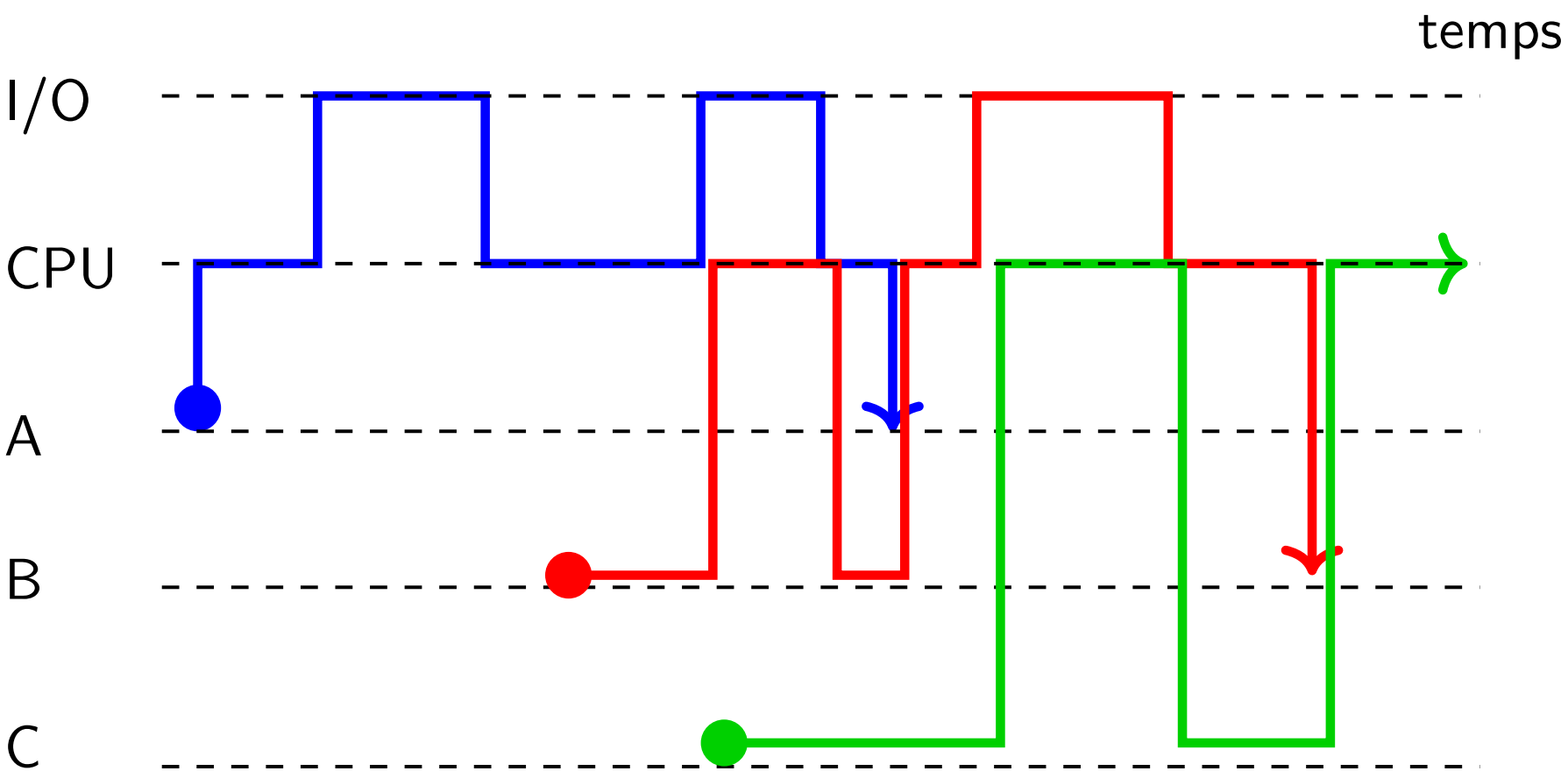
Gestion des tâches

Systèmes monoprogrammés



Gestion des tâches

Systèmes multiprogrammés



Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente		
Utilisé par le système		
Consommé par les utilisateurs		

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	
Utilisé par le système		
Consommé par les utilisateurs		

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	
Utilisé par le système	10%	
Consommé par les utilisateurs		

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	
Utilisé par le système	10%	
Consommé par les utilisateurs	40%	

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	5%
Utilisé par le système	10%	
Consommé par les utilisateurs	40%	

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	5%
Utilisé par le système	10%	35%
Consommé par les utilisateurs	40%	

Gestion des tâches

Temps unité centrale	monoprogrammé	multiprogrammé
En attente	50%	5%
Utilisé par le système	10%	35%
Consommé par les utilisateurs	40%	60%

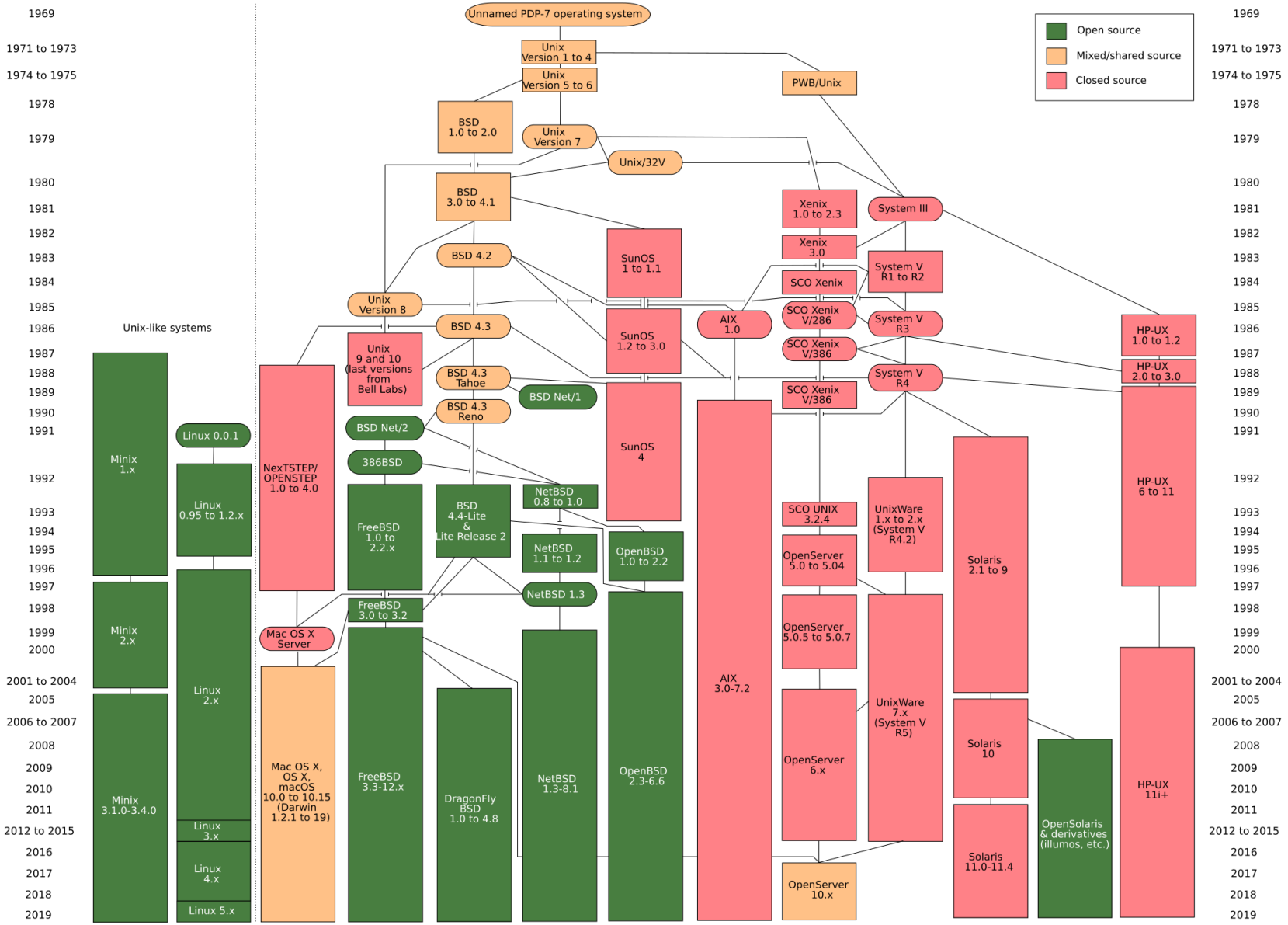
Types de système d'exploitation

Distinction de trois familles de systèmes d'exploitation selon leurs fonctionnalités :

- Systèmes temps réel
  Caractérisés par la prise en compte des contraintes temporelle de l'environnement,
- Systèmes de développement
  Systèmes à temps partagé interactifs,
- Systèmes pour l'exploitation
  Utilisé sur des « *main-frame* », traitement par lots (traitement de masse). Objectif : optimisation de l'utilisation coûteuse des ressources matériels.

Dans chacune de ces familles, on pourra de nouveau distinguer les caractéristiques des systèmes d'exploitations selon leur gestion des utilisateurs ou des tâches.

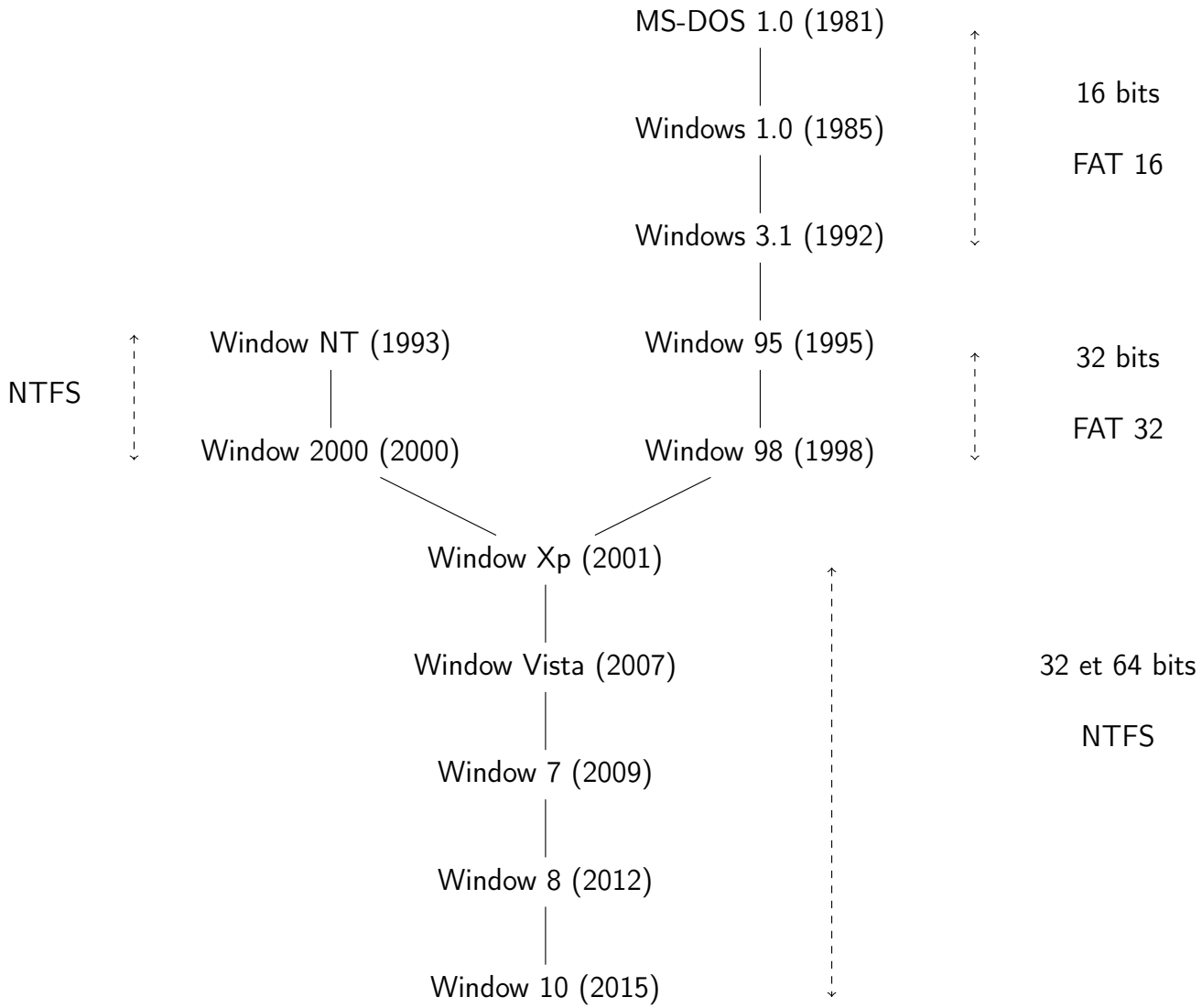
Unix


Open source
Mixed/shared source
Closed source
Unix-like systems
1969
1971 to 1973
1974 to 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 to 2004
2005
2006 to 2007
2008
2009
2010
2011
2012 to 2015
2016
2017
2018
2019
Unnamed PDP-7 operating system
Unix Version 1 to 4
Unix Version 5 to 6
PWB/Unix
BSD 1.0 to 2.0
Unix Version 7
Unix/32V
BSD 3.0 to 4.1
Xenix 1.0 to 2.3
System III
Xenix 3.0
BSD 4.2
SunOS 1 to 1.1
System V R1 to R2
SCO Xenix
Unix Version 8
SCO Xenix V/286
AIX 1.0
System V R3
BSD 4.3
SunOS 1.2 to 3.0
HP-UX 1.0 to 1.2
Unix 9 and 10 (last versions from Bell Labs)
SCO Xenix V/386
HP-UX 2.0 to 3.0
BSD 4.3 Tahoe
BSD Net/1
System V R4
BSD 4.3 Reno
SCO Xenix V/386
Minix 1.x
Linux 0.0.1
BSD Net/2
SunOS 4
NexTSTEP/ OPENSTEP 1.0 to 4.0
386BSD
HP-UX 6 to 11
Linux 0.95 to 1.2.x
FreeBSD 1.0 to 2.2.x
BSD 4.4-Lite & Lite Release 2
NetBSD 0.8 to 1.0
SCO UNIX 3.2.4
UnixWare 1.x to 2.x (System V R4.2)
Solaris 2.1 to 9
NetBSD 1.1 to 1.2
OpenBSD 1.0 to 2.2
OpenServer 5.0 to 5.04
Minix 2.x
NetBSD 1.3
FreeBSD 3.0 to 3.2
OpenServer 5.0.5 to 5.0.7
Mac OS X Server
AIX 3.0-7.2
Linux 2.x
UnixWare 7.x (System V R5)
Solaris 10
HP-UX 11i+
OpenServer 6.x
Mac OS X, OS X, macOS 10.0 to 10.15 (Darwin 1.2.1 to 19)
FreeBSD 3.3-12.x
DragonFly BSD 1.0 to 4.8
NetBSD 1.3-8.1
OpenBSD 2.3-6.6
Minix 3.1.0-3.4.0
Linux 3.x
Linux 4.x
Linux 5.x
Solaris 11.0-11.4
OpenSolaris & derivatives (illumos, etc.)
OpenServer 10.x

Unix

- Écrit à l'origine en assembleur,
- Le langage C a été créé pour Unix et Unix a été réécrit en C pour plus de portabilité,
- A l'origine de la norme Posix,
- Permet la communication entre machine,
- Système multi-utilisateurs,
- Système multi-tâches,
- Système de développement : beaucoup d'outils disponibles de base,
- Dispose d'interpréteurs de commandes puissants (shell),
- ...

Windows


MS-DOS 1.0 (1981)
Windows 1.0 (1985)
Windows 3.1 (1992)
16 bits
FAT 16
Window NT (1993)
Window 95 (1995)
NTFS
32 bits
FAT 32
Window 2000 (2000)
Window 98 (1998)
Window Xp (2001)
Window Vista (2007)
Window 7 (2009)
Window 8 (2012)
Window 10 (2015)
32 et 64 bits
NTFS

Ordonnancement
Allocation du processeur central

Allocation de l'UC

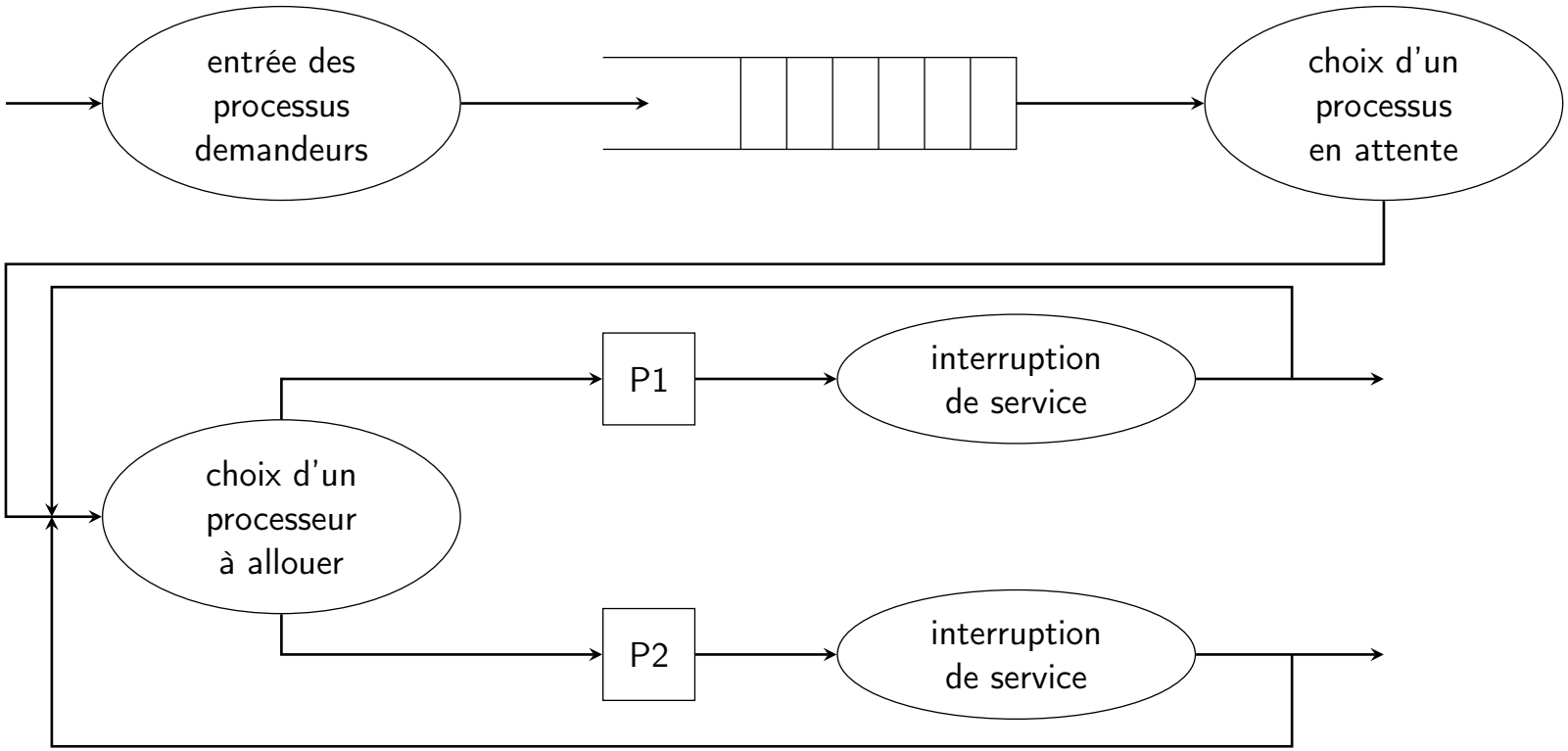
Objectif : trouver un équilibre entre la durée d'utilisation des ressources par les processus et le temps qu'ils mettent pour s'exécuter.
Il faut satisfaire les demandes d'exécution des programmes tout en respectant certaines contraintes :

- garantir un temps d'attente fini et une durée minimale d'allocation,
- respecter certaines priorités entre les processus,
- garantir l'allocation avant une date fixée,
- arrêter un processus dépassant une durée donnée,
- ...

Stratégies d'allocation

Les processus circulent selon le schéma suivant :

1. entrée des processus demandeurs,
2. choix d'un processus en attente,
3. choix d'un processeur à allouer ;
4. interruption de service.

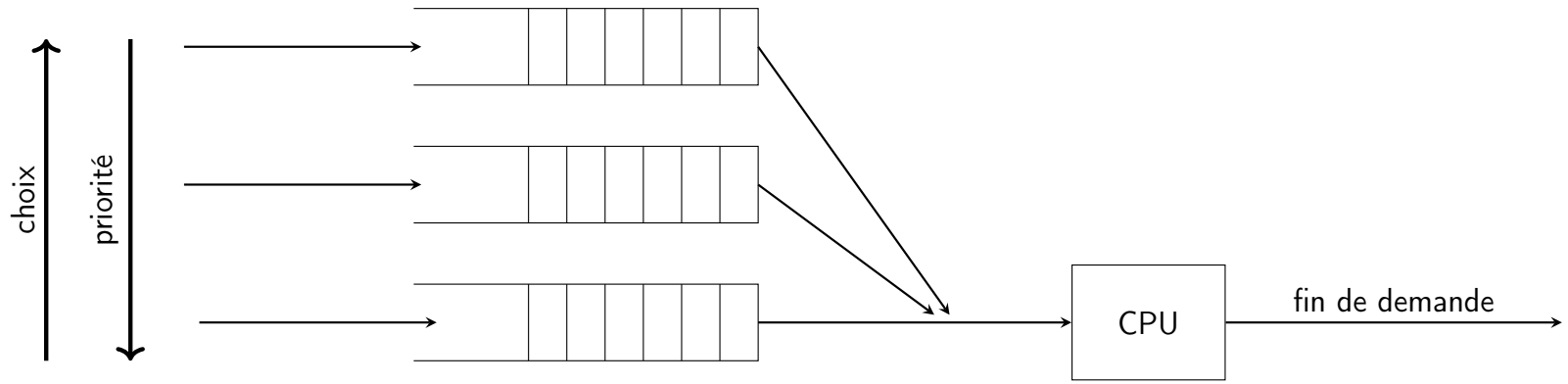


Stratégies sans recyclage des demandes

L'allocation de l'unité centrale est effectuée pour une demande complète, il n'y a pas de **préemption**.

La stratégie FIFO est simple et *a priori* efficace :

- FIFO impératif : on mélange les petits et les gros demandeurs, défavorise les petits ;
- FIFO indicatif : la file est triée sur un critère, avec insertions en milieu de file. L'efficacité est raisonnable, mais il y a pénalisation des gros demandeurs ;
- FIFO à priorités : plusieurs files gérées FIFO, classées par priorité, on ne choisit un demandeur dans une file que si les files de priorité supérieure sont vides.



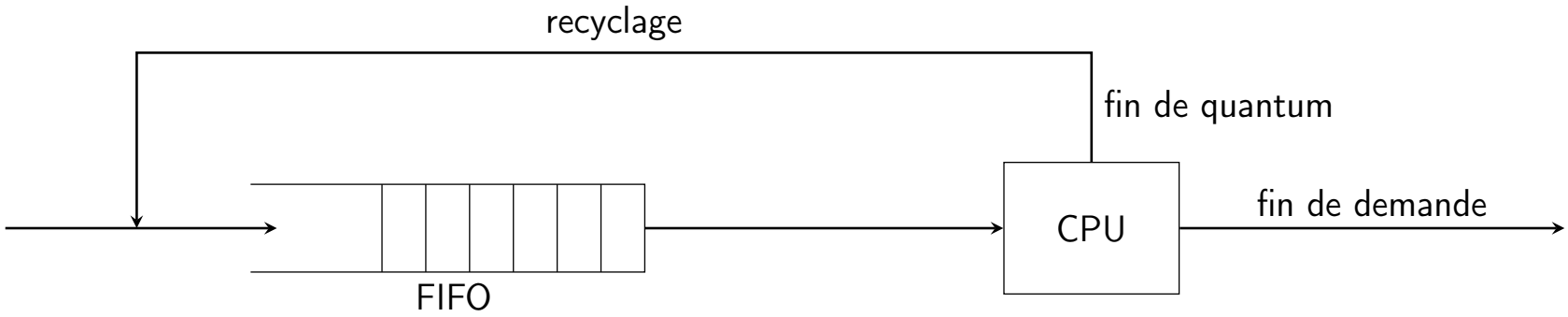
Stratégies avec recyclage

Allocation de l'unité centrale par tranche de temps appelée **quantum**, les demandes ne sont pas servies globalement, mais par fraction.
Le choix de la durée du quantum est fondamental :

- un quantum très court favorise les traitements interactifs,
- un quantum long augmente le débit global du système.

La stratégie du tourniquet

La stratégie du tourniquet (Round Robin) effectue l'allocation de l'unité centrale par quantum.



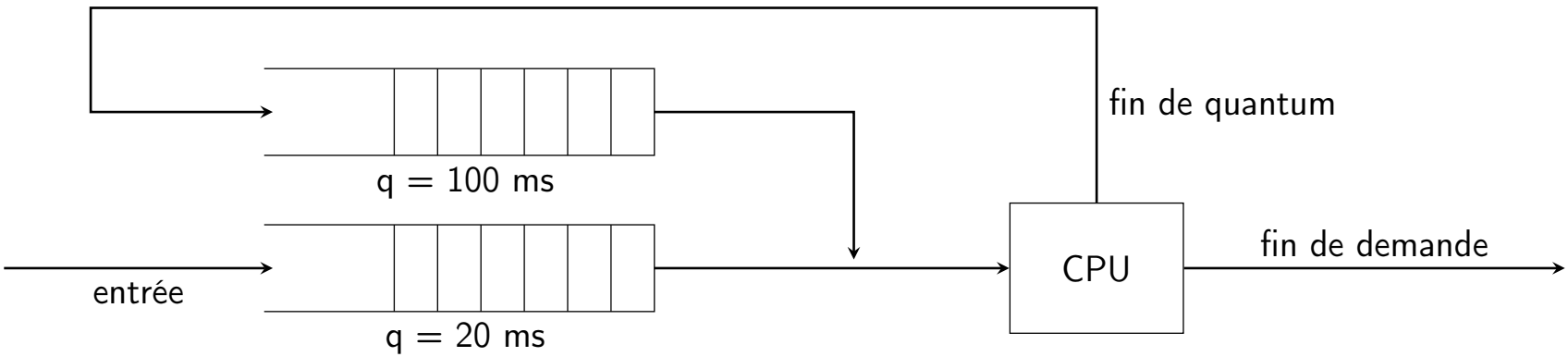
La stratégie du tourniquet à deux niveaux

Étude de la charge sur un système à temps partagé (IBM CP/CMS).
Constat : deux catégories de demandes

- les demandes courtes à caractère fortement interactif : on leur alloue un quantum de 20 ms,
- les demandes longues, correspondant aux travaux différés : on leur alloue un quantum de 100 ms.

Le recyclage provoque :

- une baisse de la priorité réalisée par un changement de file,
- une allocation plus longue.

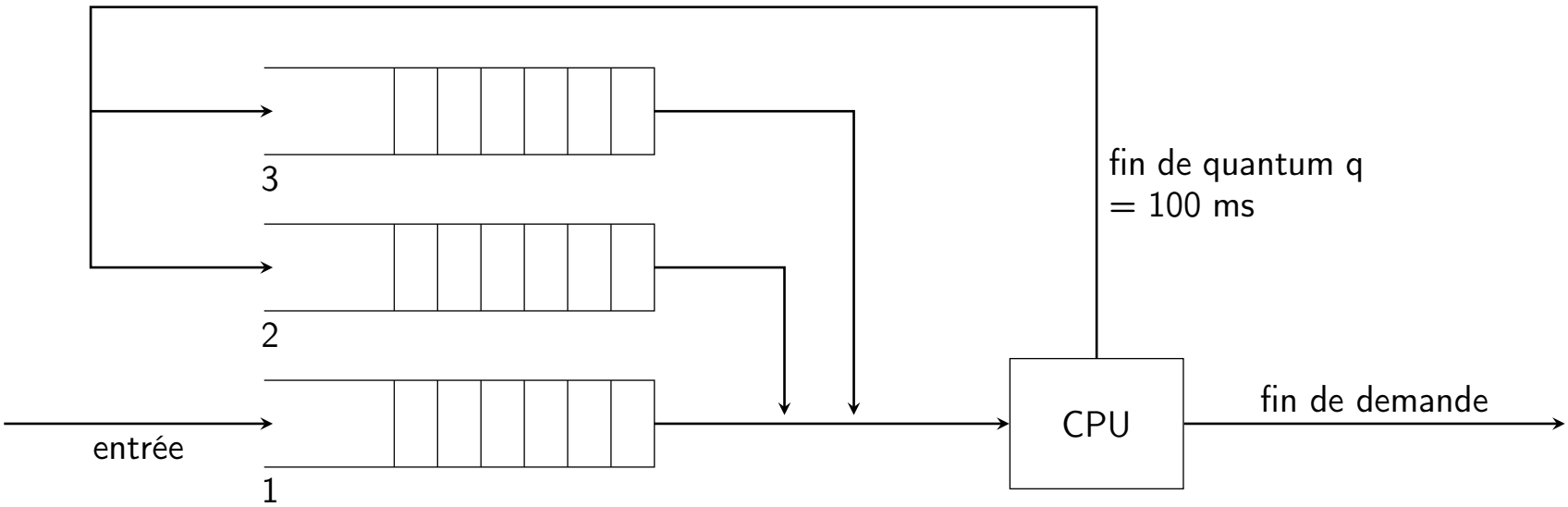


Unix

Stratégie tourniquet multiniveau à quantum unique :

- chaque processus prêt a une priorité flottante (reflète la consommation et la demande de CPU),
- chaque niveau de priorité correspond à une file FIFO,
- le tourniquet s'applique sur chaque file,
- la durée du quantum est de 100 ms.

Durée du quantum déterminée empiriquement et correspond à la durée maximale ne dégradant pas les travaux interactifs.



Stratégies avec échéance : Deadline

Essentiellement utilisées dans les systèmes temps réel pour le contrôle de processus industriels.

Exemple : de correction de la position d'un satellite

- Les coordonnées du satellite sont envoyées au calculateur tous les ΔT.
- Le traitement d'une position dure k secondes, doit être achevé avant l'arrivée de la position suivante.
- Lors de l'arrivée des coordonnées d'une position X à l'instant t, on détermine l'échéance E comme la date $t + \Delta T - k$.
- Si le traitement des coordonnées de cette position X n'est pas commencé au temps E, le traitement n'est pas ordonnancé et ne sera pas effectué.

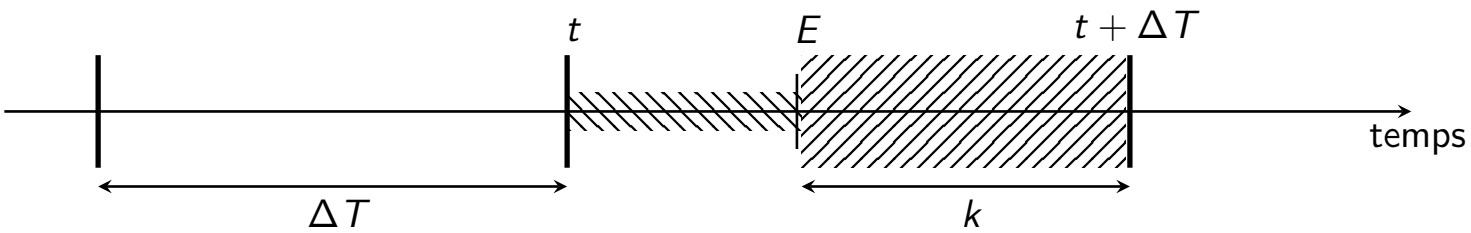


Figure – Stratégie à échéance.

Allocation de l'unité centrale dans le système Unix BSD

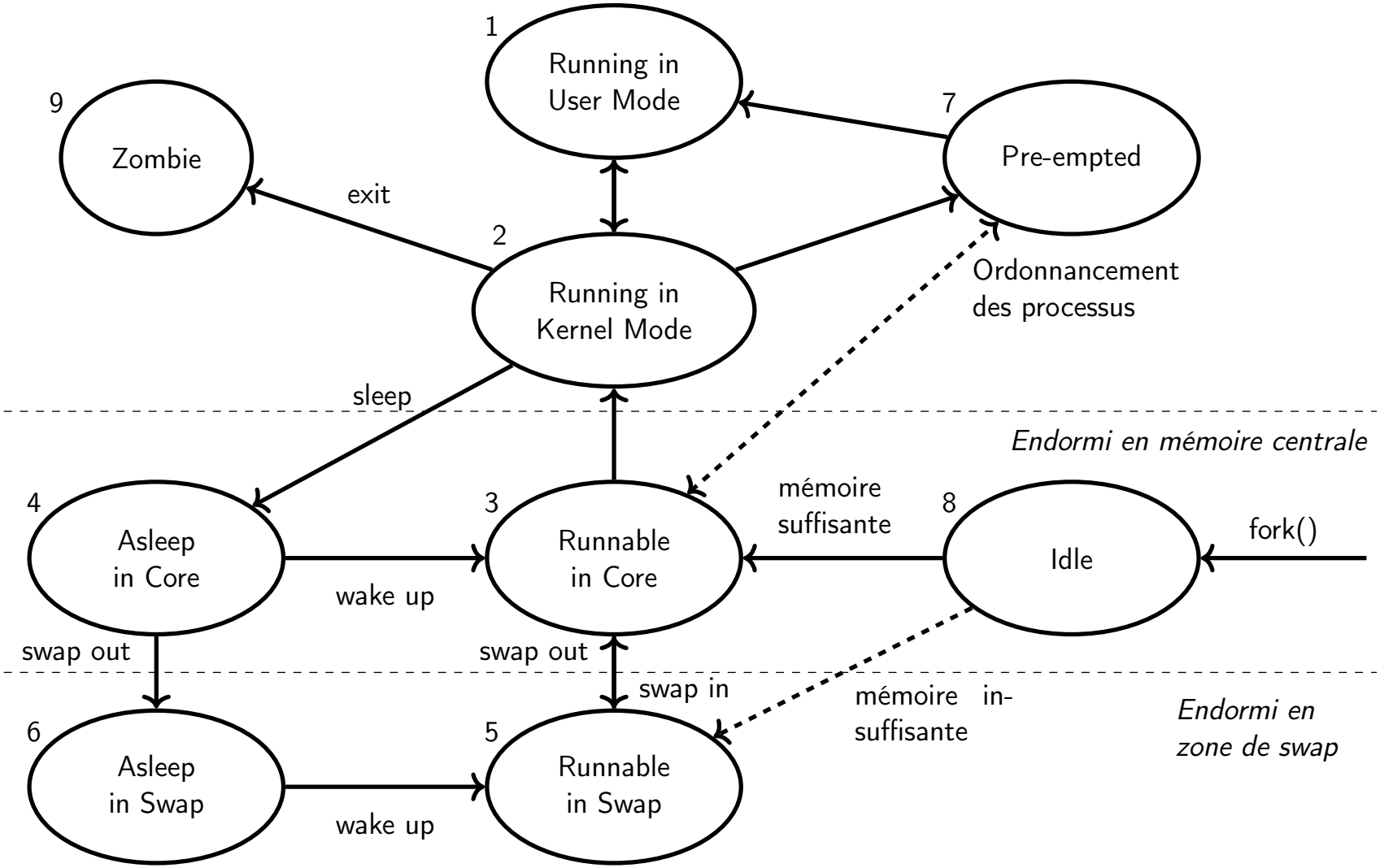
États d'un processus

Au cours de leur exécution les processus peuvent se trouver dans différents états avec des transitions possibles.
Les principaux états sont :

- *Idle* : le processus en cours de création (embryonnaire), et ne peut s'activer ;
- *Runnable* ou *Prêt* : le processus attend son tour pour utiliser le processeur ;
- *Running* ou *Actif* : le processus utilise le processeur, c'est son tour ;
- *Zombie* : le processus est en cours de destruction, il ne peut plus s'activer.

Les processus se trouvant dans l'état *Runnable* sont regroupés dans les files d'attente gérées par l'ordonnanceur.

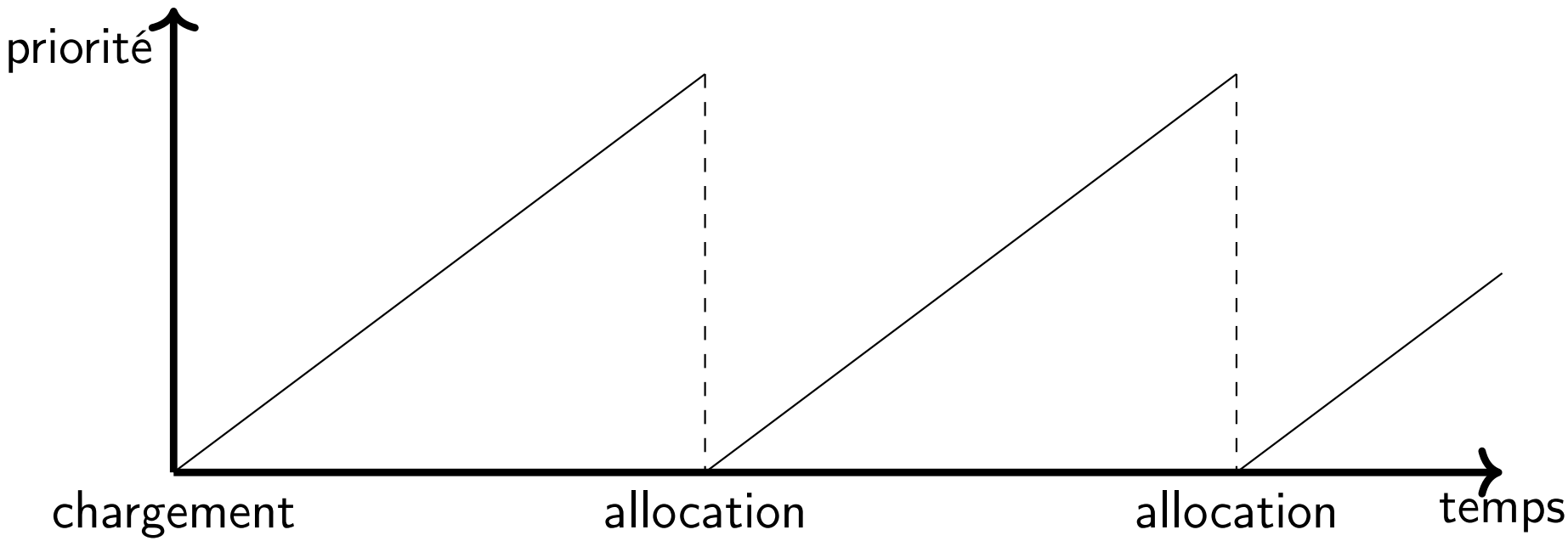
États d'un processus



L'ordonnancement des processus

Stratégie du tourniquet multi-niveaux géré en FIFO avec quantum.
Choix sur la priorité flottante des processus.
Évolution de la priorité

- la priorité augmente avant le service
- la priorité diminue fortement après le service



Structuration de l'ordonnanceur

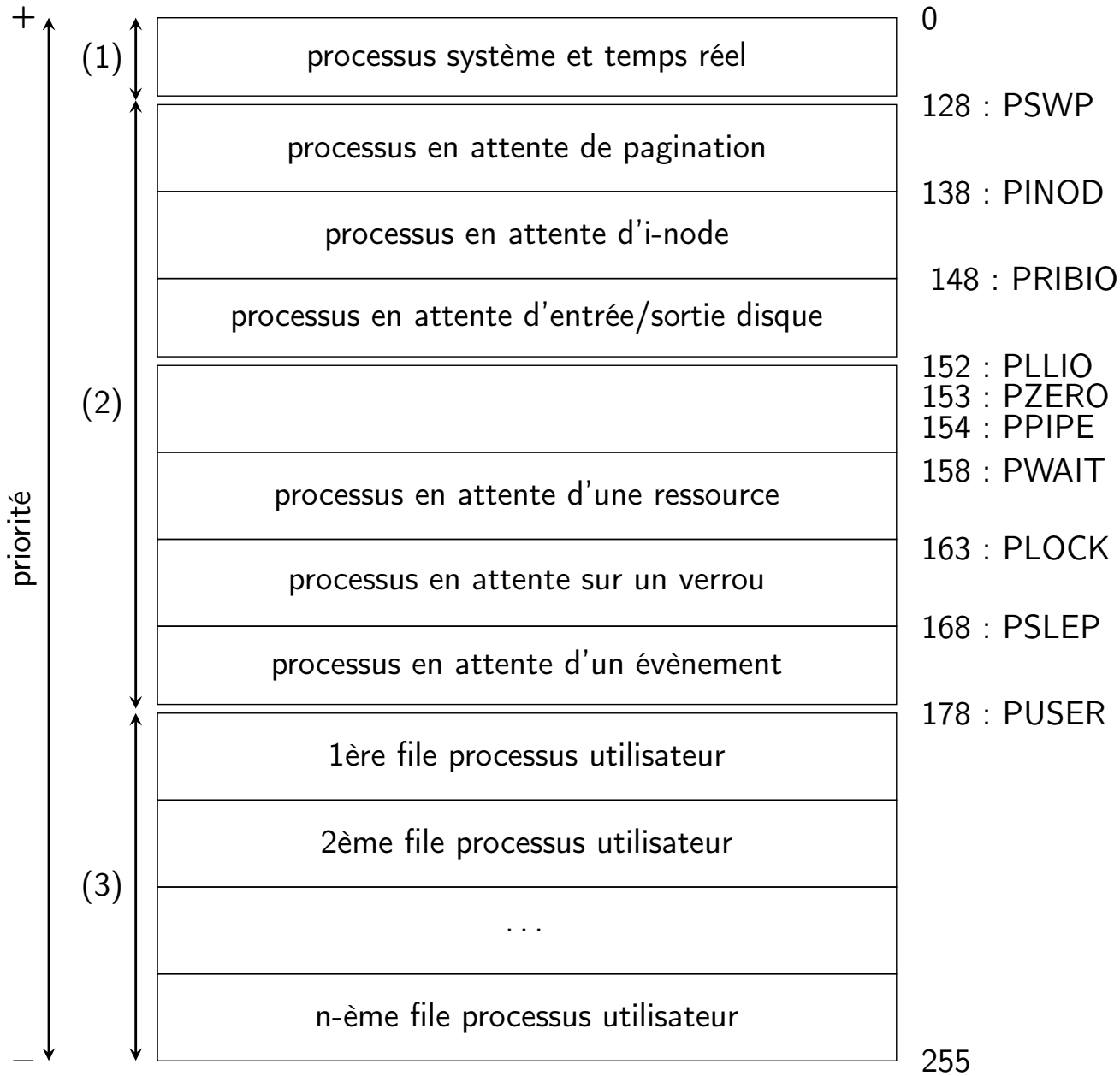
Deux familles de fonctions noyau assurent la gestion des processus :

- les fonctions appelées par les processus pour leur synchronisation et leur communication.
- les fonctions appelées par les activités liées au traitement des interruptions horloge. Cet ensemble de fonctions constitue l'ordonnanceur :
  - `schedcpu` est activée toutes les secondes et recalcule la priorité flottante des processus ;
  - `roundrobin` est activée 10 fois par seconde et assure le temps partagé en allouant à chaque processus un quantum de 100 ms et en faisant circuler les processus dans les files ;
  - `hardclock` est activée 100 fois par seconde et met à jour la variable `p_cpu` comptabilisant la consommation en temps unité centrale du processus ;
  - `setpri` est activée toutes les 40 ms et recalcule la priorité flottante des processus ;
  - `switch` recherche le processus le plus prioritaire et opère le changement de contexte pour son activation.

Organisation des files d'attente dans le système HP-UX

- Les files d'attente du système HP-UX regroupent l'ensemble des processus prêts ou en attente ;
- 256 niveaux de priorité flottante (de 0 à 255) ;
- À chaque niveau correspond une file d'attente ;
- Les processus en attente sont rangés dans les files correspondant à leur priorité de réveil ;
  - Priorité supérieure aux processus utilisateurs prêts car exécute au réveil une fonction système prioritaire ;
  - La priorité d'attente est totalement différente de la priorité des processus utilisateurs prêts gérés par l'ordonnanceur.

Organisation des files d'attente dans le système HP-UX



Algorithme simplifié de l'ordonnanceur

Pour tous les processus appartenant à la *runqueue* :

1. choisir le processus résidant en mémoire centrale ayant la plus forte priorité,
2. ôter le processus de la *runqueue* et effectuer un changement de contexte pour l'exécuter,
3. si aucun processus n'est éligible, mise de la machine dans l'état *idle*.

Principe du calcul de la priorité

- Pour calculer la priorité flottante des processus prêts, l'ordonnanceur prend en compte deux paramètres :
  - le temps CPU précédemment consommé par le processus et comptabilisé dans la variable *p_cpu*,
  - le temps d'attente passé dans la *runqueue*.
- La priorité des processus en attente est fonction :
  - des caractéristiques de l'évènement ou de la ressource attendu,
  - des ressources bloquées par le processus en attente.
- La priorité de réactivation des processus après une attente est calculée en utilisant la durée de l'attente comptabilisée dans la variable *p_slptime*.

Règles de calcul de la priorité flottante (Unix BSD)

Différentes variables :

- *load* : estimation de la charge, mémorise la longueur de la file d'attente des processus prêts ;
- *p_nice* : permet à l'utilisateur de moduler sa priorité ;
- *p_cpu* : est incrémentée à chaque interruption horloge, appelée tick (toutes les 10 ms).

Toutes les secondes, la variable *p_cpu* est ré-ajustée en utilisant une fonction de filtrage prenant en compte un estimateur de la charge du système.

$$p\_cpu = \left(\frac{2\ load}{2\ load + 1}\right) p\_cpu + p\_nice \qquad (1)$$

Règles de calcul de la priorité flottante (Unix BSD)

La priorité flottante est recalculée toutes les 40 ms (après 4 ticks d'horloge) selon la formule suivante :

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2\ p\_nice \tag{2}$$

Cette formule provoque une baisse linéaire de la priorité selon le temps unité centrale consommé.
Lors de la réactivation d'un processus utilisateur en attente dans l'état *endormi*, l'ordonnanceur calcule la valeur de la variable p_cpu à l'aide de la formule suivante :

$$p\_cpu = p\_cpu \left(\frac{2\ load}{2\ load + 1}\right)^{p\_slptime} \tag{3}$$

Le système comptabilise le temps d'attente dans la variable $p\_slptime$ qui est réinitialisée à zéro lors de la mise en attente.

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

- Un seul processus est actif et consomme toute l'unité centrale : la variable *load* vaut 1,
- Ce processus accumule T_i ticks à la fréquence de l'horloge pendant la durée i,
- Toutes les secondes, le filtre est appliqué avec la formule suivante :

$$p\_cpu = 0.66\ p\_cpu + p\_nice \tag{4}$$

- On suppose que p_nice vaut 0.

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

$$p\_cpu = \qquad 0.66\ T_0$$

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

$$p\_cpu = 0.66\ T_0$$
$$p\_cpu = 0.66\ (T_1 + 0.66\ T_0) = 0.66\ T_1 + 0.44\ T_0$$

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

$$
\begin{aligned}
p\_cpu &= 0.66\ T_0 \\
p\_cpu &= 0.66\ (T_1 + 0.66\ T_0) = 0.66\ T_1 + 0.44\ T_0 \\
p\_cpu &= 0.66\ T_2 + 0.44\ T_1 + 0.30\ T_0
\end{aligned}
$$

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

$$
\begin{aligned}
p\_cpu &= 0.66\ T_0 \\
p\_cpu &= 0.66\ (T_1 + 0.66\ T_0) = 0.66\ T_1 + 0.44\ T_0 \\
p\_cpu &= 0.66\ T_2 + 0.44\ T_1 + 0.30\ T_0 \\
p\_cpu &= 0.66\ T_3 + \cdots + 0.20\ T_0
\end{aligned}
$$

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

$$
\begin{aligned}
p\_cpu &= 0.66\ T_0 \\
p\_cpu &= 0.66\ (T_1 + 0.66\ T_0) = 0.66\ T_1 + 0.44\ T_0 \\
p\_cpu &= 0.66\ T_2 + 0.44\ T_1 + 0.30\ T_0 \\
p\_cpu &= 0.66\ T_3 + \cdots + 0.20\ T_0 \\
p\_cpu &= 0.66\ T_4 + \cdots + 0.13\ T_0
\end{aligned}
$$

Règles de calcul de la priorité flottante (Unix BSD)

Exemple

Simulation de l'évolution de la priorité :

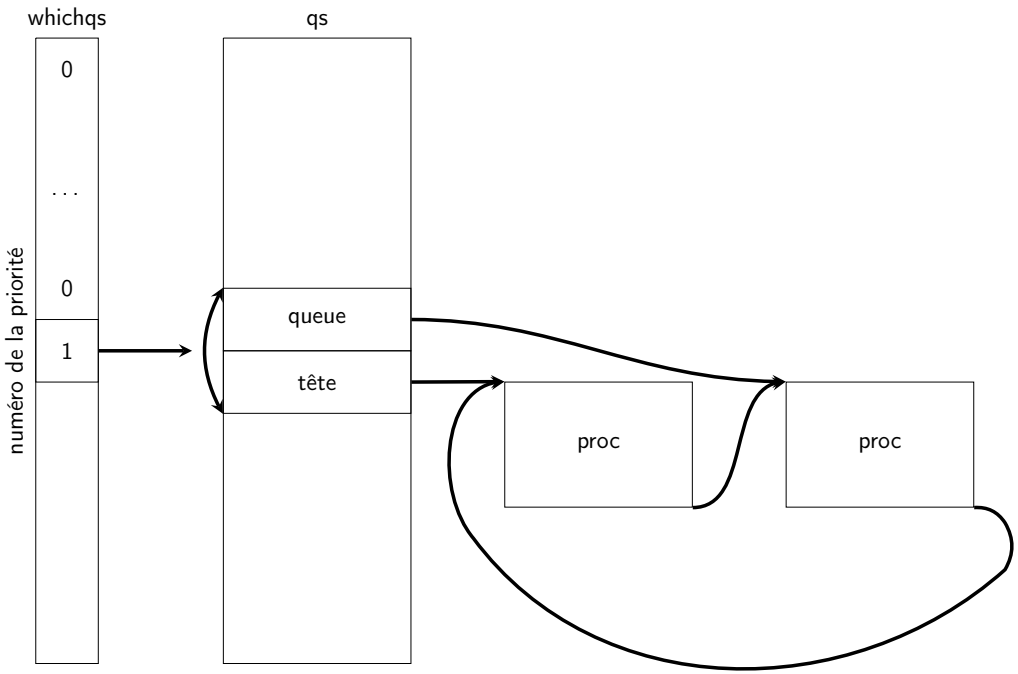
$$
\begin{aligned}
p\_cpu &= 0.66\ T_0 \\
p\_cpu &= 0.66\ (T_1 + 0.66\ T_0) = 0.66\ T_1 + 0.44\ T_0 \\
p\_cpu &= 0.66\ T_2 + 0.44\ T_1 + 0.30\ T_0 \\
p\_cpu &= 0.66\ T_3 + \cdots + 0.20\ T_0 \\
p\_cpu &= 0.66\ T_4 + \cdots + 0.13\ T_0
\end{aligned}
$$

Après cinq application du filtre (après 5 secondes), près de 90% du temps unité centrale consommé lors du premier quantum a été "*oublié*".

Implantation de la run queue (Unix BSD)

Run queue : ensemble des processus dans l'état *Runnable* (file des processus prêts).

- les niveaux de priorité flottante sont regroupés par 4,
- une liste chaînée des processus est associée à chaque groupe,
- la table *qs* contiente les têtes et queues de listes des files,
- la table booléenne *whichqs* (liée à *qs*) indique l'occupation de chaque file.



Priorité flottante dans Unix System V

A la fin de chaque quantum T_i on recalcule la consommation de temps CPU dans la variable CPU_usage selon la formule suivante :

$$CPU\_usage(T_i) = \frac{CPU\_usage(T_{i-1}) + nb(T_i)}{2}$$

où $nb(T_i)$ représente le nombre de ticks CPU accumulés pendant T_i.
Le calcul de la nouvelle priorité se fait selon cette formule :

$$Priorit(T_i) = P\_USER + \frac{CPU\_usage(T_i)}{2}$$

Priorité flottante dans Unix System V

Exemple

Soit le contexte suivant :

- 3 processus A, B et C dans la même file de priorité 178,
- un quantum dure 100 ticks d'horloge,
- le recalcul de la priorité se fait en fin de quantum,
- les processus ne font pas d'appels noyau,
- aucun autre processus ne se trouve dans l'état *Runnable*.

Priorité flottante dans Unix System V

Exemple

	Processus A			Processus B			Processus C		
Quanta	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks
T_1	**exécution** **178**	0 (P_USER)	1 . **100**	178 (P_USER)	0	0 . .	178 (P_USER)	0	0 . .
T_2									
T_3									
T_4									
T_5									

Priorité flottante dans Unix System V

Exemple

	Processus A			Processus B			Processus C		
Quanta	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks
T_1	**exécution** **178**	0 (P_USER)	1 . **100**	178 (P_USER)	0	0 . .	178 (P_USER)	0	0 . .
T_2	203	50 ($= \frac{0+100}{2}$)	0 . .	**exécution** **178**	0	1 . **100**	178	0	0 . .
T_3									
T_4									
T_5									

Priorité flottante dans Unix System V

Exemple

	Processus A			Processus B			Processus C		
Quanta	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks
T_1	**exécution** **178**	0 (P_USER)	1 . **100**	178 (P_USER)	0	0 . .	178 (P_USER)	0	0 . .
T_2	203	50 ($= \frac{0+100}{2}$)	0 . .	**exécution** **178**	0	1 . **100**	178	0	0 . .
T_3	190	25 ($= \frac{50+0}{2}$)	0 . .	203	50	0 . .	**exécution** **178**	0	1 . **100**
T_4									
T_5									

Priorité flottante dans Unix System V

Exemple

	Processus A			Processus B			Processus C		
Quanta	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks
T_1	**exécution** **178**	0 (P_USER)	1 . **100**	178 (P_USER)	0	0 . .	178 (P_USER)	0	0 . .
T_2	203	50 ($= \frac{0+100}{2}$)	0 . .	**exécution** **178**	0	1 . **100**	178	0	0 . .
T_3	190	25 ($= \frac{50+0}{2}$)	0 . .	203	50	0 . .	**exécution** **178**	0	1 . **100**
T_4	**exécution** **184**	12 ($= \frac{25+0}{2}$)	1 . **100**	190	25	0 . .	203	50	0 . .
T_5									

Priorité flottante dans Unix System V

Exemple

	Processus A			Processus B			Processus C		
Quanta	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks	Priorité	CPU_usage	ticks
T_1	**exécution** **178**	0 (P_USER)	1 . **100**	178 (P_USER)	0	0 . .	178 (P_USER)	0	0 . .
T_2	203	50 ($= \frac{0+100}{2}$)	0 . .	**exécution** **178**	0	1 . **100**	178	0	0 . .
T_3	190	25 ($= \frac{50+0}{2}$)	0 . .	203	50	0 . .	**exécution** **178**	0	1 . **100**
T_4	**exécution** **184**	12 ($= \frac{25+0}{2}$)	1 . **100**	190	25	0 . .	203	50	0 . .
T_5	206	56 ($= \frac{12+100}{2}$)	0 . .	**exécution** **184**	12	1 . **100**	190	25	0 . .

Les structures de données décrivant les processus

Dans le système, deux tables décrivent les processus :

- la table `proc`, décrite par la structure `proc.h` ;
- la table `user`, décrite par la structure `user.h` ;
- les structures `proc` et `user` d'un processus sont au même indice dans les deux tables ;
- possèdent *nproc* entrées, une entrée par processus.

La structure *proc*

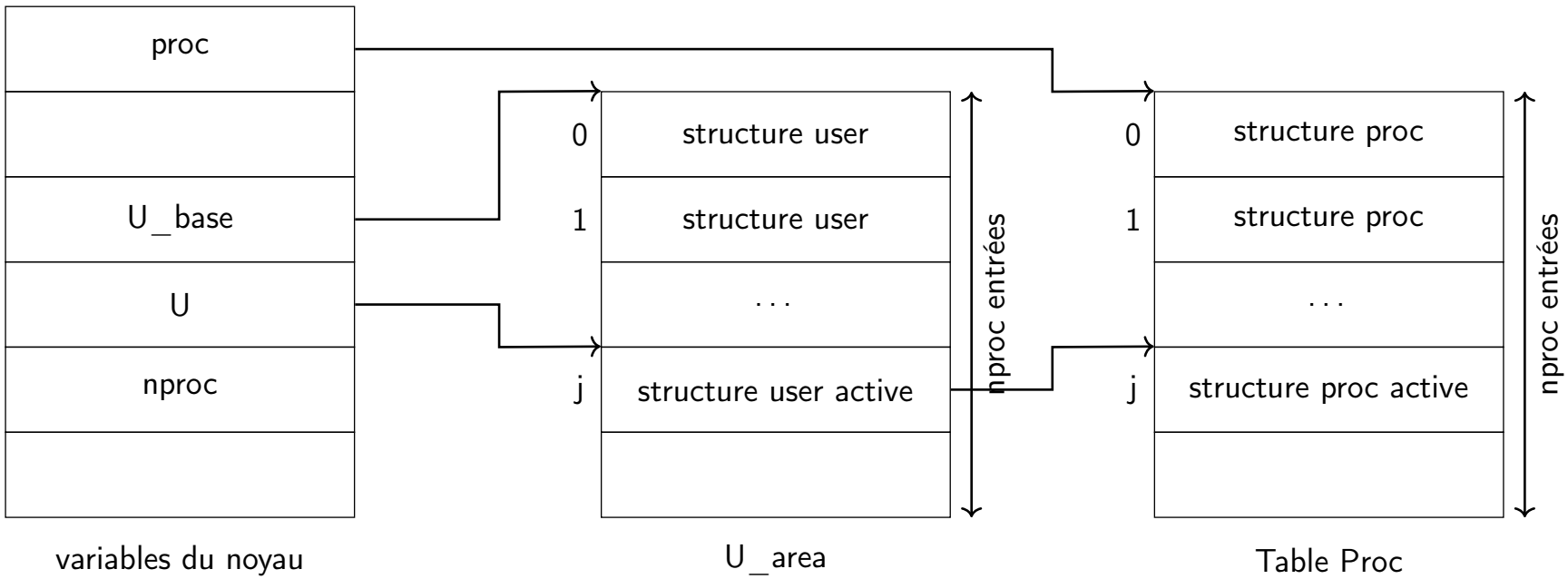
Réside en mémoire et contient essentiellement :

- Le chaînage vers les autres entrées de la table (les files d'attente) :
  - La file des processus de même priorité,
  - La liste des processus partageant le même texte exécutable,
  - La liste des processus prêts (run queue).
- Les identificateurs associés au processus,
- Les variables associées à la gestion des priorités,
- Les variables associées à la gestion du temps,
- Les variables associées à la gestion des signaux,
- Les liens vers d'autres structures du système.

La stucture `user`

- Possède une entrée par processus présent en mémoire ;
- Accompagne les processus durant leurs va-et-vient entre la mémoire centrale et la mémoire secondaire ;
- Contient essentiellement :
  - Les pointeurs vers les zones de sauvegarde et l'entrée `proc` du processus,
  - Le nom du processus,
  - Les identificateurs associés au processus,
  - Les informations liées à la gestion mémoire,
  - Les informations liées à la gestion des signaux,
  - Les descripteurs de fichiers ouverts,
  - Les noms des chemins (`pathname`) du répertoire de travail (`working directory`) et du répertoire personnel (`home directory`).

Les structures user et proc.



Filiation des processus

- Le mécanisme de création des processus implique l'héritage des attributs ;
- La connaissance de l'arbre de filiation est fondamentale.
- La première structure `proc` de la table `proc` correspond au processus ancêtre de tous les autres : le processus `Init`.
- Chaque structure `proc` possède les pointeurs nécessaires à la description de la filiation :
  - Le pointeur sur le fils le plus récemment créé,
  - Le pointeur sur le père,
  - Le pointeur sur le cadet (frère plus jeune),
  - Le pointeur sur l'aîné (frère plus vieux, mais pas le plus vieux).

Filiation des processus