ENSA KHOURIBGA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES

Systèmes d'Exploitation et Unix



Pr : OMARI Kamal

Filière : GI

A.U : 2024/2025

Chapitre 2:
Gestion des processus

1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus
6. Création de processus

II.1 Introduction et définitions

1) Qu'est-ce qu'un processus?

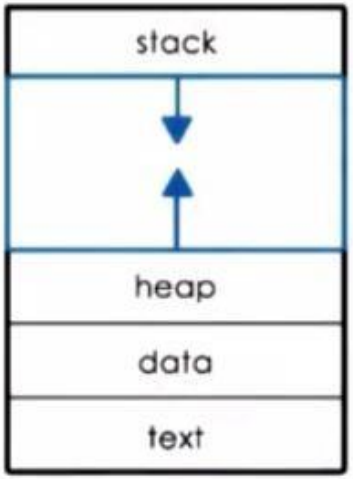
- Une tâche fondamentale des SE est d'assurer l'exécution de divers programmes.
- Un programme est une **entité statique** stockée dans le disque.
- Une fois chargé en mémoire pour s'exécuter, le programme devient un processus, qui est une **entité active**.
- *«Un processus peut être définit comme étant **une instance de programme en cours d'exécution ».***
- L'exécution d'un processus est en général une **alternance** de calculs effectués par l'UCT et de requêtes d'E/S effectuées par les périphériques.
- Un processus va **concourir** avec d'autres processus pour l'obtention d'une ressource (UCT, périphérique E/S,...).
- La gestion d'accès aux ressources est dirigée par la partie du SE appelée **ordonnanceur**.

II.1 Introduction et définitions

1) Qu'est-ce qu'un processus?

- Un processus est composé principalement:

  - ✓ Du code du programme (aussi appelé section texte du processus).
  - ✓ Du contenu des registres de l'UCT et de la valeur du compteur de programme (activité courante du processus).
  - ✓ De la pile du processus (stack), contenant les données temporaires (paramètres des fonctions, variables locales,…).
  - ✓ De la section données (data), contenant les variables globales du programme.
  - ✓ D'un tas (heap), qui est une mémoire dynamiquement allouée pendant l'exécution du processus (pour lecture de fichiers,…)

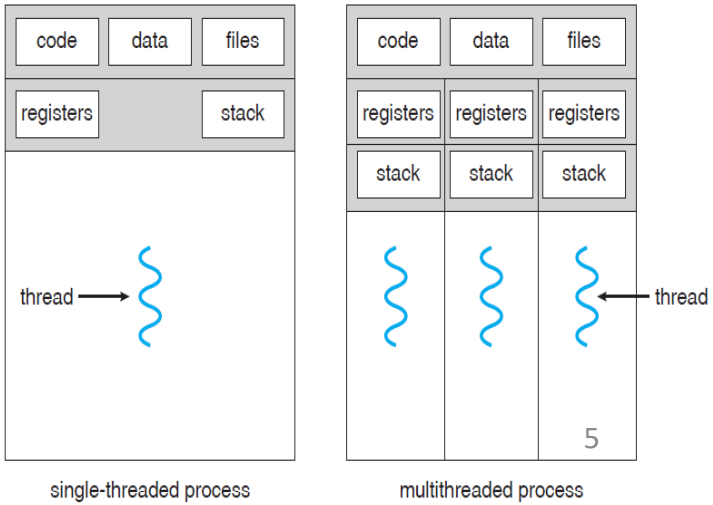


- Si un même programme est exécuté plusieurs fois, il correspond à plusieurs processus.
- Un processus peut communiquer des informations avec d'autres processus.

II.1 Introduction et définitions

2) Qu'est-ce qu'un processus léger (ou thread)?

- Un processus peut être composé d'un ou de plusieurs processus légers (threads).
- *«Un **thread** est une unité d'exécution rattachée à un processus, chargée d'en exécuter une partie. »*
  - Ex: pour un même document MS-Word, plusieurs threads: Interaction avec le clavier, sauvegarde régulière du travail, contrôle d'orthographe…)
- Un processus possède un ensemble de ressources (code, fichiers, périphériques…) que ses threads partagent.
- Cependant, chaque thread dispose :
  - d'un compteur programme (pour le suivi des instructions à exécuter)
  - de registres systèmes (pour les variables de travail en cours)
  - d'une pile (pour l'historique de l'exécution)



2) Qu'est-ce qu'un processus léger (ou thread)?

- Avantages des threads:
  - Réactivité: Le processus peut continuer à s'exécuter même si certaines de ses parties sont bloquées (en chargement de fichiers par exemple).
  - Economie d'espace mémoire: Partage de ressources, surtout la mémoire, entre threads d'un même processus.
  - Economie de temps: Les threads partagent les ressources du processus auquel ils appartiennent. Ainsi, il est plus économique de créer et de gérer les threads que les processus entre eux.
  - Scalabilité: Un processus à thread unique ne peut s'exécuter que sur une CPU. Alors qu'un processus à multithreads, peut s'exécuter sur plusieurs CPU (quand elles existent) en même temps.

3) Catégories des SE

Après avoir défini ces notions, les SE peuvent être divisés en 3 familles:

- Les SE mono-processus à thread unique(ex. DOS):
  - configuration la plus simple et la plus ancienne où un seul processus est exécuté à la fois.

- Les SE multiprocessus à thread unique (ex. Unix):
  - sur ces systèmes, l'allocation des ressources et l'ordonnancement de l'UCT agissent sur le processus et non pas les threads.

- Les SE multiprocessus multithread (ex: windows):
  - sur ces systèmes, des ressources sont allouées aux processus, mais l'ordonnancement de l'UCT agit sur les différents threads.

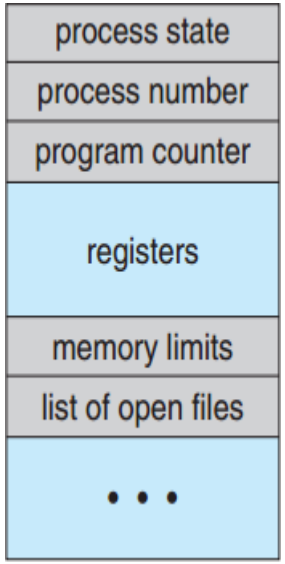
II.1 Introduction et définitions

4) Le bloc de contrôle du processus PCB

•Pour localiser et gérer tous les processus, le SE maintient une structure de données appelée «**table des processus**» qui contient les informations sur tous les processus créés.

•Le **Bloc de Contrôle de Processus** (Process control Bloc ou **PCB**) est une entrée dans cette table, composée principalement de:

- État de processus (En exécution, prêt, bloqué, ...)
- Identifiant du processus PID (unique)
- Compteur de programme (adresse prochaine instruction à exécuter par ce processus).
- Registres de l'UCT : varient en nombre et type selon l'architecture de l'ordinateur (dont l'accumulateur, le registre d'indexe, pointeurs de pile,…)
- Information d'ordonnancement (dont la priorité du processus, pointeurs sur les files d'ordonnancement, …)
- Information sur la gestion de la mémoire (dont les limites de la mémoire attribuée au processus).
- Information sur le statut des E/S (dont la liste des périphériques d'E/S alloués au processus)



5) Le changement de contexte

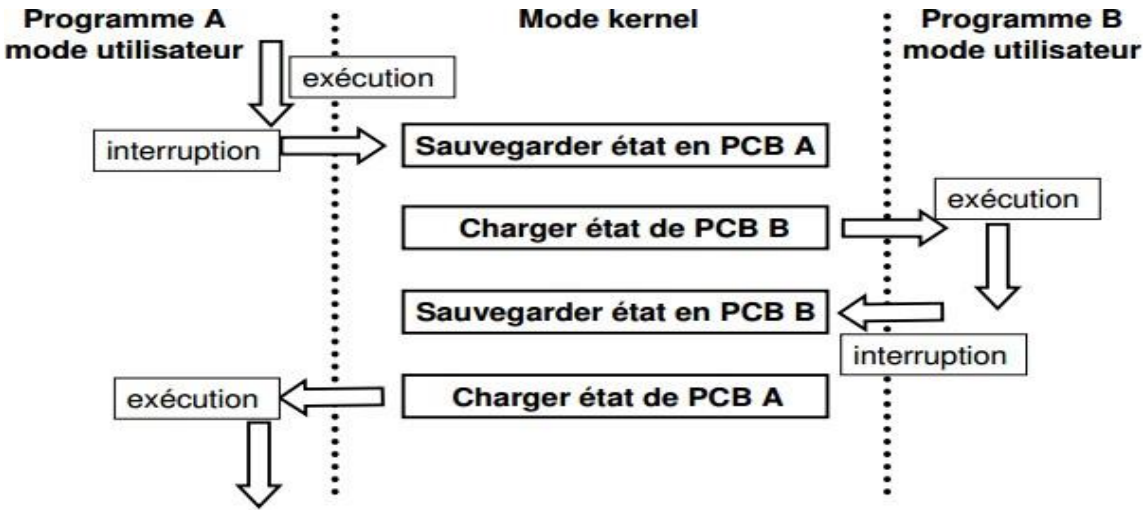
Pourquoi a-t-on besoin de toutes ces données (càd PCB)?

•Dans un système multiprogrammé, on a souvent besoin d'interrompre un processus et de redonner le contrôle de l'UCT à un autre processus.

•Il faut mémoriser toutes les informations nécessaires pour pouvoir relancer le processus courant dans le même état.

•Le processus en cours est interrompu et un ordonnanceur est appelé. Ce dernier s'exécute en mode noyau (kernel) pour pouvoir manipuler les PCB.

•Le changement de contexte a un coût: il va consommer de la mémoire et des cycles UCT, pour décharger le PCB du processus qui était en cours d'exécution et charger le PCB du processus qui va s'exécuter.



1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus
6. Création de processus

II.2 Niveaux d'ordonnancement des processus

Définition générale de l'ordonnanceur: partie du SE chargée d'allouer les ressources aux processus.

1) Notions utiles: équilibrage de travaux

•On peut distinguer entre 2 types de processus, selon le type de ressource qu'ils utilisent le plus:

- Les processus **tributaires de l'E/S:** utilisent peu l'UCT et beaucoup l 'E/S.
- Les processus **tributaires de l 'UCT:** utilisent beaucoup l'UCT et peu d'E/S.

Quel équilibre l'ordonnanceur devrait-il réaliser?

•Le temps d'UCT non utilisé par les processus tributaires de l'E/S peut être utilisé par les processus tributaires de l'UCT et vice-versa.

•L'UCT doit rester le moins possible inactive, sans pour autant saturer la mémoire principale du système.

•Equilibrage et priorité:

- Processus longs et non-urgents Vs Processus courts et urgents.
- L'ordonnanceur pourra donner la priorité aux deuxièmes et exécuter les premiers , quand il y a du temps machine disponible.

II.2 Niveaux d'ordonnancement des processus

1) Notions utiles: Traitement par lots Vs. Traitement interactifs

- Traitement par lots (batch):
  - Processus (ou Job) non-urgents qui sont soumis au système groupés et exécutés les uns après les autres (d'où le nom par lots), pour récupérer la réponse plus tard.
  - Il existe en général une relation entre les jobs successifs.
  - Exemple: Tri de fichier, calcul d'une fonction complexe, sauvegarde régulière de fichiers usagers, etc.
- Traitement Interactif:
  - Processus qui demandent une interaction continue avec l'ordinateur.
  - L'utilisateur reçoit le(s) résultat(s) immédiatement.
  - Exemple: édition de documents ou d'un programme.

II.2 Niveaux d'ordonnancement des processus

1) Notions utiles: les interruptions

Une interruption est un signal pour arrêter un processus, qui peut avoir plusieurs causes:

- Interruptions causées par le programme utilisateur:
  - Exception: Division par 0, débordement, tentative d'exécuter une instruction protégée, référence au delà de l'espace mémoire du programme
  - Appels Système: demande d'entrée-sortie, demande d'autre service du SE, minuterie établie par le programme lui-même.

- Interruptions causées par le SE:
  - Le processus doit céder l'UCT à un autre processus (Préemption).

- Interruptions causées par les périphériques ou par le matériel:
  - Fin d'une E/S.

II.2 Niveaux d'ordonnancement des processus

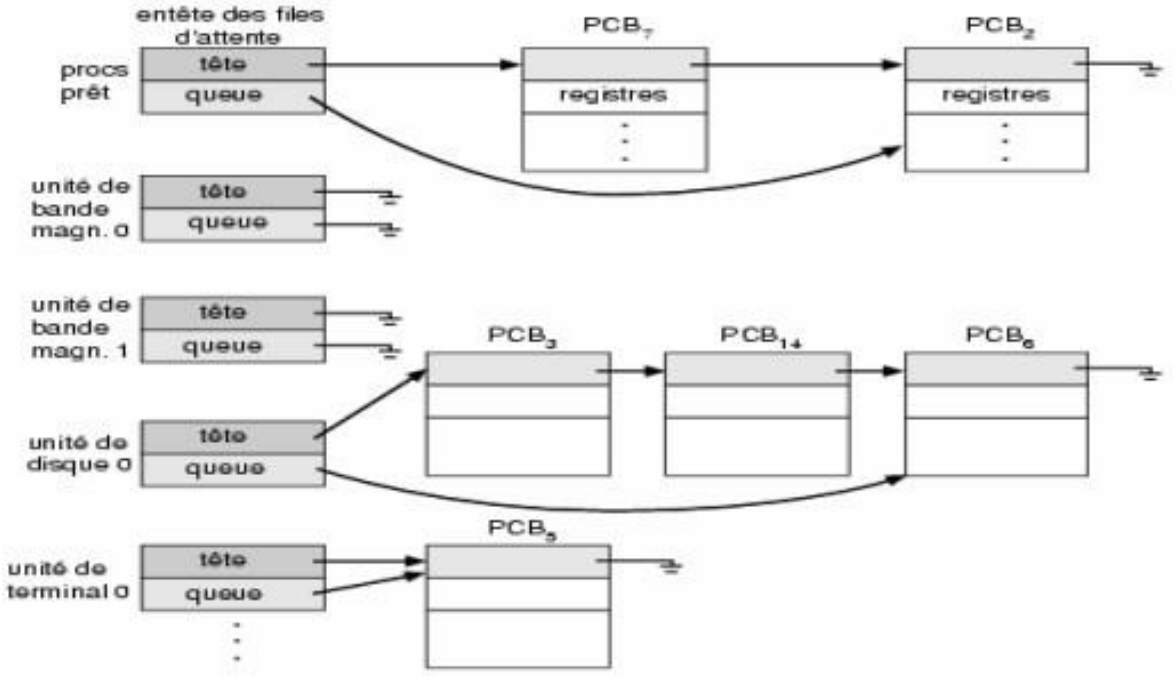
1) Notions utiles: les files d'attente

•Les processus qui résident dans la mémoire principale et sont prêts et en attente d'exécution sont conservés sur une liste appelée **File des Processus Prêts** (ou **Ready Queue**).

•**Chaque ressource** a sa propre file de processus en attente.

•C'est généralement une liste chaînée, contenant un pointeur vers le PCB du processus, et un pointeur vers le PCB du processus suivant dans la file.

•En changeant d'état, les processus se déplacent d'une file à l'autre.



II.2 Niveaux d'ordonnancement des processus

2) Définition générale des 3 niveaux d'ordonnancement

- Un processus passe une bonne partie de sa durée de vie dans divers files d'attente.

- La sélection d'un processus à partir de ces files d'attente est effectuée par l'ordonnanceur.
- L'ordonnanceur opère sur 3 niveaux:
  - ***L'ordonnanceur à long terme*** *(ou ordonnanceur de travaux)*: décide du moment où les processus vont être chargés en mémoire.
  - ***L'ordonnanceur à moyen terme*** *(ou ordonnanceur de mémoire ou permutateur)*: décide de la suspension/reprise des processus lors d'un manque de mémoire.
  - ***L'ordonnanceur à court terme*** *(ou ordonnanceur de processeus ou répartiteur)*: décide quel processus aura le contrôle de l'UCT.

II.2 Niveaux d'ordonnancement des processus

3) Ordonnanceur de travaux (à long terme)

Dans un système par lots, il existe plus de processus soumis que ceux capables d'être exécutés immédiatement.

✓**Ils sont alors chargés sur une file d'attente au niveau d'un périphérique de stockage de masse (le disque) pour être exécutés plus tard.**

•Le rôle de l'ordonnanceur de travaux est de:

1. Sélectionner les processus à partir de cette file d'attente.
2. Les charger dans une nouvelle file d'attente **des processus prêts** pour accéder à l'UCT.
3. Déterminer le niveau de multiprogrammation: nombre de processus en mémoire pouvant être exécutés en parallèle par le SE.

○ Le nombre est choisi d'une manière à établir un équilibre entre les processus tributaires de l'UCT et ceux tributaires des E/S.

•Une fois que le processus est admis par l'ordonnanceur de travaux, il n'en sort que lorsqu'il est terminé ou s'il est détruit par le SE (suite à une erreur grave ou à la demande de l'utilisateur).

•**N.B**: La plupart des systèmes interactifs multiprogrammés ne disposent pas d'ordonnanceur de travaux. Chaque nouveau processus est mis en mémoire principale pour être pris en charge par l'ordonnanceur à court terme.

II.2 Niveaux d'ordonnancement des processus

4) Ordonnanceur de mémoire ou permutateur (à moyen terme)

- Les SE multiprogrammés introduisent un ordonnanceur à moyen terme.
- Le rôle du permutateur est d'effectuer:
  - **Swap out:** Supprimer un processus de la mémoire principale et le placer en mémoire secondaire. Il ne sera plus en concurrence avec les autres pour les ressources.
  - **Swap in:** Ré-introduire plus tard le processus en mémoire principale et son exécution pourra se poursuivre là où elle a été interrompue.
  - Réduire ainsi le niveau de multiprogrammation.

- Le swapping (permutation) est nécessaire pour:
  - améliorer l'équilibre entre processus tributaires de l'E/S et ceux tributaires de l'UCT
  - remédier au problème de dépassement de la mémoire principale disponible.

- Le swapping ne doit pas être trop fréquent pour ne pas gaspiller la bande passante des disques.

5) Ordonnanceur de l'UCT ou répartiteur (à court terme)

Il est utilisé par tout type de SE et son rôle est principalement de:

- Choisir, parmi la file d'attente des processus prêts, à quel processus sera alloué l'UCT et pour quel laps de temps.
- Être très rapide pour ne pas ralentir le SE puisqu'il est très fréquemment utilisé.

1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus
6. Création de processus

II.3 Etats des processus

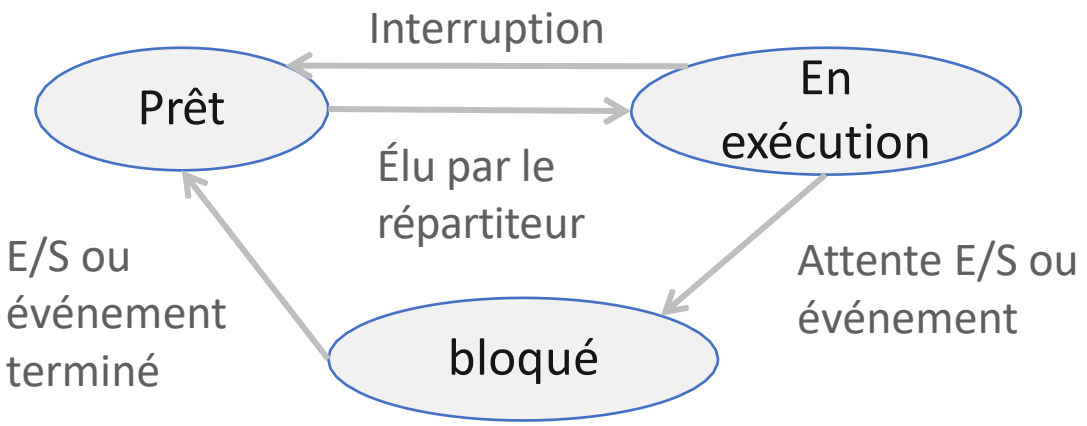
1) Les états des processus dans un répartiteur (ordonnanceur de processus)

•Les processus sont concurrents et se partagent l'UCT, ils ne peuvent être continuellement actifs. Ils ont donc, si on ne considère pour commencer que le répartiteur**, 3 états et 4 transitions possibles**:

•Prêt (ready): état d'un processus qui n'est pas alloué à l'UCT, mais qui est prêt à être exécuté.

•En exécution (running) : état d'un processus exécuté sur une UCT.

•Bloqué (blocked): état d'attente d'un événement extérieur, tel qu'une E/S, nécessaire à la poursuite de l'exécution du processus.

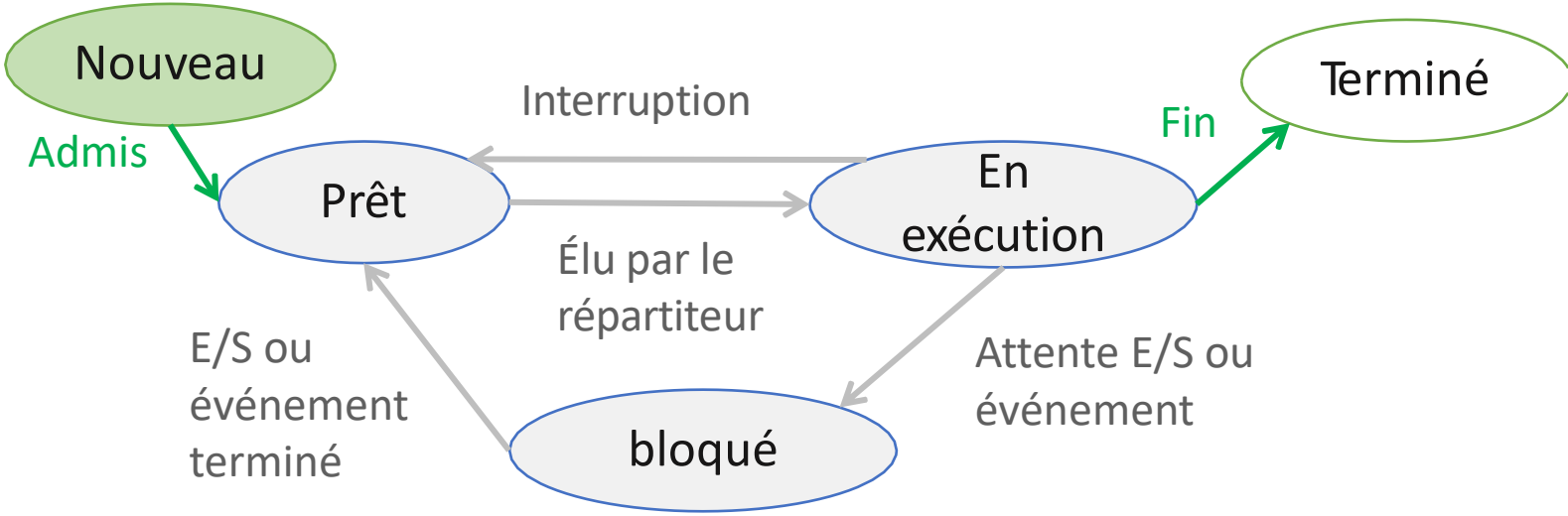


2) Les états des processus dans un ordonnanceur de travaux

2 états sont ajoutés aux états précédents lorsqu'un ordonnanceur de travaux est utilisé:

- Nouveau (New): le processus vient d'être créé mais n'est pas encore admis par l'ordonnanceur de travaux pour concurrencer à l'accès à l'UCT.

- Terminé (Terminated): le processus a achevé sa tâche. Il sera détruit prochainement par le SE pour libérer l'espace. Il est parfois conservé pendant un temps à l'état terminé en attendant qu'une E/S s'achève ou que les données de ce processus soient exploitées par un autre processus. On parle alors de processus " zombie".



N.B: Quelque soit son état, un processus peut prendre fin suite à une action externe: le SE ou un autre processus peuvent mettre fin à un processus, en le passant en état terminé.

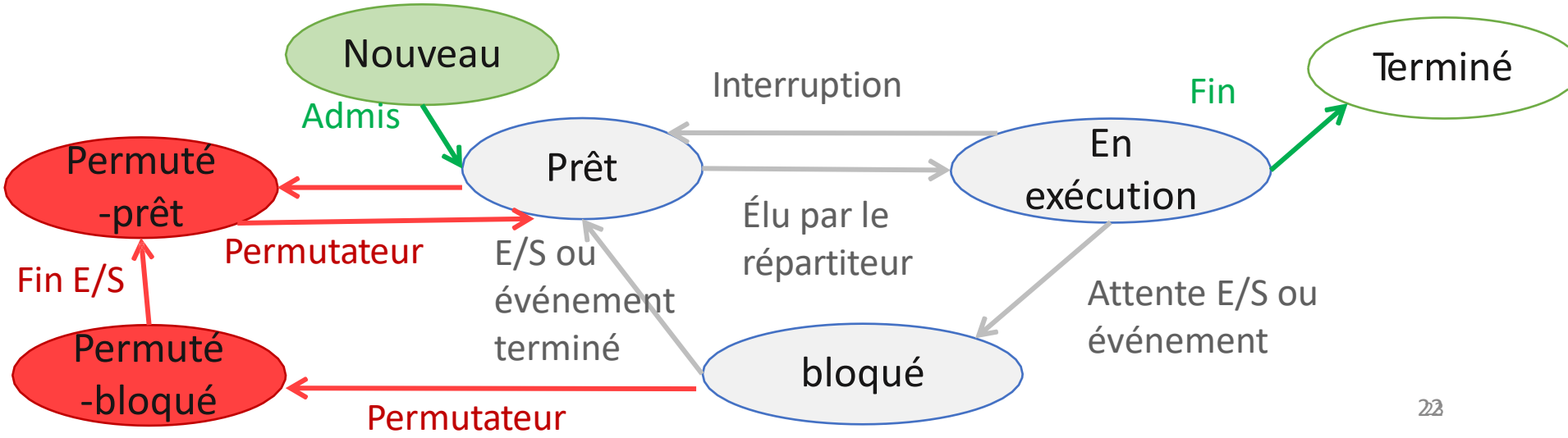
II.3 Etats des processus

3) Les états des processus dans un permutateur

2 états sont ajoutés aux états précédents lorsqu'un permutateur est utilisé:

- Permuté-prêt (Swapped-ready): le processus est pour l'instant transféré en mémoire secondaire. Le processus est réintroduit plus tard par le permutateur.

•Permuté-bloqué (Swapped-blocked): le processus était bloqué en attendant une E/S par exemple, puis a été transféré sur la mémoire secondaire pour faire de la place en mémoire principale. Lorsqu'il termine ses E/S, il passe à l'état permuté-prêt.



1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus

II.4 Algorithmes d'ordonnancement

1) Généralités

•Rôle: décider de l'allocation d'une ressource aux processus qui l'attendent: **algos d'ordonnancement pour l'UCT.**

•Objectif: aboutir à un partage efficace du temps d'utilisation de l'UCT:
Mais que veut dire efficace?

- L'algo doit identifier le processus qui conduira à la **«meilleure»** performance possible du système.
- Il existe **différents critères** pour mesurer la performance et dont l'importance est relative à l'algo lui même.

II.4 Algorithmes d'ordonnancement

2) Les critères de performance

- **Utilisation UCT** à maximiser: pourcentage d'utilisation pendant une période d'observation donnée.

•**Débit** (ou rendement, Throughput) à maximiser : nombre de processus complétés pendant une période d'observation donnée.

•**Temps de rotation** (ou de service, turnaround time) à minimiser: Temps écoulé entre le moment où un processus devient prêt à s'exécuter et le moment où il finit de s'exécuter.

•**Temps d'attente** (waiting time) à minimiser : somme de tout le temps passé en file prêt.

•**Temps de réponse** (response time) à minimiser: utile pour les systèmes interactifs. Temps écoulé entre la soumission d'une requête et la première réponse obtenue.

•**Equité :** degré auquel tous les processus reçoivent une chance égale de s'exécuter. On essaie ainsi d'éviter **la famine**: c'est le cas où un processus n'obtient pas la ressource

•**Priorités :**attribue un traitement préférentiel aux processus dont le niveau de priorité est élevé.

Remarque: En général, on tente d'optimiser **les valeurs moyennes** pour tous les processus mis en jeu pendant une période d'observation donnée, pour les temps **d'attente**, de **rotation** et de **réponse**.

II.4 Algorithmes d'ordonnancement

3) Non préemptif Vs Préemptif

Il existe 2 types d'algo. Ordonnancement:

- **Non préemptif** (ou coopératif ou sans réquisition):
  - **Définition:** le processus sélectionné garde le contrôle de l'UCT jusqu'à ce qu'il se bloque ou qu'il termine.

  ✓Avantages: facile à mettre en œuvre. Ne nécessite pas de mécanismes matériels spécifiques (horloges, …)

  ↓Inconvénients: Correspond difficilement aux systèmes interactifs ou le temps de réponse est important.

  - **Préemptif** (avec réquisition) :
  - **Définition:** l'algo retire l'UCT au processus en cours d'exécution pour l'attribuer à un autre processus. Ce type est indispensable pour les système interactifs.

  ✓Avantages: Convient aux systèmes interactifs

  ↓Inconvénients: Commutation fréquente de contexte des processus, ce qui peut diminuer le débit.

4) First come first served (FCFS)

•Algo non préemptif très simple. Il est aussi appelé PAPS (Premier Arrivé Premier Servi).

•L'ordonnancement est fait dans l'ordre d'arrivée en gérant une file FIFO (First In First Out) unique des processus prêts, sans priorité ni réquisition : le processus élu est celui qui est en tête de liste.

•Chaque processus s’exécute jusqu’à son terme.

✓**Avantages:**

- Simple
- Pas de famine

↓**Inconvénients:**

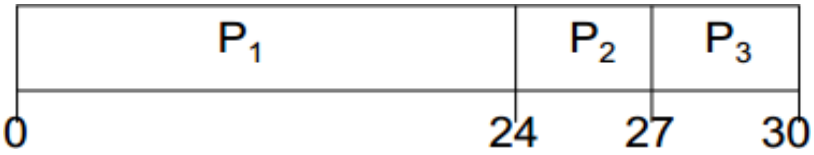
- Temps d’attente moyen très important.
- Non adapté aux systèmes interactifs.

4) First come first served (FCFS) – Exemple

- Soient les processus P1 , P2 , P3 qui arrivent à l'instant 0 dans cet ordre.

Processus	Cycle UCT
P1	24
P2	3
P3	3

- Le diagramme de Gant suivant correspondant à l'algo FCFS:



- **Les mesures de performances sont:**
  - Utilisation UCT= 30/30 (100%)
  - Temps d'attente moyen= (0+24+27)/3=17
  - Temps de rotation moyen=(24+27+30)/3=27
  - Débit= 3/30=0.1

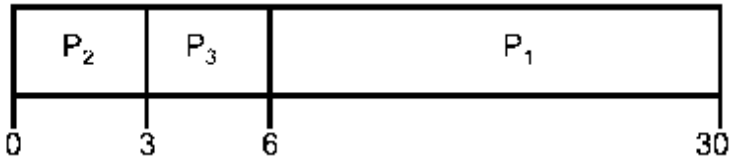
Processus	Temps attente	Temps de rotation
P1	0	24
P2	24	27
P3	27	30

4) First come first served (FCFS) – Exemple 2

- Soient les mêmes processus P1 , P2 , P3 qui arrivent à l'instant 0 dans l'ordre P2, P3, P1

Processus	Cycle UCT
P1	24
P2	3
P3	3

- Le diagramme de Gant correspondant à l'algo FCFS:



- **Les mesures de performances sont:**
  - Temps d'attente moyen= (6+0+3)/3=3
  - Temps de rotation moyen=(30+3+6)/3=13
  - Débit= 3/30=0.1

Processus	Temps attente	Temps de rotation
P1	6	30
P2	0	3
P3	3	6

- **Remarque:** Lorsque les processus les plus courts sont arrivés en premier (donc élu en premier), les performances sont nettement meilleurs. On pourrait donc penser à utiliser un algo qui avantage les processus courts => **algo SJF**

II.4 Algorithmes d'ordonnancement

5) Shortest Job First (SJF)

- Algo non préemptif. Il est aussi appelé Plus court temps d'exécution (PCTE)
- Le processus qui a le cycle UCT le plus court est exécuté en premier.
- Le FCFS est utilisé en cas d'égalité.

✓**Avantages:**

- Le meilleur pour le temps d'attente moyen (lorsque tous les processus arrivent en même temps.)

↓**Inconvénients:**

- Risque de famine: les processus longs peuvent ne jamais s'exécuter
- Nécessite de connaitre à l'avance le temps du cycle UCT (adapté aux traitements par lots où une estimation de la durée du cycle est donnée). Sinon, il devrait être prédit

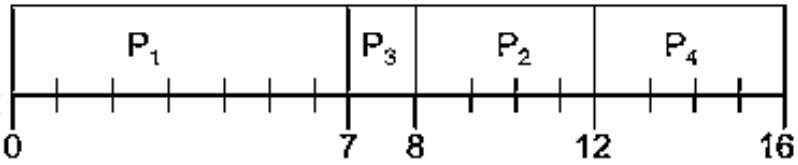
II.4 Algorithmes d'ordonnancement

5) Shortest Job First (SJF) – Exemple

- Soit les processus P1 , P2 , P3, P4 qui arrivent selon différents temps d'arrivée.

Processus	Arrivée	Cycle UCT
P1	0	7
P2	2	4
P3	4	1
P4	5	4

- Le diagramme de Gant correspondant à l'algo SJF:



- **Les mesures de performances sont:**
  - Temps d'attente moyen= (0+6+3+7)/4=4
  - Temps de rotation moyen=(7+10+4+11)/4=8
  - Débit= 4/16=0.25

Processus	Temps attente	Temps de rotation
P1	0	7
P2	8-2=6	12-2=10
P3	7-4=3	8-4=4
P4	12-5=7	16-5=11

6) Shortest Remaining Time First(SRTF)

•C'est la version préemptive du SJF. Il est aussi appelé Plus court temps d'exécution avec Réquisition (PCTER)

•Chaque fois qu'un nouveau processus est introduit dans la file des processus prêts, l'ordonnanceur compare sa durée du cycle UCT à la durée restante du processus en cours d'exécution. Si la durée du nouveau processus est inférieure, le processus en cours d'exécution est réquisitionné.

✓ **Avantages:**

- Plus efficace que SJF, car le temps d'attente moyen optimal est garantit quelque soit le moment d'arrivée des processus

↓ **Inconvénients:**

- Risque de famine
- Besoin de connaitre la durée du cycle UCT à l'avance

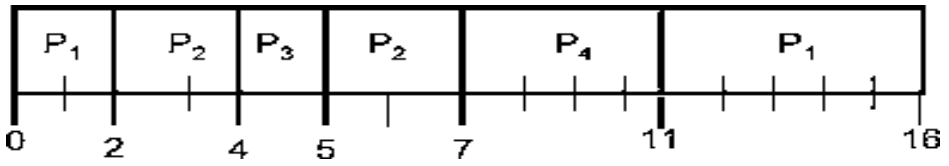
II.4 Algorithmes d'ordonnancement

6) Shortest Remaining Time First (SRTF)

Soit les processus P1 , P2 , P3, P4 qui arrivent selon différents temps d'arrivée.

Processus	Arrivée	Cycle UCT
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Le diagramme de Gant correspondant à l'algo SRTF:



Les mesures de performances sont:

- Temps d'attente moyen= (9+1+0+2)/4=3
- Temps de rotation moyen=(16+5+1+6)/4=7
- Débit= 4/16=0.25

Processus	Temps attente	Temps de rotation
P1	9	16
P2	1	7-2=5
P3	0	5-4=1
P4	2	11-5=6

II.4 Algorithmes d'ordonnancement

7) Round-Robin (RR)=Tourniquet

- Algo préemptif le plus utilisé en pratique.
- A chaque processus est allouée une tranche de temps, appelée quantum (généralement entre 10 et 100 ms.), pour s'exécuter.
- 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 (en fin de file d'attente).

✓ Avantages:

- Pas de monopolisation de l'UCT, équitable
- Pas de famine
- Bon temps de réponse

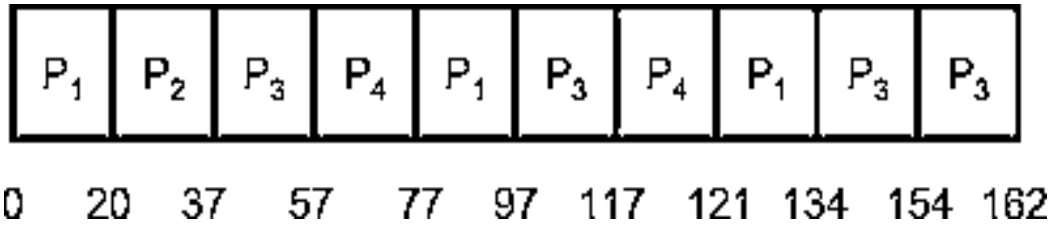
↓ Inconvénients:

- Temps d'attente moyen en général important.
- Influence de la valeur du quantum, difficile à déterminer (grand : FIFO, petit: perte de temps dans les changements de contexte)

7) Round-Robin (RR)=Tourniquet - Exemple

Soit les processus P1 , P2 , P3, P4. Le diagramme de Gant correspondant à l'algo RR avec quantum = 20

Processus	Cycle UCT
P1	53
P2	17
P3	68
P4	24



Processus	Temps de rotation
P1	134
P2	37
P3	162
P4	121

Les mesures de performances sont:

- Utilisation UCT= 162/162=100%
- Temps de rotation moyen=(134+37+162+121)/4=113.5
- Débit= 4/162=0.025
- Temps de rotation et temps d'attente moyens sont beaucoup plus élevés que les algos précédents, mais meilleur temps de réponse moyen.
- Le RR suppose que tous les processus sont aussi importants, mais en pratique, ce n'est pas le cas (processus vidéo plus important que processus qui affiche l'heure) => **Algo HPF**

8) Comparaison

Soit les processus P1 , P2 , P3, P4.

Processus	Arrivée	Cycle UCT
P1	0	7
P2	1	4
P3	2	7
P4	3	5

Remarques:
- SRTF fournit le meilleur temps d'attente moyen.
- RR avec petit quantum augmente le temps d'attente moyen.

Temps d'attente:

Processus	FCFS	SJF	SRTF	RR (2)	RR(5)	RR(10)
P1	0	0	9	14	14	0
P2	6	6	0	7	4	6
P3	9	14	14	14	14	9
P4	15	8	2	14	11	15
Moyenne	**7.5**	**7**	**6.25**	**12.25**	**10.75**	**7.5**

II.4 Algorithmes d'ordonnancement

8) HPF (Highest Priority First ou haute priorité d'abord)

- Affectation d'une priorité à chaque processus (souvent nombre entier, avec 0 la plus haute).L'UCT est donnée au processus prêt avec la plus haute priorité.

•Peut être préemptif ou non. Quand un nouveau processus arrive:

- **Cas préemptif:** comparer sa priorité à celle du processus en cours d'exécution. L'UCT est alors réquisitionnée en cas de plus haute priorité. Le processus sorti sera remis en tête de la liste d'attente prêt correspondant à sa priorité.
- **Cas non préemptif:** placer le processus dans la file d'attente FIFO correspondant à sa priorité. Une fois qu'il sera élu, il ne sera pas interrompu par l'ordonnanceur.

•**N.B:** Il y a une file d'attente prêt pour chaque niveau de priorité. HPF choisit toujours dans la file la plus prioritaire.

✓**Avantages:**

- Simple, Prise en compte de l'importance des processus

↓**Inconvénients:**

- Risque de famine pour les processus moins prioritaire.
- **Solution:** Augmenter la priorité des processus qui attendent depuis longtemps.

II.4 Algorithmes d'ordonnancement

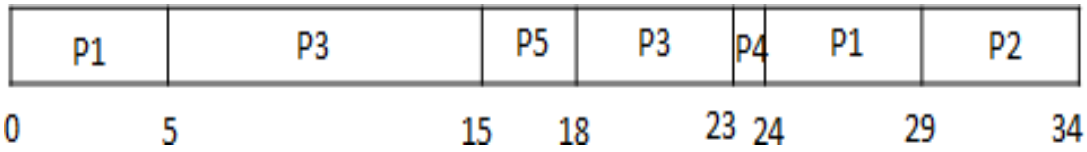
8) HPF (Highest Priority First ou haute priorité d'abord) - Exemple

Soit les processus P1 , P2 , P3, P4, P5. Le diagramme de Gant correspondant à l'algo HPF:

Processus	Arrivée	Cycle UCT	Priorité
P1	0	10	3
P2	2	5	7
P3	5	15	2
P4	10	1	2
P5	15	3	1

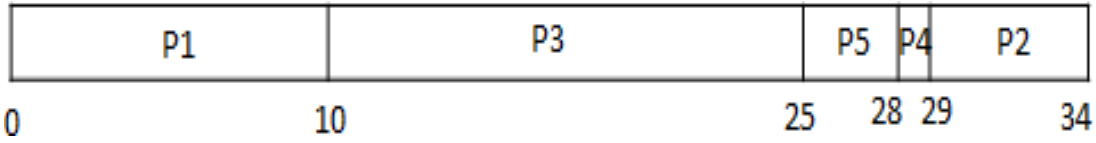
- **préemptif:** Temps de rotation moyen=

(29+32+18+14+3)/5 =19.2



- **Non préemptif:** Temps de rotation moyen=

(10+32+20+19+13)/5 =18.8



Processus	Temps de rotation préemptif	Temps rotation non préemptif
P1	29	10
P2	34-2=32	34-2=32
P3	23-5=18	25-5=20
P4	24-10=14	29-10=19
P5	18-15=3	28-15=13

- Avec HPF non préemptif, les processus prioritaire risquent d'attendre plus longtemps.

1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus
6. Création de processus

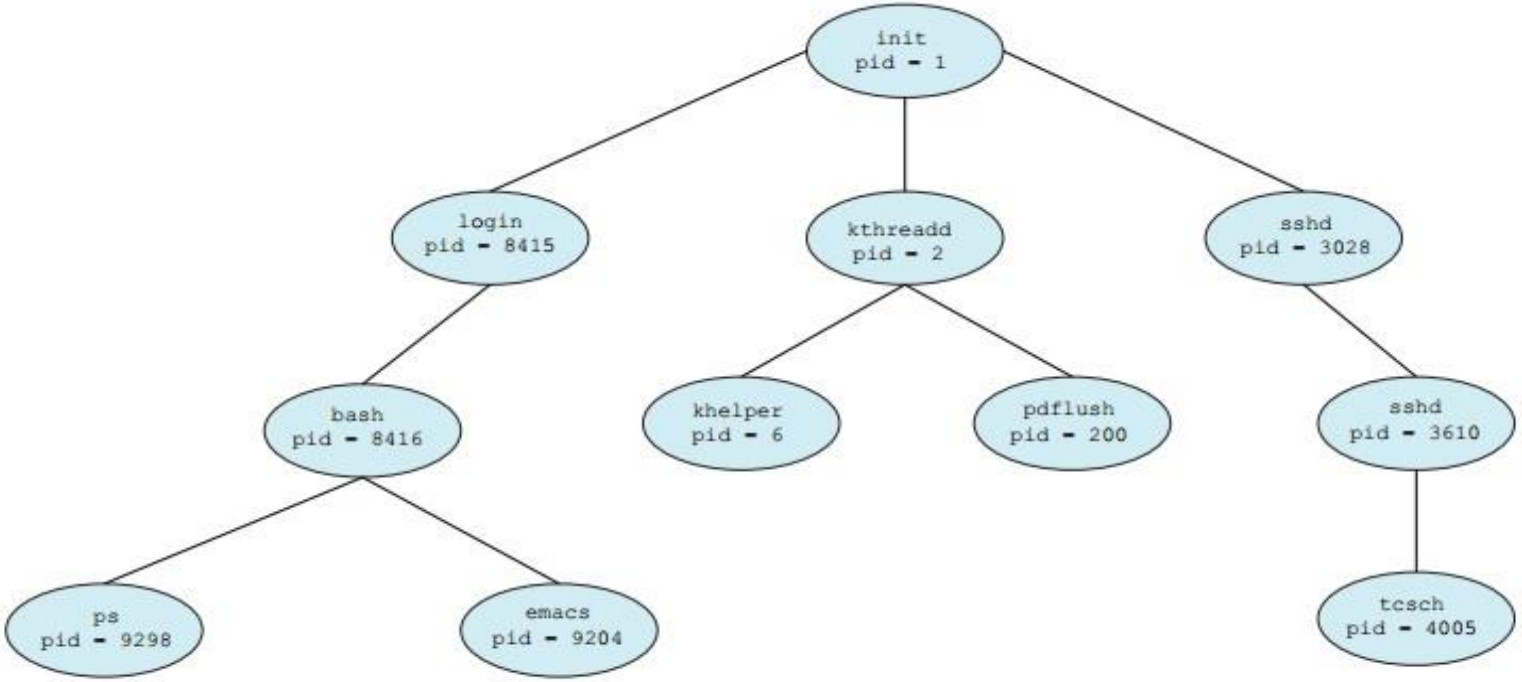
II.5 Superviseur des processus

1) Création de processus

•Pendant son exécution, un processus (**père**) peut créer de nouveaux processus (**fils**), qui peuvent à leur tour créer des fils, formant ainsi une ***arborescence*** de processus.

•La majorité des SE identifient les processus par un **numéro unique** (process id ou **PID**) et font référence au père par son PID noté **PPID**.

•Pour le SE Linux par exemple, lorsque le SE démarre, il crée un **processus initial** nommé **init** (avec **pid=1**). Si un utilisateur souhaite se connecter, **init** crée le processus login(pid 8415).



II.5 Superviseur des processus

1) Création de processus

La majorité des SE proposent des mécanismes de création de processus fils à partir du processus père:

•**Fork():**

- Le SE crée un **nouveau PCB** pour le fils et y copie **les mêmes éléments** du PCB du père.
- Père et fils **continuent** leur exécution à partir de l'instruction suivant le fork(), puisque les deux processus ont **les mêmes valeurs des registres** dans leur PCB.
- La **valeur de retour de fork()** est **0 pour le fils**, alors que pour **le père**, elle correspond au **PID du fils**. Avec un code qui teste cette valeur, le père et le fils peuvent être dirigés vers différents segments du code.

•**Exec():**

- Typiquement, après l'appel système fork(), le fils utilise l'appel système exec() pour **remplacer les éléments de son PCB** par un nouveau programme.
- De cette manière, les deux processus peuvent communiquer facilement et effectuer chacun leur exécution.
- Le père peut se placer en attente jusqu'à la terminaison de son fils

2) Terminaison de processus

3 appels systèmes sont liés à la terminaison des processus:

•**wait()** : Permet à un processus **père d'attendre jusqu'à ce que son fils** termine. Il retourne l'identifiant du processus fils et son état de terminaison.

•**exit() :** Permet au processus de **finir volontairement** son exécution car il a terminé ses instructions et retourne son état de terminaison.

•**kill():** Permet de **forcer l'arrêt** d'un autre processus. Habituellement, un tel appel ne peut être invoqué que par le père du processus qui doit être terminé.

• Un père peut mettre fin à l'exécution de l'un de ses enfants pour diverses raisons:

- Le fils a dépassé son utilisation de certaines des ressources allouées.
- La tâche assignée au fils n'est plus nécessaire.
- Le parent sort et le SE ne permet pas au fils de continuer si son parent se termine.

Remarque: Sur certains SE, lorsqu'un père prend fin, tous ses fils se terminent automatiquement. Puis, les fils de ces derniers s'achèvent à leur tour, ce mécanisme est connu sous le nom de **terminaison en cascade**.

2) Terminaison de processus – le processus zombie

•Lorsqu'un processus se termine, ses ressources sont désaffectées par le SE. Cependant, **son PCB doit rester dans la table des processus** jusqu'à ce que son père appelle **wait().**

•Un processus qui s'est terminé, mais dont le père n'a pas encore appelé **wait(),** est connu sous le nom de **processus zombie**.

•Tous les processus passent à cet état lorsqu'ils terminent, mais généralement ils n'y restent que brièvement. Une fois que le père appelle **wait(),** le PID du zombie et son PCB sont libérés.

•Si un père n'a pas appelé **wait ()** et a terminé, ses processus fils vont devenir **orphelins.**

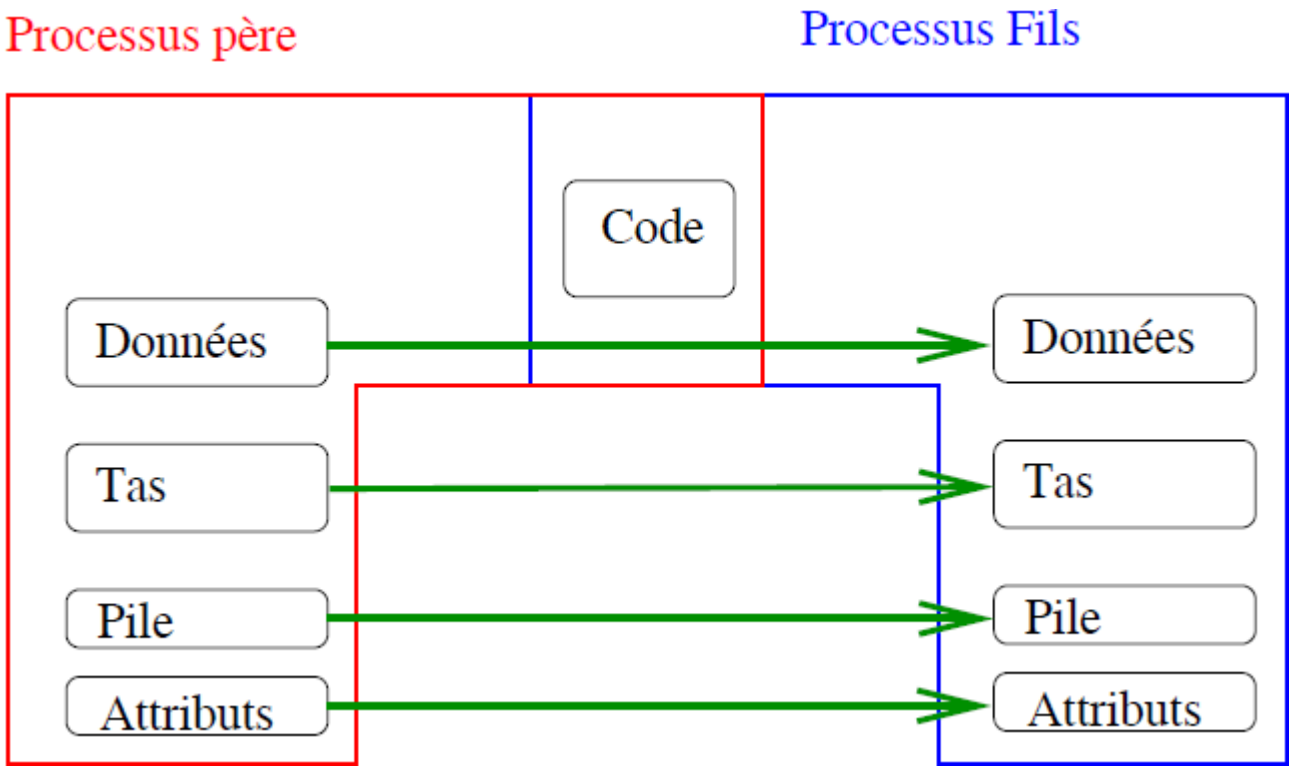
•Linux et UNIX attribuent le processus d'initialisation (init) en tant que nouveau père des processus orphelins. Init invoque périodiquement **wait()**, ce qui permet de **collecter** le statut de sortie de tout processus orphelin et de libérer son PID et son PCB.

1. Introduction et Définitions
2. Niveaux d'ordonnancement des processus
3. Etats des processus
4. Algorithmes d'ordonnancement
5. Superviseur des processus
6. Création de processus

II.6 Création de processus

1) La primitive système fork()

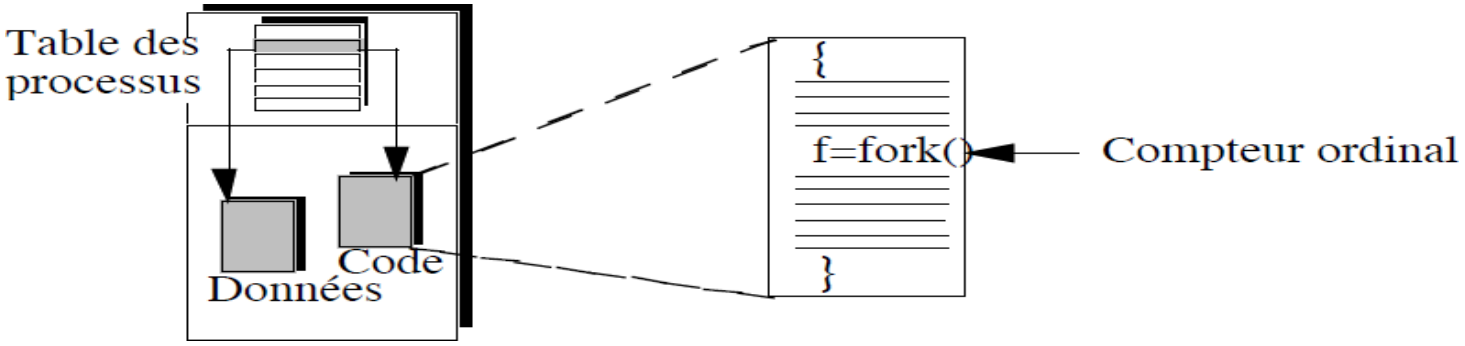
- Recopie les données et les attributs du processus père vers son processus fils auquel il attribue un nouveau pid.
- Le fils continue son exécution à partir de cette primitive



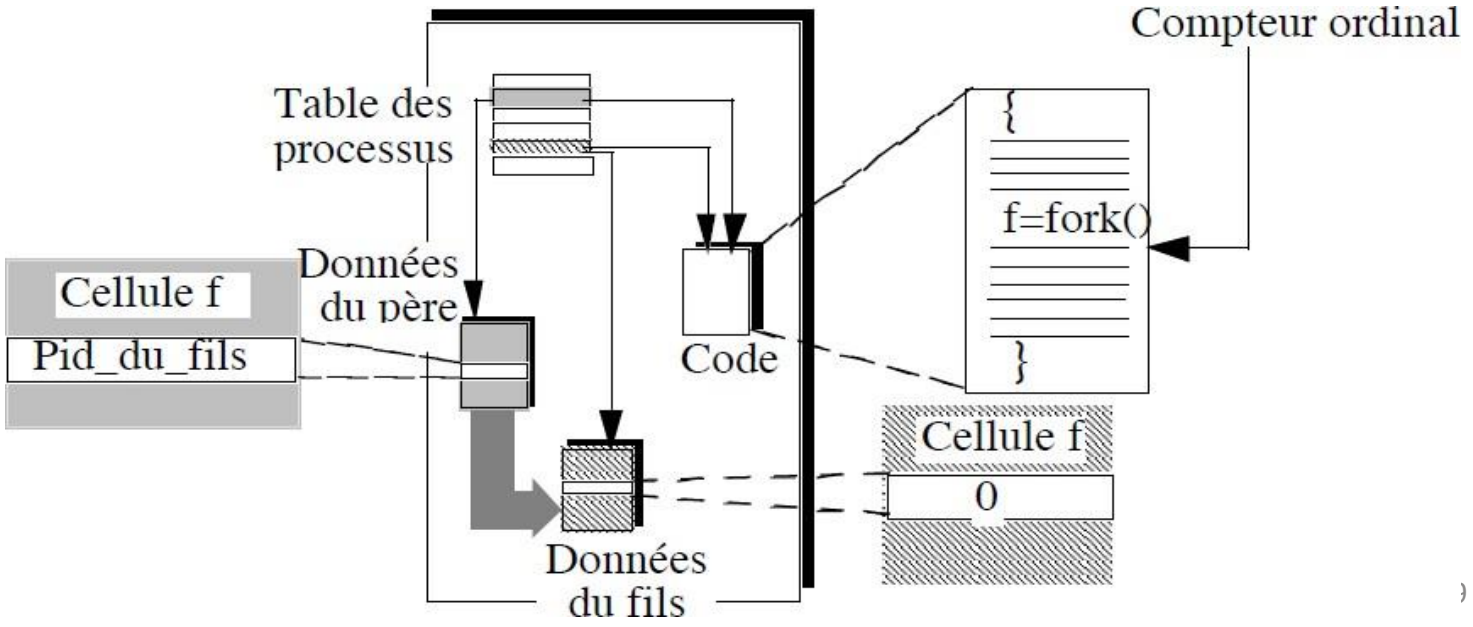
II.6 Création de processus

1) La primitive système fork() – avant vs après

Avant fork()



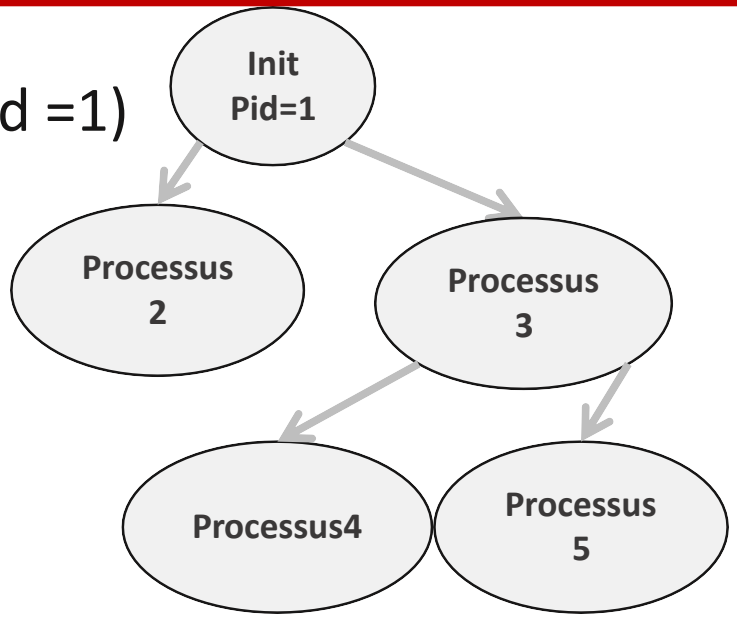
Après fork()



II.6 Création de processus

1) La primitive système fork() – Arborescence

- L'itération de fork() conduit à une arborescence à partir du processus init (pid =1)



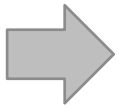
- Différencier le père du fils : Code de retour du fork()
  - Dans le père : le fork retourne le pid du processus fils
  - Dans le fils : le fork retourne 0

- Pourquoi ?
  - Le fils peut connaitre le pid de son père avec getppid().
  - Alors que pour le père, le seul moyen pour connaitre le pid du processus fils est le retour du fork.

II.6 Création de processus

1) La primitive système fork() – porté du code

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main () {
int f;
f = fork();
printf (" Valeur retournee par la fonction fork: %d\n", (int)f);
printf (" Je suis le processus numero %d\n", (int)getpid());
return 0;
}
```



```
Valeur retournee par la fonction fork: 3952
Je suis le processus numero 3951
Valeur retournee par la fonction fork: 0
Je suis le processus numero 3952
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(){
int k ;
printf("Je suis seul au monde\n");
k=fork();
if (k==-1) { printf("Erreur fork()");
}

if (k== 0) {
printf("Je suis le processus fils\n");
}
else
printf("Je suis le processus pere\n");
printf("Et la qui suis-je %d\n",getpid());
}
```

```
Je suis seul au monde
Je suis le processus pere
Et la qui suis-je 2850
Je suis le processus fils
Et la qui suis-je 2851
```