Centre Universitaire de Mila

3 ème année licence LMD Informatique

Module : Systèmes d'exploitation 2

Bessouf Hakim

SYSTÈMES D'EXPLOITATION II

Objectifs de l’enseignement

Introduire la problématique du parallélisme dans les systèmes d’exploitation et étudier la mise en œuvre des mécanismes de synchronisation, de communication dans l’environnement centralisé

Contenu de la matière

1. Notions de parallélisme, de coopération et de compétition
2. Synchronisation
3. Communication
4. Interblocage
5. Étude de cas : système Unix

CHAPITRE 1:

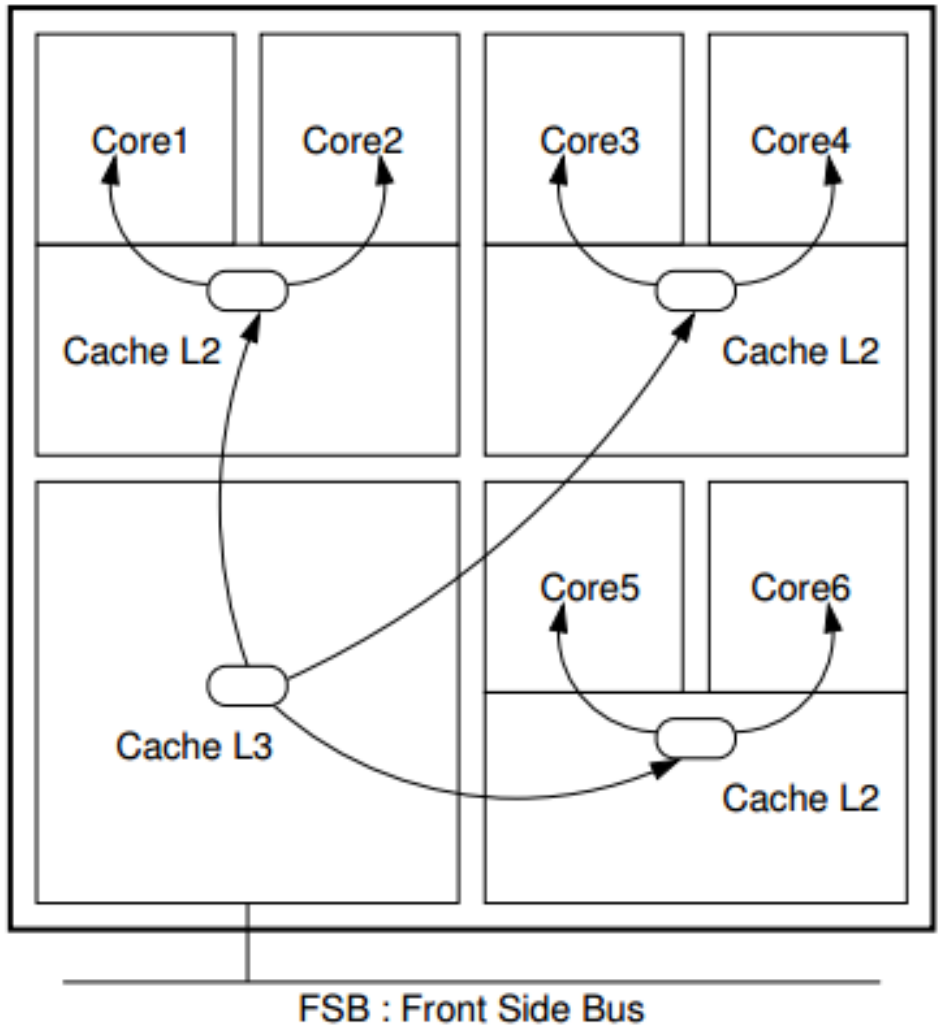
Notions de parallélisme, de coopération et de compétition

1. Systèmes de tâches, outils d'expressions
2. Déterminisme et parallélisme maximal
3. Thread

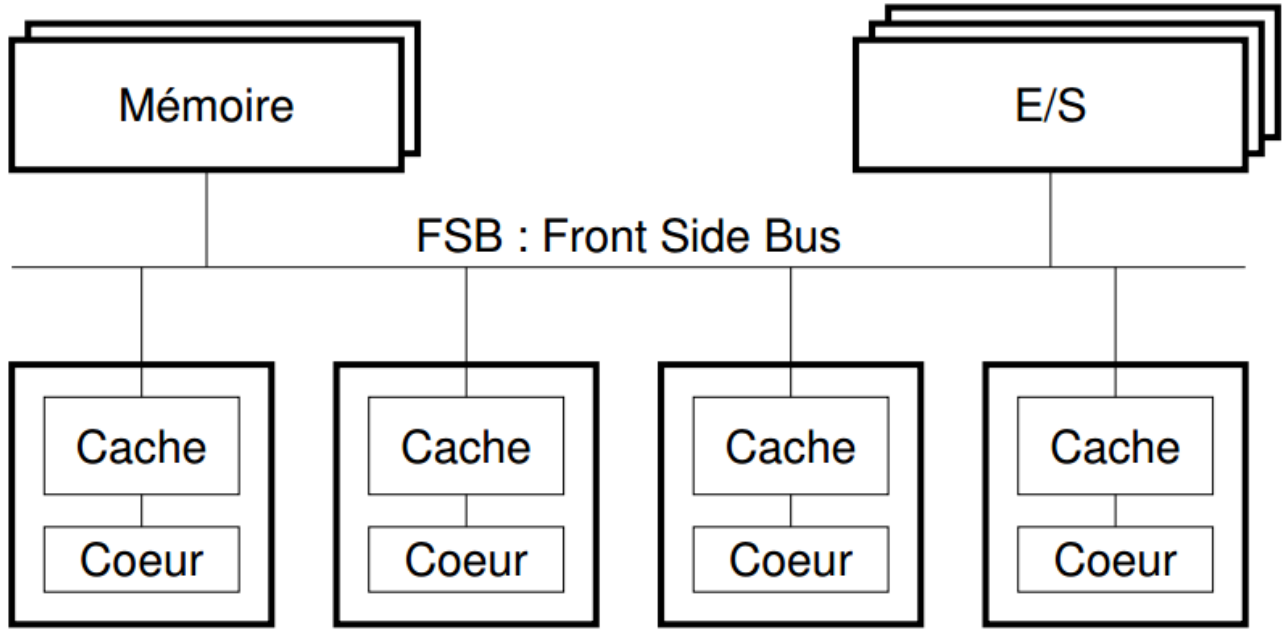
Introduction

- Taxonomie de Flynn:

	Flux d'instructions	
Flux de données	**SISD :** correspond à l'architecture de Von Neumann	**MISD :** Cette architecture n'a pas beaucoup d'applications.
	SIMD :(parallélisme de données) Les machines vectorielles font partie de cette catégorie	**MIMD :** C'est l'architecture parallèle la plus intuitif et la plus utilisé. Architectures : à mémoire partagé, réparties



Processeurs multi-cœurs avec mémoires cache commune



Une machine multiprocesseurs, chaque processeur a un seul cœur

- **Remarque**
  Dans les exemple précédents on parle de processeurs avec plusieurs **cœurs physiques**. Il existe aussi des processeurs avec des **cœurs logiques** dans ces derniers certaine parties matériel du processeur sont dupliqués pour accélérer la vitesse de commutation entre les processus. Un processeur qui est équipé de deux cœurs logiques par exemple ne peut a un instant donnée exécuter qu'une seule instruction mais peut traiter deux processus en même temps et commuter rapidement entre eux.

Le processus

- La tache principale des systèmes d'exploitation est de gérer les programmes. Un **programme en exécution** dans le système est appelé **processus.**
- Un processus est compose du code exécutable et du contexte :

– **Le code exécutable** : C'est le code produit par l’éditeur de liens.

– **Le contexte** : Il décrit l'environnement du programme en exécution (le compteur ordinal , le mots d’état PSW, les registres généraux, ..etc.)

Le processus

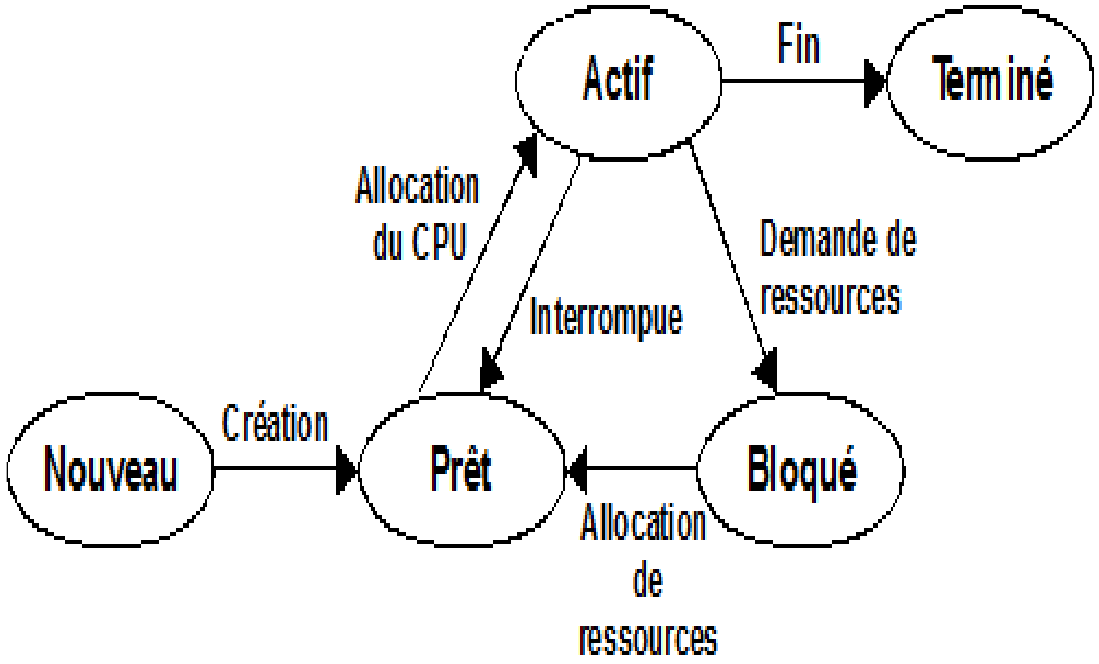
- Un programme n'est pas un processus, le programme est une entité **passive** comme n'importe quel autre fichier stocke sur disque tandis que le processus est une entité **active**.
- Si par exemple deux utilisateurs exécute deux copies du même programme on aura deux processus séparés.

Le processus

- **Exemple :**
- Considérons la maman qui prépare un gâteau dont elle a une recette et, dispose dans sa cuisine de farine, œufs, sucre etc.
- Dans cette exemple la **recette** représente le **programme** (une suite d'instructions), la **maman** représente le **processeur** (CPU) et les ingrédients sont les donnes a fournir au programme. Le **processus** est l'**activité de la maman** qui lie la recette, trouve les ingrédients nécessaires et fait cuire le gâteau.

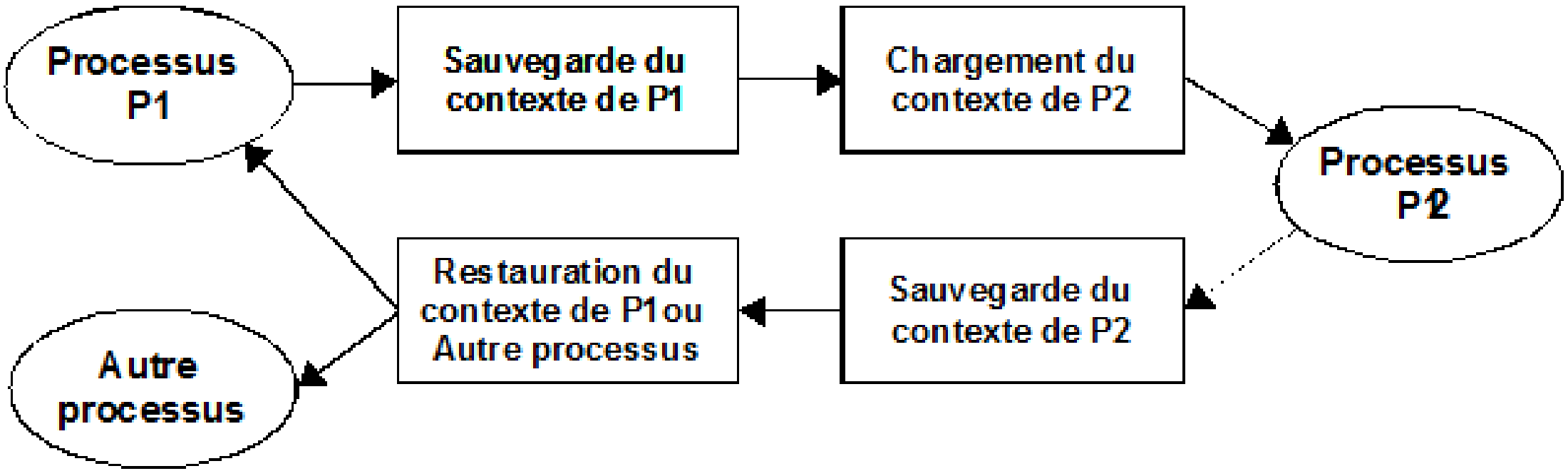
Les différents états du processus

- **Prêt** : le processus attend la libération du processeur pour s'exécuter.
- **Actif** (en exécution, élu) : Le processus est en cours d'exécution.
- **Bloqué** (en attente) : le processus attend un évènement pour pouvoir continuer (exemple : fin d'une E/S, allocation de mémoire .. etc.)
- **Nouveau** : le processus est en cours de création
- **Terminé** : le processus a terminé sont exécution.



Les transitions d'états d'un processus.

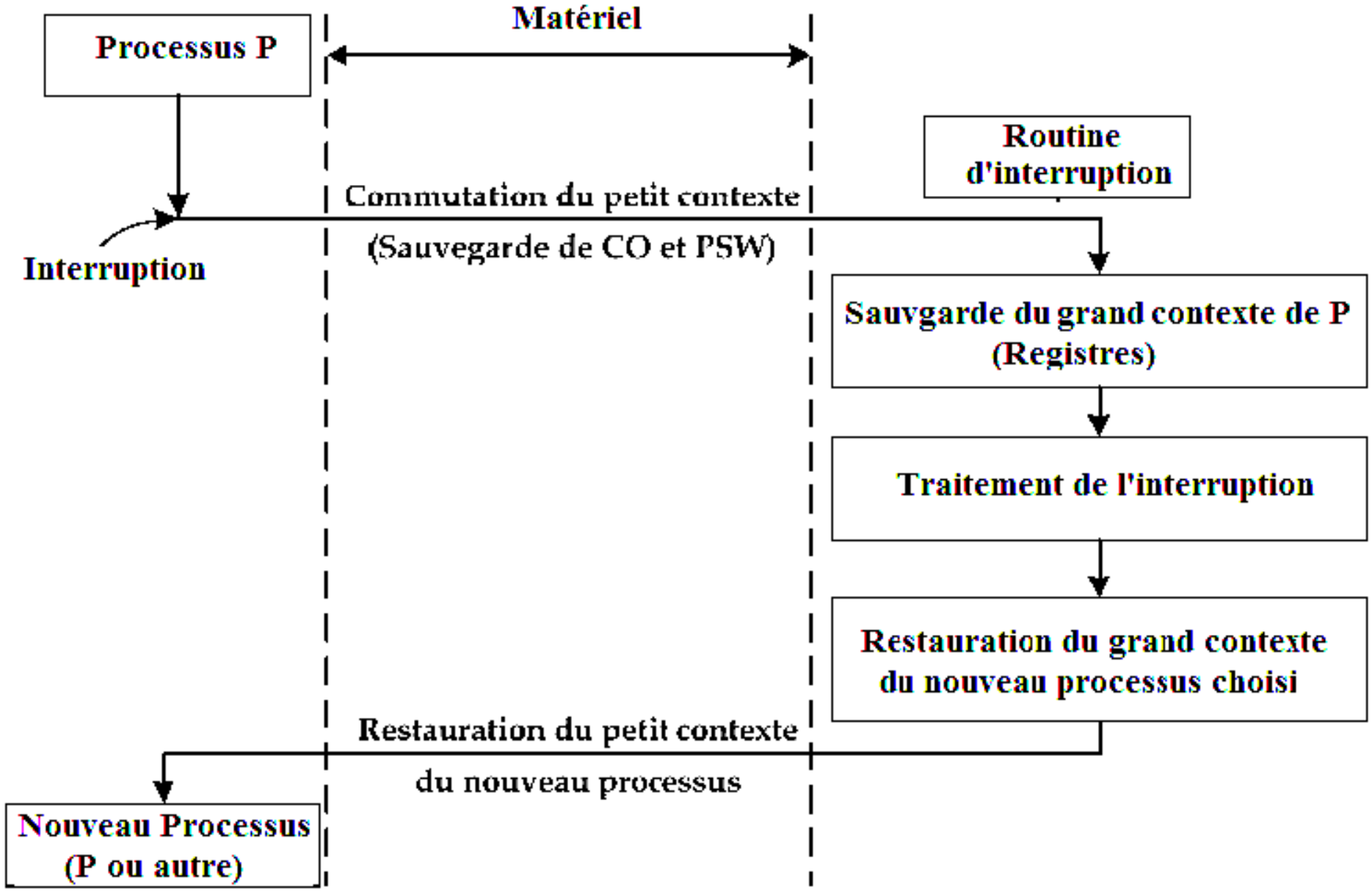
La commutation de contexte



La commutation de contexte

La commutation de contexte

- Dans les systèmes multiprogrammes plusieurs processus sont exécutés en parallèle (**pseudo-parallélisme**).
- Le passage d'un processus a un autre nécessite une opération de **sauvegarde du contexte** du processus arrêté et de **chargement du contexte** du nouveau processus. Cette opération est appelé **commutation de contexte**.
- Le **petit contexte** comprend le conteur ordinale (CO) et le mot d'état (PSW),
- Le **grand contexte** comprend les registres généraux et la pile.


Matériel
Processus P
Routine d'interruption
Commutation du petit contexte
(Sauvegarde de CO et PSW)
Interruption
Sauvgarde du grand contexte de P (Registres)
Traitement de l'interruption
Restauration du grand contexte du nouveau processus choisi
Restauration du petit contexte
du nouveau processus
Nouveau Processus (P ou autre)

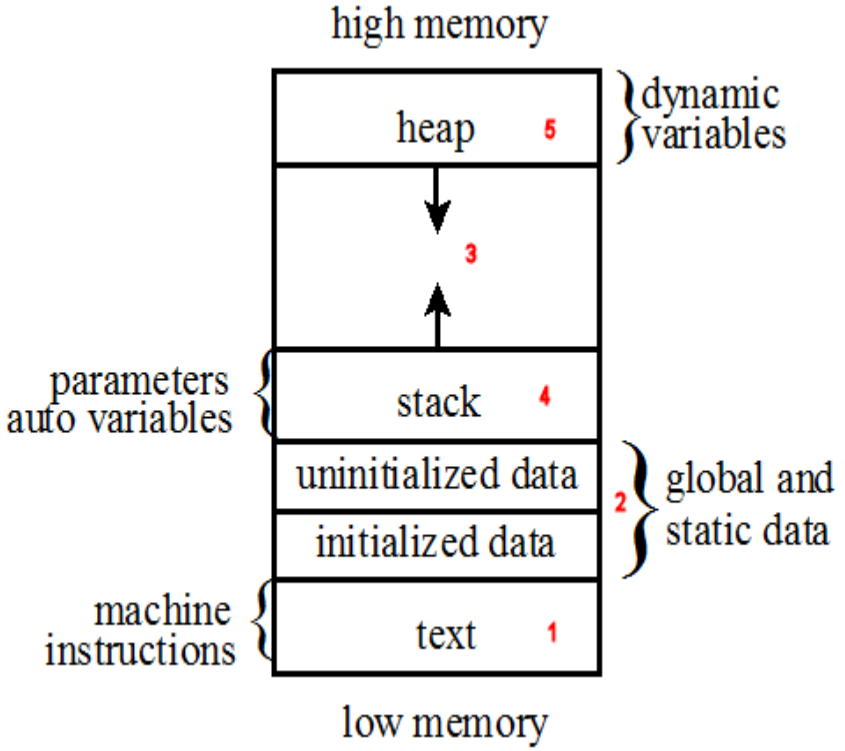
- L’espace mémoire alloué à un processus est appelé **image mémoire** (**Memory Map**) du processus , Il est divisé en plusieurs parties :

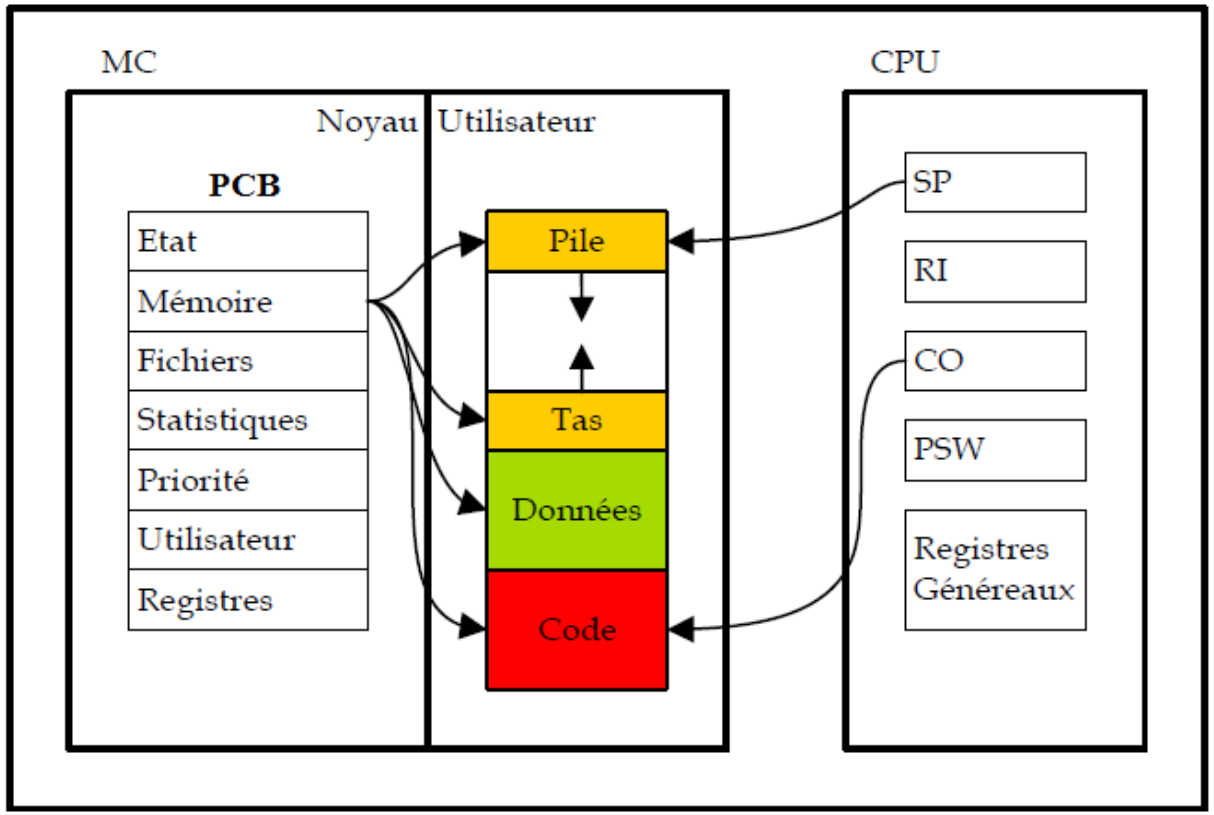
Le Code : C'est le code des instructions du programme à exécuter. L’accès à cette zone mémoire se fait en **lecture seulement** (**ReadOnly**).

Les Données (Data) : Il contient l’ensemble des constantes et variables déclarées**.**

La Pile (Stack) : Elle permet de stocker: les valeurs des registres, Variables locales, paramètres de fonctions et les adresses de retour des fonctions.

Le Tas (Heap) : C'est une zone à partir de laquelle l’espace peut être alloué dynamiquement **en cours d’exécution du processus**, en utilisant par exemple les fonctions **new** et **malloc**





PCB

Identité : - Nom du processus
Gestion du processus : - Compteur ordinal - mot d'état - Registres - Temps de traitement utilisé - processus fils - … etc
Gestion de la mémoire : - Limites mémoire - ..etc
Gestion des fichier : - Répertoire racine - Fichiers utilisés - ..etc

- Pour gérer les processus le système d'exploitation utilise une table en mémoire centrale appelé **table des processus** ou **bloc de contrôle des processus** (PCB). Cette table contient plusieurs informations concernant chaque processus (conteurs ordinal, mots d'état, pointeur de pile, ressources utilisés ..etc)

- **Manipulation des processus**

Le système d'exploitation utilise plusieurs primitives (procédures) pour manipuler les processus :

Création : utilisé par un processus pour créer un autre processus, Le premier processus est appelé **processus père**, le deuxième processus est appelé p**rocessus fils**. Un processus père peut avoir plusieurs processus fils.

Activation : Elle permet de mettre un processus prêt dans l'état actif,

Suspension : Elle permet de suspendre un processus,

Destruction : Elle permet de terminer un processus et de libérer toutes les ressources qu'il utilise (mémoires, fichiers ..etc).

l'ordonnanceur des processus (scheduler)

- L'ordonnanceur est un module du système d'exploitation qui **partage le processeur** central (CPU) entre plusieurs processus en attente d'exécution (dans l'état prêt), suivant une **politique** définit a l'avance par les concepteurs du système d'exploitation.
- L'ordonnanceur donc est un programme qui **choisit le prochain processus a exécuter**.
- Le choix du processus a exécuter est appelé **ordonnancement** (**scheduling**).

Objectifs de l'ordonnanceur (scheduler)

- Les principaux objectifs de l'ordonnanceur sont :
- **L'équité** : l'ordonnanceur doit allouer le processeur central aux processus de même priorité de manière **juste** et **équitable**.
- **Utilisation maximal des ressources** :
  l'ordonnanceur doit assurer une utilisation maximal des ressources de la machine (E/S, mémoire, ..etc)

Objectifs de l'ordonnanceur (scheduler)

- Dans **les systèmes par lots :**
  ➢ L'ordonnanceur doit exécuter un maximum de travaux par heur,
  ➢ L'ordonnanceur doit réduire le délais entre la soumission du travail et la sortie des résultats,
  ➢ L'ordonnanceur doit utiliser au maximum le processeur central,
- Dans **les systèmes interactifs** (il y a interaction entre l'utilisateur et la machine) **:**
  ➢ L'ordonnanceur doit minimisé le temps de réponse, il doit donc répondre rapidement aux requêtes des utilisateurs,
- Dans **les systèmes temps réel** (il y a des contraintes temporelles comme dans centrales nucléaire, usines ..etc) :
  ➢ L'ordonnanceur doit respecter les délais.

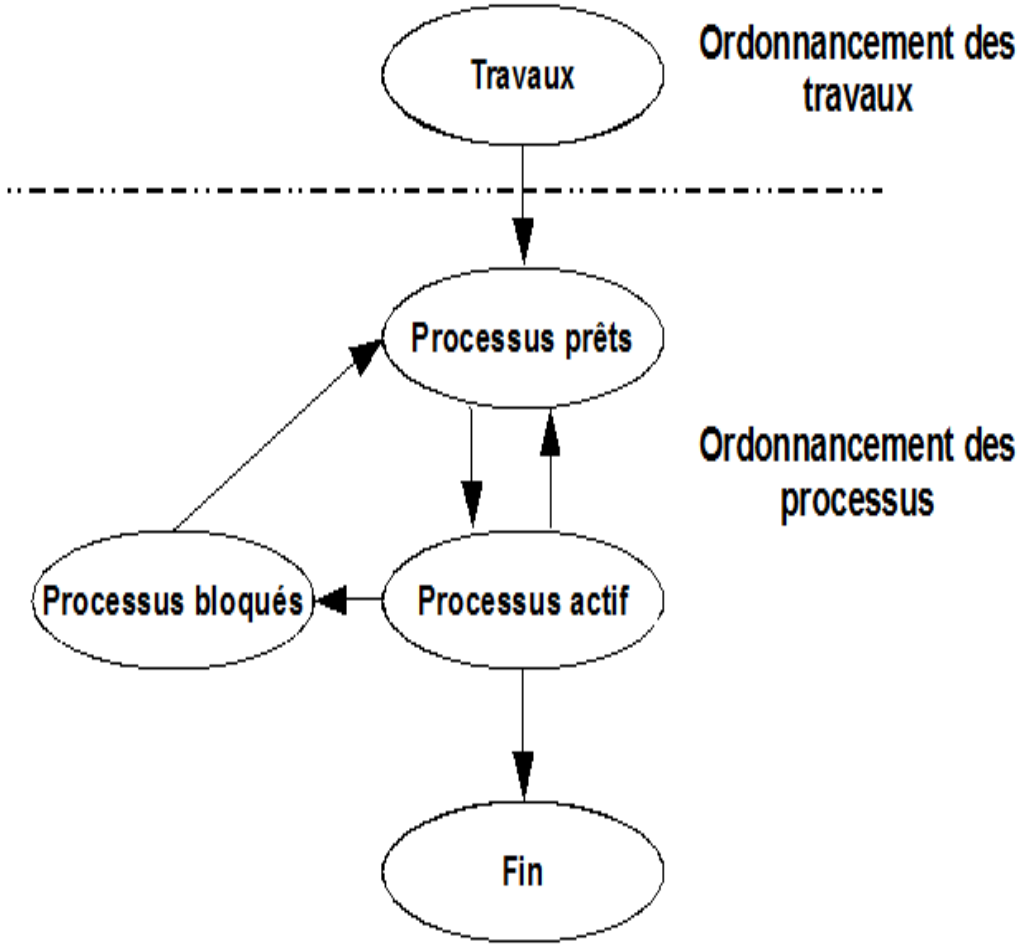
Les critères d'ordonnancement (scheduling)

- **Disponibilité des ressources: l**es programmes sont ordonnancés suivant les ressources qu'il demandes, et suivant les ressource disponible dans la machine.
- **Classe des programmes: l**es programmes sont ordonnancés suivant leurs type (interactif, temps réel, orienté calcul, orienté E/S ..etc).
- **Ordonnancement avec ou sans préemption:** Dans les politique d'ordonnancement **sans préemption** un programme s'exécute jusqu'à ce qu'il se bloque (pour attendre une E/S), ou il se termine. Par contre dans les politiques d'ordonnancement **avec préemption** l'ordonnanceur peut suspendre un programme en exécution et exécuter un autre programme.
- **Ordonnancement avec ou sans priorité:** Les programmes peuvent ou non avoir des priorités d'exécution. Par exemple un programme système peut être prioritaire qu'un programme utilisateur.

Niveaux d'ordonnancement

Il existe deux nivaux d'ordonnancement :

- L'**ordonnancement des travaux** permet de choisir le travail a admettre dans le système et créer le processus correspondant.
- L'**ordonnancement des processus** permet de choisit le processus a exécuter.



1. Politiques d'ordonnancement des travaux

1.1 Politique d'ordonnancement sans préemption

La politique du premier arrivé, premier servie (FCFC : First Come First Served, FIFO : First In First Out) : Dans cette politique le travail qui arrive le premier est servie le premier. Cette politique *désavantage les processus courts*.

La politique du travail le plus court d'abord (SJF : short job first): Dans cette politique le travail le plus court est servie le premier. Cette politiques *favorise les travaux cours* mais elle nécessite de *connaître a l'avance le temps d'exécution* des travaux.

L**a politique d'ordonnancement par priorités:** Dans cette politique l'ordonnanceur exécute le travail qui a la plus grade priorité. La priorité des programme dans ce cas est *fixé a l'avance*. l'ordonnancement par priorité peur être préemptif ou non préemptif.

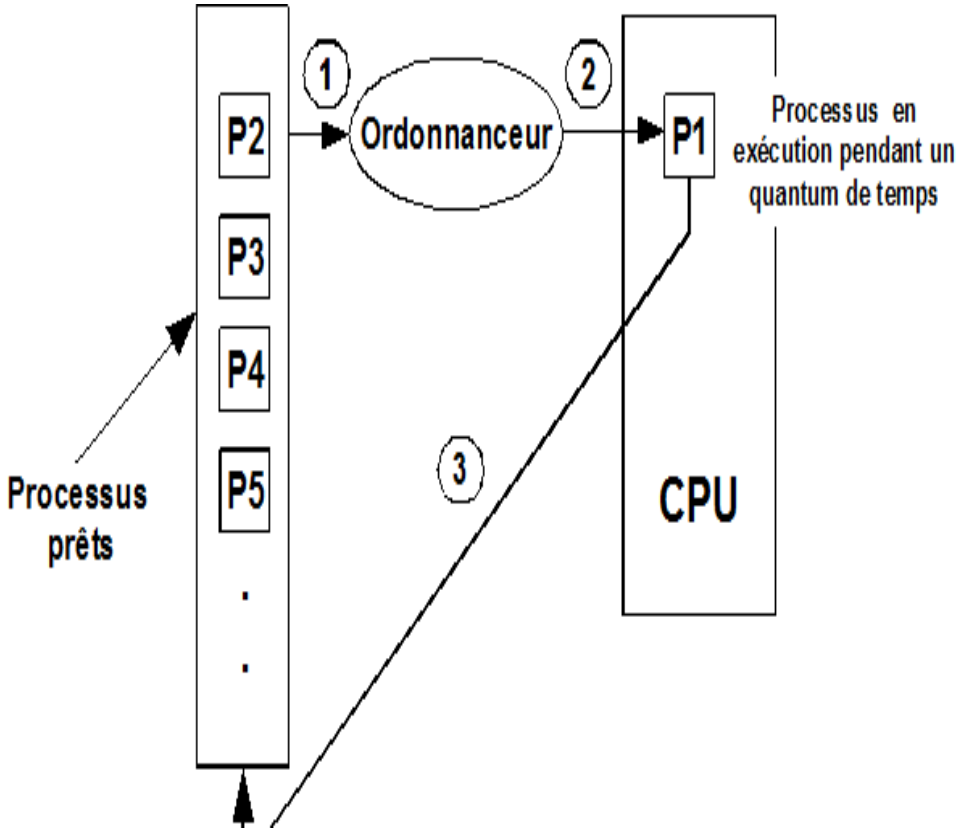
1. Politiques d'ordonnancement des travaux

1.2 Politique d'ordonnancement avec préemption
La politique du travail avec le plus court temps restant (SRTF : Shortest Remained Time First) : l'ordonnanceur choisit le travail dont le temps restant d'exécution est le plus court. C'est une version préemptive de SJF. Si par exemple un premier travail est en exécution et un nouveau travail arrive, si le temps d'exécution du deuxième travail est inférieur au temps restant au premier travail, le premier travail est arrêté et le deuxième travail est exécuté.

Cette politiques *favorise les travaux cours* mais elle nécessite de *connaître a l'avance le temps d'exécution* des travaux

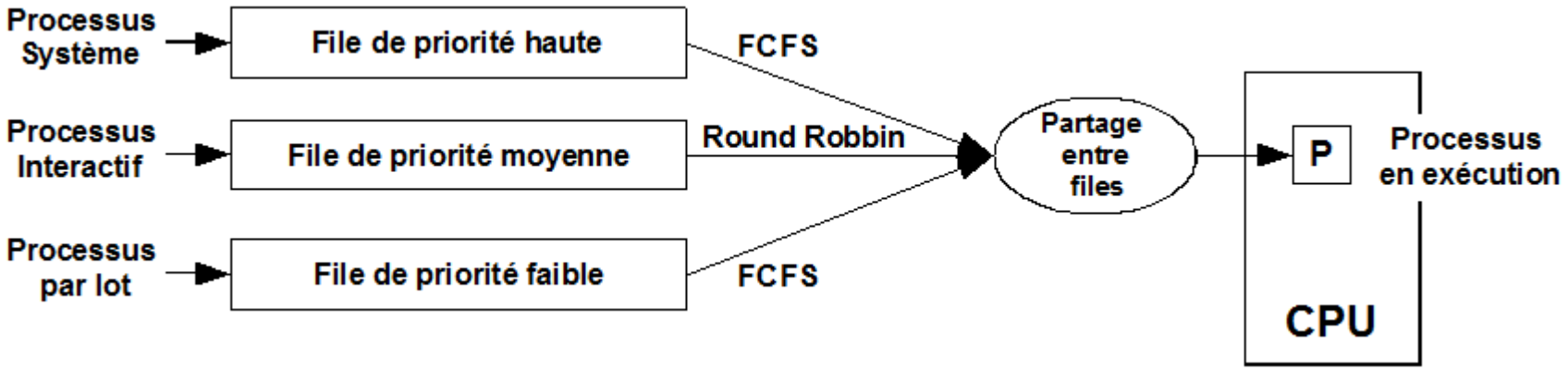
Politiques d'ordonnancement des processus

- **La politique du tourniquet (Round Robbin)** Elle est utilisé dans les systèmes a temps partagé, elle est similaire a la politique FCFS mais on rajoute la réquisition. Dans cette politique chaque processus prêt est exécuté pendant une tranche de temps appelé **quantum**, a la fin de ce temps l'ordonnanceur exécute un autre processus.
- Si le quantum est très grand cette politique ressemblera a la politique FCFS, par contre si le quantum est très petit chaque processus aura l'impression de posséder son propre processeur mais on aura une surcharge due a la **commutation de contexte** des processus.



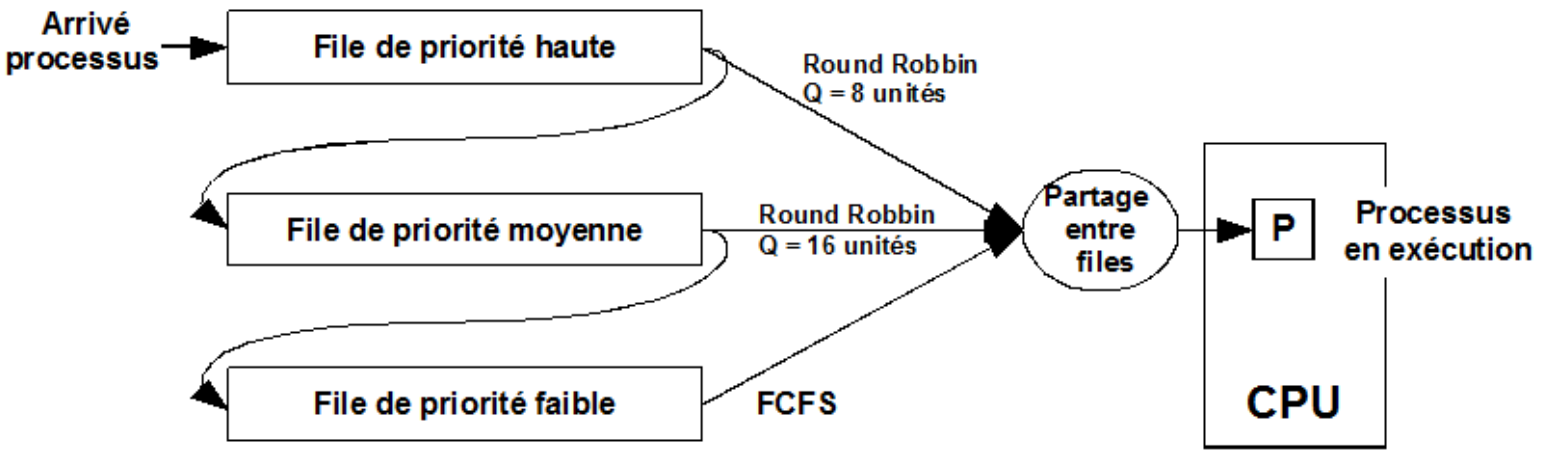
Politiques d'ordonnancement des processus

- **La politique à plusieurs niveaux de queues**
  Dans cette politique les processus sont regroupés en plusieurs fils selon leurs type (interactif, temps réel, ..etc) . Chaque file est ensuite géré par une politique d'ordonnancement qui s'adapte le mieux a cette file. Une autre politique est utilisé pour ordonnancer les files entre eux. Un exemple de cette politique est donné dans la figure suivante :



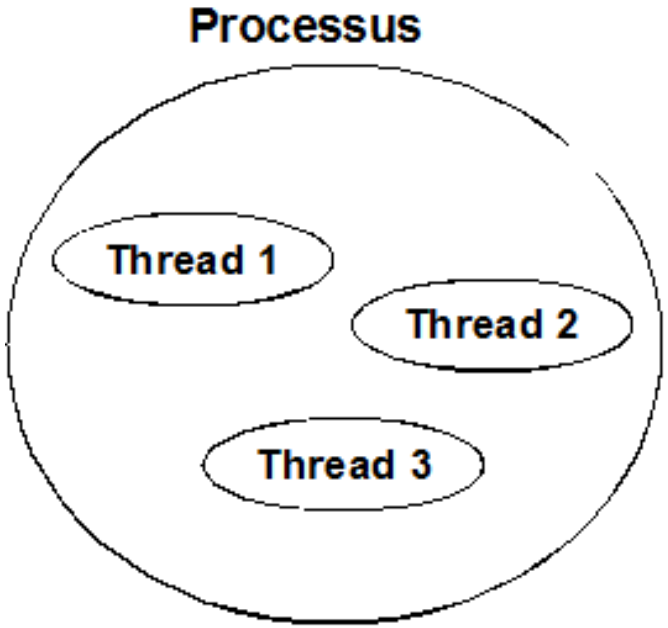
Politiques d'ordonnancement des processus

- **La politique à plusieurs niveaux de queues dépendantes**
  Dans cette politique un processus peut changer de file selon son comportement durant son exécution.



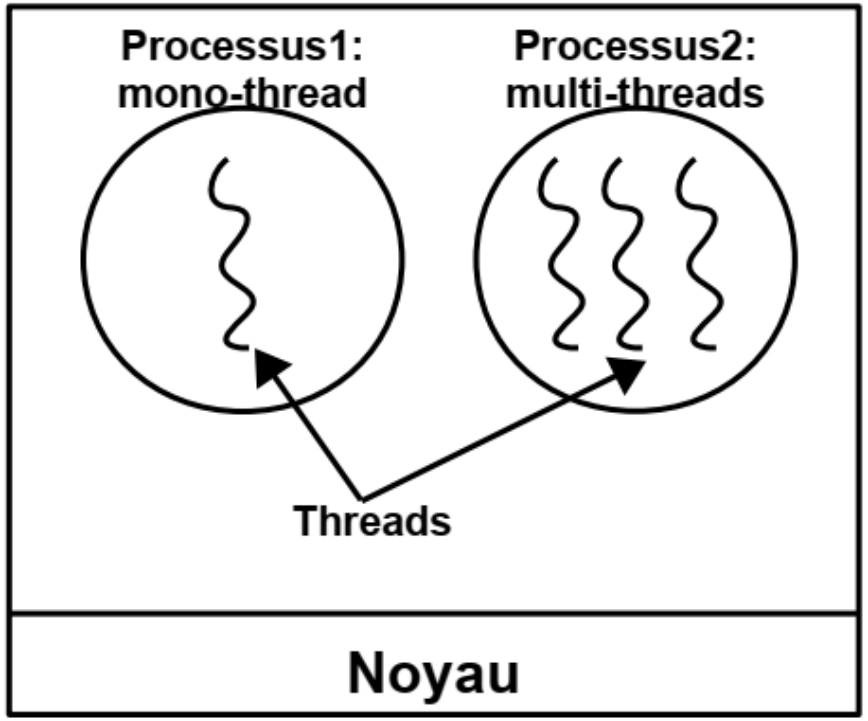
Les Threads (les flots ou les processus légers)

- plusieurs petits processus dans un même processus qui manipulent les données et calcules les résultats. Ces petits processus sont appelés **threads**.
- Ces petits processus partagent le même espace d'adressage et s'exécutent en parallèle (ou en pseudo-parallélisme)
- Par exemple un premier thread peut lire les données, en même temps un deuxième thread calcule les résultats et en même temps un troisième thread affiche ces résultats. Ces trois threads partagent le même espace d'adressage

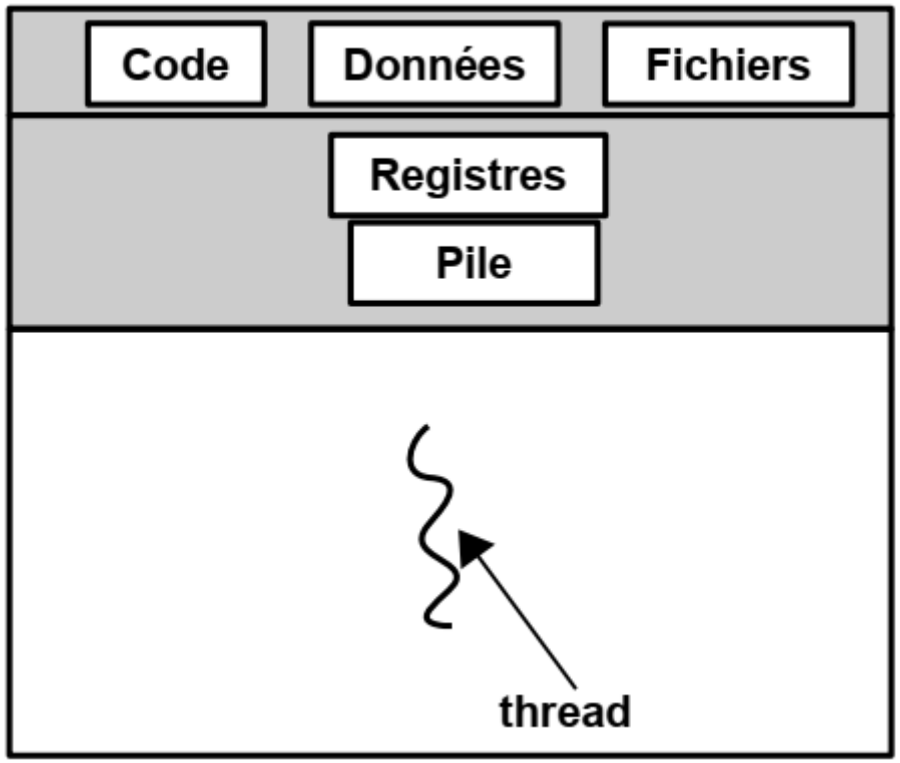


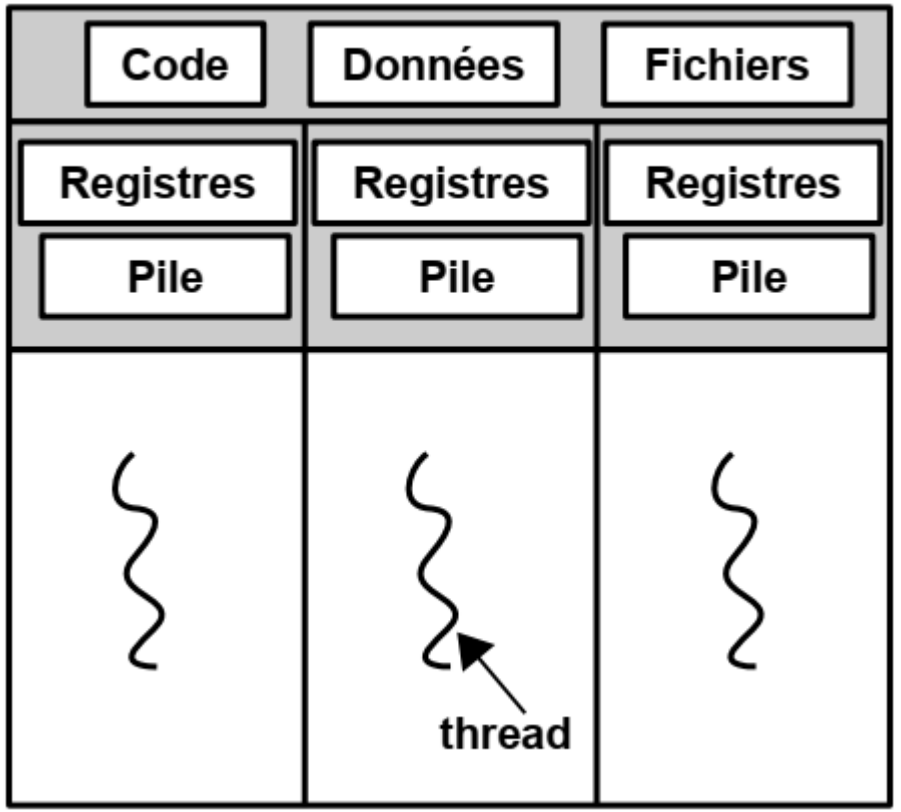
Les Threads (les flots ou les processus légers)

- Par exemple dans une application de vidéo conférence comme skype, un premier thread peut recevoir le flue vidéo d'internet et l'afficher, un deuxième thread peut lire le microphone, un troisième thread peut lire la caméra, et un quatrième thread peut transmettre les messages tapés par l'utilisateur, tous ces threads s'exécutent en parallèle et sont contenu dans un seul processus.



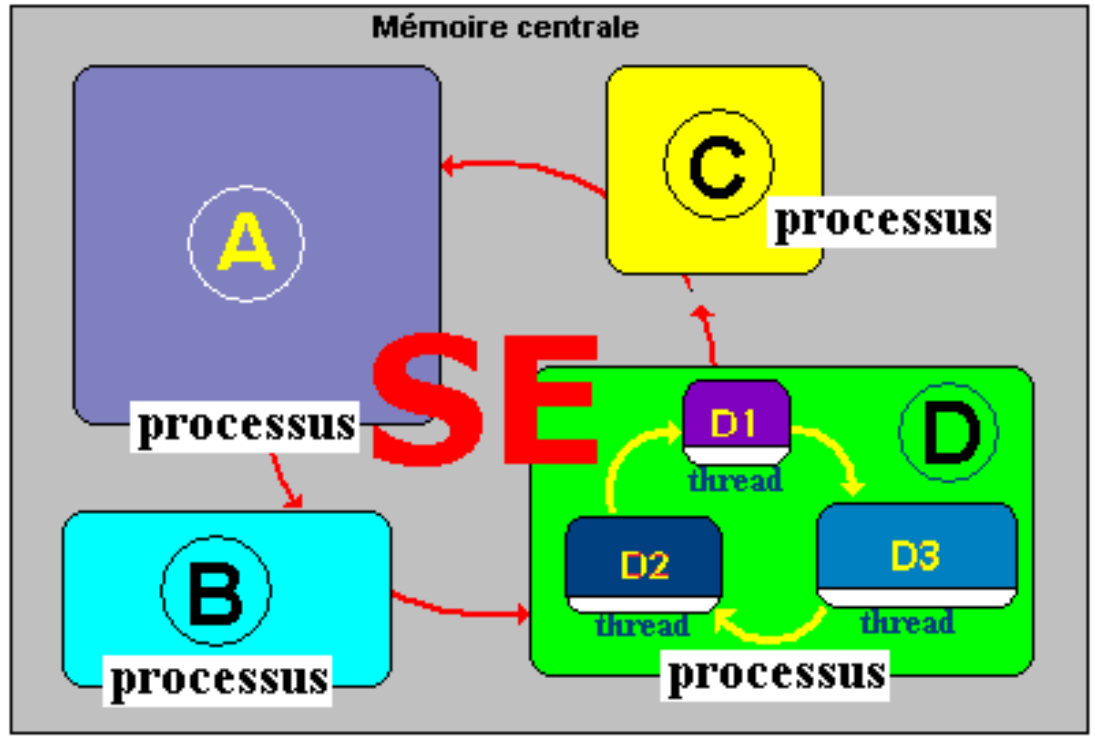
Les Threads (les flots ou les processus légers)

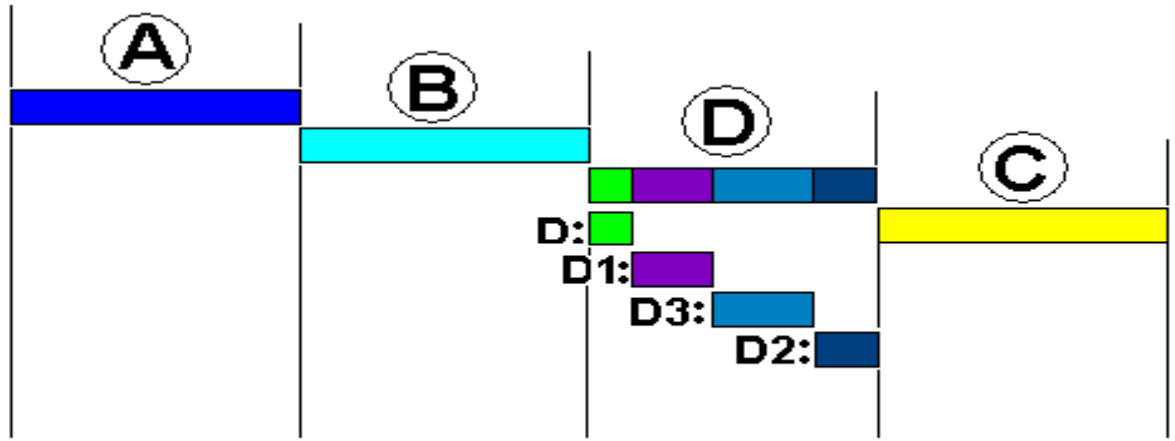




Les Threads (les flots ou les processus légers)

- Les threads peuvent partager le même espace d'adressage et manipuler les mêmes données,
- La création et destruction des threads est très rapide par rapport a la création et a la destruction des processus,
- L'utilisation des threads permet a une application d'exécuter plusieurs tâches en même temps ce qui accélère l'exécution de l'application.
- L'utilisation des threads est très intéressante quand la machine est équipé de plusieurs processeurs chaque thread peut s'exécuter sur un processeur différent..





Tranches de temps allouées pendant l'exécution

Modèles de Représentation des Processus

1. **Le système de tâches**

Un processus est décomposé en plusieurs unités de traitement élémentaires appelés tâches. Chaque tâche est composé d'une ou plusieurs instruction machine.

- Quelles sont les tâches qui peuvent être exécutées en parallèle ?

Si par exemple une tâche Tj doit être exécuté après la fin d'une tache Ti on écrit « Ti < Tj ». La relation « < » est appelé relation de précédence.

exemple: T1 < T2

Programme P

T1 : a ← 5 ;
T2 : b ← 6 ;
T3 : a ← a + 1 ;
T4 : a ← a + 2 ;
T5 : c ← a + b ;
T6 : *Écrire*(c) ;

Modèles de Représentation des Processus

- les relations de précédence :

T1<T3, T1<T4, T1<T5, T1<T6

T2<T5, T2<T6,

T3<T4, T3<T5, T3<T6,

T4<T5, T4<T6,

T5<T6.

- Si Ti < Tj et Tj < Tk alors Ti < Tk. (relation transitive).
- Si on supprime les relations qui peuvent être déduites par transitivité on aura les relations de précédence suivantes :

T1 < T3, T2<T5, T3<T4, T4<T5, T5<T6.

- Si les tâches T1 à Tn d'un programme P doivent être exécutés séquentiellement, on écrit: P =T1T2 T3...Tn.

Modèles de Représentation des Processus

2. Conditions de Bernstein:

Les conditions de Bernstein permettent de déterminer si deux instructions (ou tâches) sont exécutable en parallèle ou pas.

Soit une instruction inst ;

- On note par R(inst) : l'ensemble des variable contenu dans l'instruction Inst qui ne changent pas par l'exécution de cette Instruction. Cet ensemble est appelé **domaine de lecture,**

- On note par W(inst) : l'ensemble des variable contenu dans l'instruction inst qui sont modifié par l'exécution de cette Instruction inst. Cet ensemble est appelé **domaine d'écriture**

Modèles de Représentation des Processus

- Les deux instructions inst1 et inst2 sont exécutable en parallèle si les conditions suivantes sont satisfaites :

$R(inst1) \cap W(inst2) = \varnothing$

$W(inst1) \cap R(inst2) = \varnothing$

$W(inst1) \cap W(inst2) = \varnothing$

- Dans ce cas on dit que les instructions (ou d'une manière plus générale : les tâches) inst1 et inst2 son non interférentes.

Exemple :
Soit le programme suivant :

```
Lire(a) ;
Lire(b)
c=a+b ;
Écrire(c) ;
```

Modèles de Représentation des Processus

- R(Lire(a)) = { } W(Lire(a)) = {a}
- R(Lire(b)) = { } W(Lire(b)) = {b}
- R(c=a+b) = {a,b } W(c=a+b) = {c}
- R(Écrire(c)) = {c} W(Écrire(c)) = {}

Les instructions « Lire(a) » et « Lire(b) » peuvent s'exécuter en parallèle

Les instructions « Lire(a) » et « c=a+b ; » non, puisque W(Lire(a)) ∩ R(c=a+b) = {a}

Modèles de Représentation des Processus

3. Le graphe de flot et le graphe de précédence

- Pour représenter un système de tache ont peut éventuellement utiliser le graphes de **précédence** ou le graphe de **flots**.
- Dans le graphe de précédence les nœuds représentent les tâche et les arrêtes représentent la relation de précédence (ordre) entre les tâche.
- Dans le graphe de flots les nœuds représentent les états de calcul et les arêtes représentent les tâches a exécuter.

Modèles de Représentation des Processus

Exemple :

On considère la partie du programme suivante :

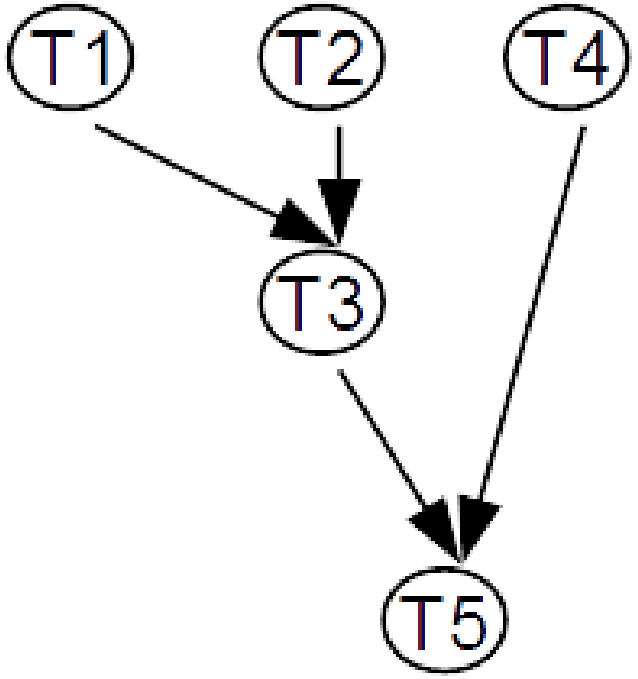
T1 : a := a + b ;
T2 : c := c * d ;
T3 : a := a / c ;
T4 : e := e - f ;
T5 : b := a / e ;

- Dans ce programme on a les relations de précédence suivantes :

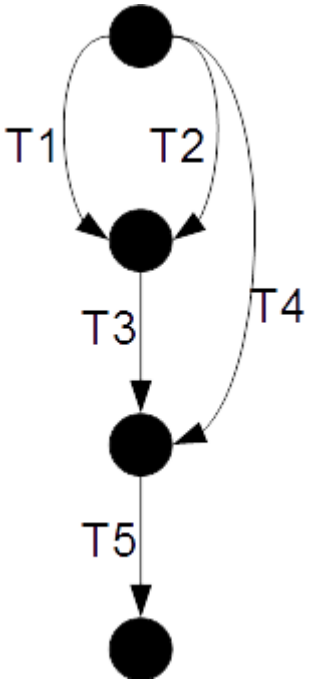
T1 < T3, T2 < T3, T3 < T5, T4 < T5

Modèles de Représentation des Processus

T1 < T3, T2 < T3, T3 < T5, T4 < T5



Le graphe de précédence



Le graphe de flots

Outils d'expression du parallélisme

- **Les primitives Fork , Join et quit (Conway 1963)**:

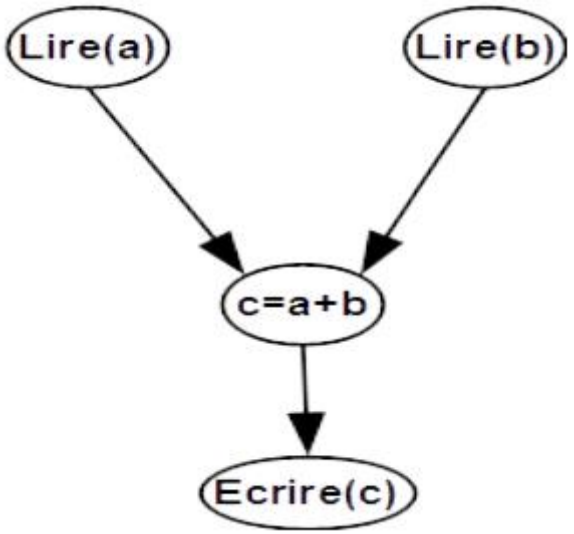
fork *etiq* : Cette instruction vas créer un nouveau processus fils qui s'exécute a partir de l'instruction etiqueté par **etiq**. Le processus père continu son exécution normalement.
quit : Cette instruction termine le processus qui l'a exécuté.
join n : Cette instruction **indivisible (atomique)** se comporte comme suit :

n = n -1 ;
if n ≠ 0 **quit** ;

Outils d'expression du parallélisme

Exemple :
Soit le programme suivant :

```
Lire(a) ;
Lire(b)
c=a+b ;
Écrire(c) ;
```



- Le programme parallèle correspondant est comme suit :

```
n = 2 ;
fork E1 ;
Lire(a) ;
goto E2 ;
E1 : Lire(b) ;
E2 : join n ;
c=a+b ;
Écrire(c) ;
```

Outils d'expression du parallélisme

- **Les primitives PARBEGIN et PAREND (Dijkstra 1968)**
  Ces instruction permettent de créer des bloc d'instructions parallèles. La syntaxe est la suivante :

PARBEGIN

inst1 ;

inst2 ;

Inst3

...

instn ;

PAREND

Les instructions inst1 à instn peuvent s'exécuter en parallèle.

- Le programme précédant vas s'écrire comme suit :

PARBEGIN
Lire(a) ;
Lire(b) ;
PAREND
c=a+b ;
Écrire(c) ;

Le déterminisme et le parallélisme maximal

- Dans un système séquentiel un programme qui s'exécute donne toujours un même résultat, on dit que le système est **déterminé**,
- Par contre dans un système parallèle un programme qui s'exécute en parallèle peut donner des résultat différents selon le comportement du système, dans ce cas on dit que le système est **indéterminé**.

Le déterminisme et le parallélisme maximal

- Soit le programme suivant constitué de quatre tâches :

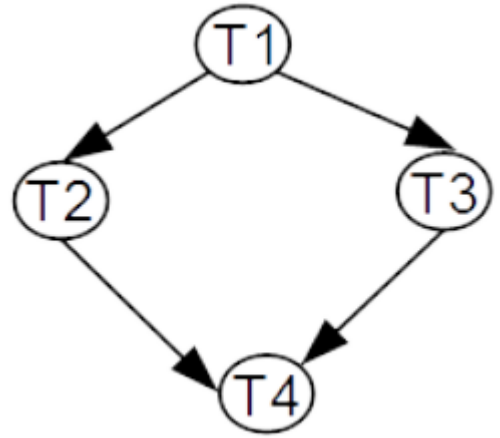
T1 : N:=0;

T2 : N:=N+1;

T3 : N:=N+1;

T4 : Écrire(N);

Le graphe de précédence est comme suit :



Le déterminisme et le parallélisme maximal

- A chaque tâche Ti, on associe sa date de **début** ou d'initialisation di (lecture des paramètres d'entrée, acquisition de ressources nécessaires, chargement d'information) et sa date de terminaison ou de fin fi (écriture des résultats, libération des ressources, sauvegarde d'information).
- Le programme précédant va donner des résultats différents selon son comportement :
- Pour un premier comportement :

w =d1f1d2f2d3f3d4f4 : le résultat = 2

Pour un autre comportement :

w' =d1f1d3d2f2f3d4f4 : le résultat = 1

Case mémoire	d1	f1	d2	f2	d3	f3	d4	f4
C1	0	0	0	1	1	2	2	2

Cellule mémoire	d1	f1	d3	d2	f2	f3	d4	f4
C1	0	0	0	0	1	1	1	1

Le déterminisme et le parallélisme maximal

- **Définition 1:** Un système de tâches S = (T ; <) est déterminé si, pour tous comportements w et w' et pour toute cellule Ci de la mémoire, V (Ci;w) = V (Ci;w')
- V(Ci,w) représentes la succession des valeurs affectés a la cellule (variable) Ci.
- Dans l'exemple précédant :
  V(C1,w) = (0,1,2)
  V(C1,w') = (0,1)

Le déterminisme et le parallélisme maximal

- **Denition 2 :** Un système S = (T;<) est de parallélisme maximal s'il est déterminé et si tout système S' = (T;<') est non déterminé,

où «<'» est obtenu en supprimant du graphe de «< » un couple qui est dans le graphe de précédence de «< ».

D'une autre façon le système est déterminé et si on supprime un arc du graphe de précédence le système devient indéterminé.

Le déterminisme et le parallélisme maximal

- Pour construire le graphe de parallélisme maximal on suit les étapes suivantes :
- 1) Si on n'a pas les conditions de Bernstein satisfaites entre deux tâches T et T' on crée un arc dans le graphe entre les deux tâches,
- 2) Après avoir construit le graphe de précédence on élimine tous les arcs redondants (par transitivité).