UNIVERSITE A.MIRA-BEJAIA
FACULTE DES SCIENCES EXACTES
DEPARTEMENT INFORMATIQUE

SUPPORT DE COURS
SYSTEMES D'EXPLOITATION
NOTIONS DE PARALLELISME/CONSURRENCE

1. Définitions:

- Un algorithme: Description pas à pas des étapes à suivre pour atteindre un résultat.
- Un algorithme séquentiel: son exécution se fera instruction par instruction→ processus.
- Un algorithme parallèle: son exécution se fera plusieurs instructions à la fois.
- Un algorithme distribué: son exécution se fera plusieurs instructions sur plusieurs machines à la fois (donc parallèle aussi).

Niveaux de Parallélisme

- Niveau bit: accès parallèle aux données.
- Niveau instruction: paralléliser l'exécution d'une instruction
- Niveau Programme: le programme est décomposé en tâches indépendantes.
- Niveau application: plusieurs programmes sur plusieurs machines avec un objectif commun.

2. Traitement du parallélisme :

Pour des activités qui se déroulent en parallèle (physiques ou logiques) deux types de relations existent :
- des relations d'ordre conflictuels (compétition/concurrence) : lorsqu'elles partagent des ressources,
- des relations de coopération : lorsqu'elles participent à un même traitement global.

Parallélisme et pseudo-parallélisme : L'exécution des programmes dépend des moyens matériels que nous disposons (un ou plusieurs CPU), du système d'exploitation utilisé (monoprogrammé, multiprogrammé, etc.) et de la nature du programme (séquentiel, parallèle, etc);
Exécution séquentielle des processus : L'exécution séquentielle des programmes se fait quand on a plusieurs programmes à exécuter, alors qu'on a une machine monoprocesseur et un système mono programmé.
Une exécution pseudo-parallèle des programme : est possible quand on a une machine monoprocesseur et un système multiprogrammé ou à temps partagé;
L'exécution parallèle des programmes : se fait quand on a plusieurs programmes à exécuter sur une machine multiprocesseurs.

Les problèmes génériques de conflit ou de coopération existent et leurs solutions relèvent du domaine de la synchronisation. La résolution de ces problèmes de synchronisation peut utiliser différents concepts ou mécanismes qui définissent donc des modèles distincts de machines abstraites logiques et parallèles. Tous ces modèles mettent en jeu le concept fondamental de processus comme outil d'abstraction et de synchronisation et de structuration des traitements parallèles. Un processus modélise un traitement séquentiel.

Deux classes de modèles peuvent être distinguées :
- modèles centralisés : la machine à processus possède une structure avec une mémoire partagée. La synchronisation est implémentée par les sémaphores, les moniteurs, etc. La programmation concurrente permet d'exploiter le travail an parallèle des processeurs et des périphériques. Les

langages comme Portal, Modula-2 et ADA permettent d'écrire entièrement un programme concurrent ou en temps réel.

- modèles répartis : la machine à processus possède une structure décentralisée ou les processus ne peuvent communiquer que par transfert de messages. Les modèles sont caractérisés par des protocoles de communication :
  - les protocoles asynchrones : lorsqu'un processus envoie un message à un autre processus, il continue son exécution indépendamment de la réception du message.
  - les protocoles synchrones : dans ce cas l'émetteur du message bloque jusqu'à l'arrivée d'un accusé de réception.

En général le protocole asynchrone implique une file d'attente de messages envoyés non encore reçus.

3. Modèle de représentation des processus

3.1. Tâche d'un processus

- On appelle **tâche** d'un processus, une unité élémentaire ayant une cohérence logique
- Si un processus ***P*** est constitué de plusieurs tâches (**Ti**), alors: **P = T1 T2 T3 … Tn**.
- Chaque tâche a un début d'exécution ($\mathbf{d_i}$) et une fin d'exécution ($\mathbf{f_i}$):**Ti =** $\mathbf{d_i}$ $\mathbf{f_i}$.

3.2. Comportements d'un processus ou mots (w)

A chaque suite de tâches T1 T2 … Tn est associé à une suite de mots (**Wi**) ou comportements ou les exécutions possible du processus. Wi **=** $\mathbf{d_1f_1}$ $\mathbf{d_2f_2}$ $\mathbf{d_3f_3}$ **…** $\mathbf{d_nf_n}$.

Suites des valeurs écrites dans les variables: On note **V(x,w)** : la suite de valeurs écrites dans la variable x correspondant au comportement w.

Domaines de tâches : A chaque tache T on associe un domaine un domaine de lecture L qui contient l'ensemble des variables nécessitant une lecture et un domaine d'écriture E qui contient l'ensemble des variables nécessitant une écriture sur le mémoire.

3.3. Système de tâches

Un système de tâches S(E, →) est un couple constitué d'un ensemble de tâches (E) et d'une relation de précédence (→) sur cet ensemble E.

Langage d'un système de tâches

On appelle Langage d'un système de tâches L(S) l'ensemble des mots qui décrivent les comportements (exécutions) possibles d'un système de tâches (S), sur l'alphabet {d_1, f_1, d_2, f_2, …d_i, f_i}.

Graphe de précédence d'un système de tâches

Un graphe de précédence permet de représenter les relations de précédence entre les tâches d'un système de tâches (S).

- Les sommets sont les tâches (Ti) du système (S);
- Les arcs sont les relations de précédence entre les tâches.

Propriétés d'un système de tâches

Ci-dessous les propriétés des systèmes de tâches:

1) **∀** $\mathbf{T_i}$ **∈ E,** on n'a pas $\mathbf{T_i \rightarrow T_i}$ (elle n'est pas réflexive)
2) **∀** $\mathbf{(T_i, T_j) \in E,}$ on n'a pas simultanément $\mathbf{T_i \rightarrow T_j}$ et $\mathbf{T_j \rightarrow T_i}$ (elle n'est pas commutative)

3) Soient $\mathbf{T_i, T_j}$ **et** $\mathbf{T_k \in E}$**.** Si, $\mathbf{T_i \rightarrow T_j}$ **et** $\mathbf{T_j \rightarrow T_k}$ alors, $\mathbf{T_i \rightarrow T_k}$ (La relation de précédence est transitive) **;**

Observations sur les contraintes de précédence

1) Si $\mathbf{T_i \rightarrow T_j}$ **alors,** $\mathbf{f_i < d_j}$ **;**
2) Si deux tâches ne sont pas liées par des contraintes de précédence alors elles peuvent s'exécuter en parallèle.

La relation de précédence est non réflexive, non commutative et transitive. C'est une relation d'ordre partiel. Si (A, B) ne sont pas liés par la relation de précédence alors A et B sont dits concurrents.

Exemple 1: Soit un système de tâches (S) associé au programme séquentiel ci-dessous:

Programme :	Système de tâches (S):
Lire (a);	**T1**: ***Lire (a);***
b = a^2;	**T2**: **b=a^2;**
Afficher (b);	**T3**: **afficher (b);**

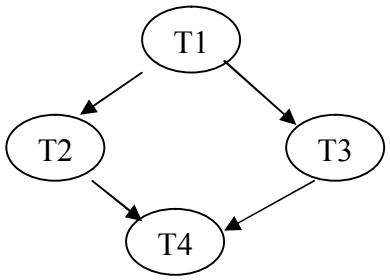
Question : Générer les relations de précédence entre ces taches et dessiner le graphe de précédence (graphe de taches).

Exemple2 : Soit le système de taches suivant et soit le comportement :
w = d1f1 d2f2 d3f3 d4f4 d5f5

Système	Domaine de Lecture	Domaine d'Écriture
*T*1 : lire X *T*2 : lire Z *T*3 : X := X + Z *T*4 : Y := X + Z *T*5 : afficher Y	L1 = ∅ L2 = ∅ L3 = {X; Z} L4 = {X; Z} L5 = {Y}	E1 = {X} E2 = {Z} E3 = {X} E4 = {Y} E5 = ∅

Les suites des valeurs écrites dans les variables:
V (X; w) 0 3 ?
V (Z; w) 0 7 ?
V (Y; w) ?

Exemple 3: Soit le graphe de précédence ci-dessous associé à un système de tâches (S). Donnez le langage de ce système.



Le langage L(S):
ω1: d1f1 d2 f2 d3f3 d4f4
ω2: d1f1 d3 f3 d2f2 d4f4
ω3: d1f1 d2 d3 f2f3 d4f4
ω4: d1f1 d2 d3 f3f2 d4f4
ω5: d1f1 d3 d2 f3f2 d4f4
ω6: d1f1 d3 d2 f2f3 d4f4

3.4. Non-déterminisme :

Un système est dit déterministe si les mêmes sorties correspondent toujours aux mêmes entrées. La programmation parallèle ou distribuée (sur plusieurs cores ou machines) peut souffrir de non-déterminisme. La synchronisation permet de garantir le déterminisme des programmes parallèles.

Exemple4 : Soit le système de tâches ci-dessous: T1 : x :=x+5 ; T2 : x :=2*x ;

- exécution T1, puis T2 : x = 30
- exécution T2, puis T1 : x = 25

Un système est déterminé si, pour toute paire de comportements w1 et w2, et pour toute cellule de mémoire (variable) X, les mêmes suites de valeurs sont écrites dans X lors de l'exécution de w1 et de w2: V (X; w1) = V (X; w2)

Exemple5 : Soit le système de taches suivant et soit les comportements :
*T*1 : lire X
*T*2 : lire Z
*T*3 : X := X + Z
*T*4 : Y := X + Z
*T*5 : afficher Y
Question : démontrer que le système n'est pas déterminé.

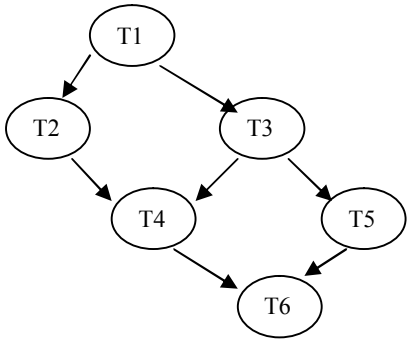
L'interférence et les conditions de Bernstein
Deux tâches *T*1 et *T*2 sont non-interférentes si l'une des conditions 1 ou 2 est vraie :
1. $L1 \cap E2 = \emptyset$ et $L2 \cap E1 = \emptyset$ et $E1 \cap E2 = \emptyset$ (taches indépendantes)
2. $T1 \rightarrow T2$ ou $T2 \rightarrow T1$ (taches non-concurrentes)

Soit *S* un système de tâches.
1) si toutes les taches du système *S* sont non-interférences alors S est déterminé.
2) si *S* est déterminé et toute tâche de *S* a le domaine d'écriture $6 \neq \emptyset$ alors toutes les taches de *S* sont non-interférentes.
Remarque : s'il existe une tache de S dont le domaine d'écriture est vide alors il peut y avoir des interférences cachées.

Parallélisme maximal
Un système de tâches *S* est de parallélisme maximal si la suppression de tout arc $T1 \rightarrow T2$ dans son graphe entraîne l'interférence de *T*1 et *T*2.
En d'autres termes, tous les arcs sont nécessaires pour un fonctionnement correct de S.

Construction du parallélisme maximal
1. prendre toute paire de tâches *T*1 et *T*2 connectées par un chemin dans *S*
2. ajouter l'arc $T1 \rightarrow T2$ à *Smax* si $L1 \cap E2 \neq \emptyset$ ou $L2 \cap E1 \neq \emptyset$ ou $E1 \cap E2 \neq \emptyset$
3. À la fin, éliminer les arcs redondants

Exemple6 : Soit le graphe de taches suivant et les variables impliquées:



Variables	Domaines Li	Domaines Ei
A1	L1, L6	E2
A2	L6	E4
A3	L2, L3	∅
A4	L2, L3, L4	E1, E6
A5	L5	E3, E5

Question : proposer le graphe de taches à parallélisme maximal.

4. Traitement syntaxique des processus :
La description explicite du parallélisme peut s'exprimer sous différentes formes syntaxiques dans un programme. On distingue trois approches :
i) primitive prédéfinie provoquant la création de processus, exemple : fork( ).
ii) introduction d'instructions parallèles dans le langage de programmation, exemple : cobegin/ coend du Concurrent Pascal.
Cobegin <bloc1> ||<bloc2>||…||<blocn> Coend → n processus
Le processus appelant attend la terminaison des processus fils avant de reprendre son exécution.
iii) introduction du type PROCESS doté d'un jeu de primitives permettant de manipuler les objets processus (exemple : Modula-2).
<déclaration de processus>=

```
PROCESS <identificateur> [" (" <paramètres formels> ")"]" ;"
        {<déclarations locales au processus>}
begin
        {<code du processus>}
end <identificateur> " ;"
```

La seule primitive indispensable permettant la création de processus est :
START <identificateur de processus> ["("<arguments>")"] ;

4.1. Primitives de Conway (*Fork, Join et Quit*)
Afin de paralléliser les tâches d'un programme, les instructions ci-dessous sont utilisées.

- L'instruction ***Fork L*** produit deux exécutions parallèles. L'une commence à l'instruction étiquetée par **L** (fils) et l'autre (père) continue juste après l'instruction ***Fork L***. Elle nécessite l'utilisation de l'instruction Goto pour se déplacer entre les étiquettes.
- L'instruction ***Join N*** permet de joindre ***N*** exécutions (processus) parallèles en ne **laissant continuer que le dernier processus**. *Join (N):* ***N = N – 1.*** *Si* ***N*** **≠ 0**, alors **terminer le processus**.
- L'instruction ***Quit*** termine le processus qui l'exécute.

4.2. Primitives de Dijkstra (ParBegin / ParEnd)
Les primitives de **Dijkstra** permettent de paralléliser les tâches d'u programme, mais elles sont moins performantes que les primitives de **Conway**.

- Les instructions ***ParBegin / ParEnd*** permettent de délimiter les blocs d'instructions (tâches) qui peuvent s'exécuter en parallèle.
- Les instructions ***Begin / End*** permettent de délimiter les blocs d'instructions (tâches) qui s'exécutent en séquentiel.

Exemple7: Soit le système de tâches ci-dessous:

T1: ***Lire (a);*** **T2**: ***Lire (b);*** **T3**: ***c = a * b ;*** **T4**: **afficher (c);**	1) Donnez sont graphe de précédence en utilisant un parallélisme maximal 2) Ecrivez le programme parallèle en utilisant les primitives de **Conway** et de **Dijkstra.**

Exemple8 : Soit le programme parallèle suivant utilisant fork/join, donner son graphe de taches :

T1; N1: = 2; fork L1; T2; T4; N2: = 2; fork L2;	T5; Go to L3; L1: T3; L2: join N1; T6; L3: join N2; T7;

5. Mesures de Performance

Soit un problème de taille n (nombre de données ou d'entrées) et un algorithme A qui résout le problème en temps T(n).

1. Temps d'exécution:

- Temps séquentiel: c'est le délai entre le début d'exécution jusqu'à sa terminaison sur une machine séquentielle (un seul processus) noté *Ts*.
- Temps parallèle: plusieurs processus, c'est le délai entre le début d'exécution jusqu'à la terminaison du dernier processus noté *Tp*.
- Comportement asymptotique: c'est le nombre d'opérations élémentaires (addition, multiplication, lecture/écriture en mémoire) effectué par l'algorithme.

Exemple: addition de n nombres: $Ts = n - 1$ noté $Ts = \theta(n)$ (borne supérieure de *Ts*)

3. Accélération (Speed-up):

C'est la mesure du bénéfice d'utiliser une solution parallèle par rapport à une solution séquentielle pour un même algorithme exécuté sur un seul processeur et sur p processeur (*Ts* représente le temps du meilleur algorithme séquentiel) :

$S(p) = Ts(n)/Tp(n)$

En théorie l'accélération ne dépasse jamais le nombre de processeurs et donc idéalement $S(p) = P$ (le travail parallèle est P fois supérieur que le travail séquentiel).

4. Efficacité (Efficiency):

Un processeur ne passe pas 100% de son temps au calcul effectif (beaucoup d'opérations d'entrée/sortie et dans les files d'attente) donc la mesure efficacité donne la fraction de temps utile ou le taux moyen d'utilisation des processeurs.

$E(p) = S(n)/P$

Idéalement $E(p) = 1$

En général: $0 < S < P$ et $0 < E < 1$

Remarque: si $E > 1$ alors on parle d'efficacité super linéaire. C'est possible dans le cas de non-déterminisme et en présence d'effets de cache.

Exemple9: addition de n nombres: Calculer l'accélération et l'efficacité si on suppose un nombre maximal de processeurs et qu'au départ chaque processeur dispose d'une donnée?

Exercice : Donner l'accélération et l'efficacité du programme donnée dans l'exemple 6.

Mme YAICI