

# Chapitre 1 : Notion de concurrence

## 1. Introduction

Avec l'introduction des machines disposants de plusieurs processeurs, l'exécution parallèle entre plusieurs séquences d'instructions est donc permise. Cependant, les problèmes actuels n'impliquent pas toujours l'exécution purement parallèle des actions les concernant. Des contraintes d'ordonnancement entre certaines actions sont alors à prendre en considération. Le besoin d'introduire des langages évolués permettant d'exprimer ces possibilités d'exécution et des mécanismes permettant de contrôler l'exécution se sont vite sentis. Il s'agit donc de permettre l'exécution en parallèle de plusieurs instructions tout en considérant les contraintes d'ordonnancement.

## 2. Processus séquentiel

Un processus séquentiel (ou simplement processus) est une suite d'instructions qui s'exécutent les unes après les autres. L'exécution d'une instruction fait changer le processus d'un état à un autre. L'état est défini par l'état de l'ensemble des variables accessibles et par l'état des ressources (ie. objets) éventuelles utilisées (par exemple l'état d'un périphérique ou d'un fichier). Les instructions considérées sont celles du langage machine. On peut aussi s'intéresser à celle d'un langage évolué ou celle d'un langage intermédiaire.

Dans un but de généralité, on introduit la notion de tâche. C'est une unité élémentaire de traitement, ayant une *cohérence logique*. Sa taille peut être variable.

Un processus P peut être décrit par une suite de tâches  $T_1, T_2, \dots, T_n$ .

### Exemple 1.1: P1

Lire (X) ;  $T_1$   
 $X \leftarrow X + 1$  ;  $T_2$   
 Ecrire (X) ;  $T_3$

## 3. Programme concurrent

Un programme concurrent est un programme dont certaines instructions ne s'exécutent pas de manière séquentielle mais partagent les mêmes variables.

Considérons l'évaluation de l'expression

$$E = (X - Y) / (Z + T)$$

L'évaluation séquentielle consiste à calculer dans l'ordre :

$$X - Y ; Z + T \text{ puis } (X - Y) / (Z + T)$$

L'évaluation concurrente consiste à calculer :

$$\text{simultanément } X - Y \text{ et } Z + T, \text{ puis } (X - Y) / (Z + T)$$

Si on considère par exemple que l'évaluation de chaque opération coûte une unité de temps, le temps total de l'évaluation parallèle est alors réduit d'une unité relativement à l'évaluation séquentielle.

## 4. Systèmes de tâches [2]

### 4.1. Définitions

La concurrence implique deux modes d'exécution, le parallélisme et la séquentialité entre les tâches. Ce qui définit la relation de précédence comme suit :

- Soit un ensemble E de tâches, la relation de précédence, notée  $<$ , entre tâches doit vérifier :

- $\forall T \in E$  on n'a pas  $T < T$  ;
- $\forall T, T' \in E$  si  $T < T'$ , on n'a pas  $T' < T$  ;
- $\forall T, T', T'' \in E$  si  $T < T'$  et  $T' < T'' \Rightarrow T < T''$ .
- Un système de tâches est défini donc par  $S = (E, <)$ .

Exemple 1.2 : Soit  $E = \{T_1, T_2, T_3, T_4\}$ , un ensemble de tâches avec les relations de précédences suivantes :  $T_1 < T_3$        $T_2 < T_4$        $T_1 < T_2$

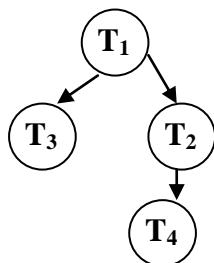
Pratiquement :

- Si  $T_i < T_j$  alors la terminaison de  $T_i$  doit avoir lieu avant l'initialisation de  $T_j$ .
- Si  $T_i \not< T_j$  et  $T_j \not< T_i$ , alors l'ordre d'exécution de  $T_i$  et  $T_j$  est sans importance, on dit que  $T_i$  et  $T_j$  sont *parallèles* ou *indépendantes*.

#### 4.2. Graphe de précédences

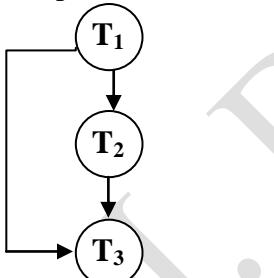
Un système de tâches décrit l'ordre dans lequel les différentes tâches du système doivent être exécutées. Cet ordre peut être décrit schématiquement par un graphe orienté dont les nœuds représentent les tâches et les arcs, les relations de précédences entre tâches.

Exemple 1.3 :



- Ce graphe représente le système de tâches de l'exemple 1.2.
- $T_3 \not< T_2$  et  $T_2 \not< T_3 \Rightarrow T_2$  et  $T_3$  s'exécutent en parallèle

Exemple 1.4 :



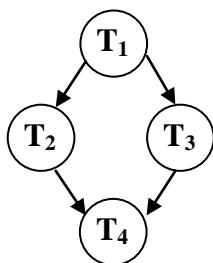
- Ce graphe est redondant à cause de l'arc  $(T_1, T_3)$ .
- La redondance peut être éliminée en gardant uniquement les arcs  $(T_i, T_j)$  vérifiant  $\forall T_k$  on n'a pas  $T_i < T_k < T_j$
- Le graphe obtenu est appelé graphe de précédence.

Le graphe de l'exemple 1.3 est un graphe de précédences, dans ce graphe :

- $T_1$  est une tâche initiale.
- $T_3, T_4$  tâches finales.
- $T_2$  est un successeur immédiat de  $T_1$ .
- $T_4$  est un successeur de  $T_1$ .
- $T_3$  et  $T_2$  sont indépendantes.

#### 4.3. Comportements d'un système de tâches

Un comportement d'un système de tâches est le résultat de l'entrelacement (ou de la linéarisation) permis(e) de l'exécution des tâches du système. L'exécution d'un système de tâches peut définir plusieurs comportements différents. L'ensemble de ces comportements, représente le langage du système de tâches.

Exemple 1.5:

$T_1$  : Lire (n) ;  
 $T_2$  :  $n := n+2$  ;  
 $T_3$  :  $n := n/3$  ;  
 $T_4$  : Ecrire (n);

Décomposons les tâches  $T_2$  et  $T_3$ :

$T_2$ : Load $R_1, n$ (a)	$T_3$ : Load $R_2, n$ (a')
ADD $R_1, 2$ (b)	Div $R_2, 3$ (b')
Store $R_1, n$ (c)	Store $R_2, n$ (c')

Supposons  $n=10$

Certains comportements possibles sont :

$T_1 ; (a)(b)(c)(a')(b')(c')$ ; $T_4$	$R_1 = 10$	$R_1 = 12$	$n = 12$
	$R_2 = 12$	$R_2 = 4$	$n = 4$
$T_1 ; (a')(b')(c')(a)(b)(c)$ ; $T_4$	$R_2 = 10$	$R_2 = 3.33$	$n = 3.33$
	$R_1 = 3.33$	$R_1 = 5.33$	$n = 5.33$
$T_1 ; (a)(a')(b)(c)(b')(c')$ ; $T_4$	$R_1 = 10$	$R_2 = 10$	
	$R_1 = 12$		$n = 12$
	$R_2 = 3.33$		$n = 3.33$
$T_1 ; (a')(a)(b')(c')(b)(c)$ ; $T_4$	$R_2 = 10$	$R_1 = 10$	
	$R_1 = 3.33$		$n = 3.33$
	$R_1 = 12$		$n = 12$

Remarquons qu'une tâche est composée de trois étapes :

- Lecture de données désignée par (d)
- Traitement local désignée par (t)
- Ecriture des résultats désignée par (f)

et seuls les lectures et écritures influent sur le comportement. Les comportements possibles du système de tâches ci-dessus se résument en :

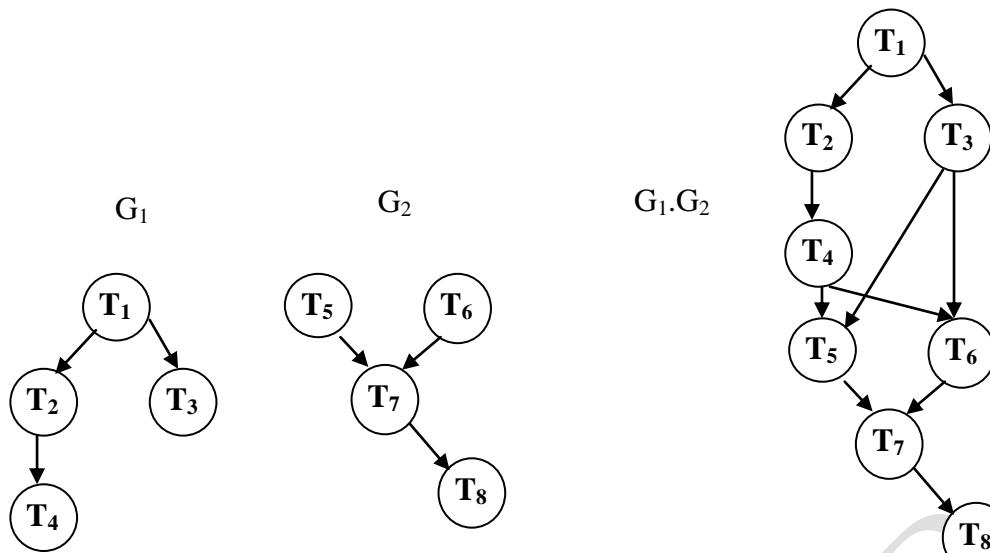
$T_1 \text{ df } d'f' T_4$   
 $T_1 \text{ d'f' df } T_4$   
 $T_1 \text{ dd' ff' } T_4$   
 $T_1 \text{ dd' f'f' } T_4$

#### 4.4. Opérations sur les graphes de précédences

On distingue deux opérations sur les graphes de précédences : le produit et la composition parallèle et sont relatives respectivement à leurs exécutions séquentielle et parallèle.

- Le produit de  $G_1$  et  $G_2$  : noté  $G_1.G_2$ , est le graphe obtenu en reliant chaque nœud terminal de  $G_1$  à chaque nœud initial de  $G_2$ .

Exemple 1.6:



Dans  $G_1.G_2$ , les tâches de  $G_2$  ne peuvent s'exécuter que si toutes les tâches de  $G_1$  se terminent.

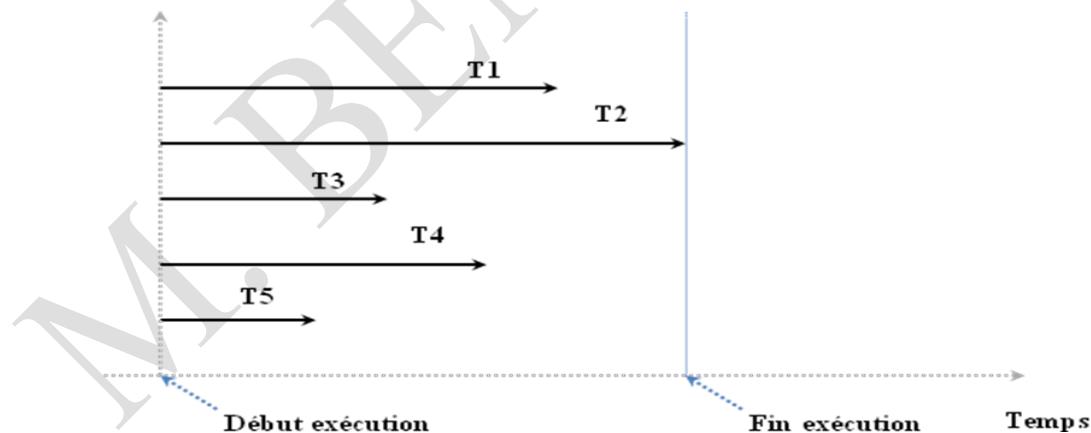
- La composition parallèle : Le graphe parallèle  $G_1//G_2$  de  $G_1$  et  $G_2$  est l'union des deux graphes  $G_1$  et  $G_2$ . Graphiquement, il suffit de juxtaposer les deux graphes  $G_1$  et  $G_2$ .

## 5. Parallélisme

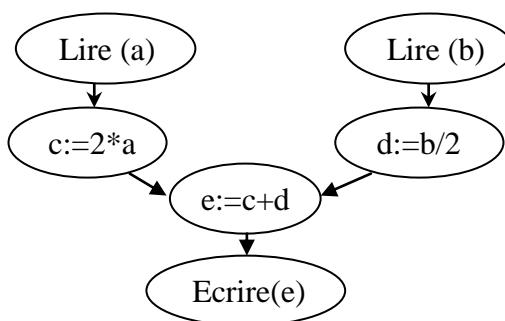
Les langages parallèles sont dotés de mécanismes permettant l'exécution des tâches en séquences ou en parallèles. Nous allons décrire deux mécanismes différents indépendamment de toute intégration dans un langage évolué.

### 5.1. Structure Parbegin/Parend

L'instruction *Parbegin*  $T_1 ; T_2 ; \dots ; T_n$  *Parend* ; permet de mettre  $n$  tâches  $T_i$ ,  $i=1..n$  en parallèles, cette instruction termine quand toutes ces tâches se terminent (voir Figure 1.1).



**Figure 1.1 :** Exemple de diagramme de temps d'exécution de parbegin parend.

Exemple 1.7:

Debut  
 Parbegin  
 Debut  
 Lire (a) ; c := 2\*a  
 Fin ;  
 Debut  
 Lire (b) ; d := b/2  
 Fin ;  
 Paren ;  
 e := c+d ;  
 Ecrire (e) ;  
 Fin.

Une autre solution est donnée par :

Debut  
 Parbegin Lire (a) ;Lire (b) Paren ;  
 Parbegin c :=2\*a ; d :=b/2 Paren ;  
 e:= c+d ; Ecrire (e)

Fin.

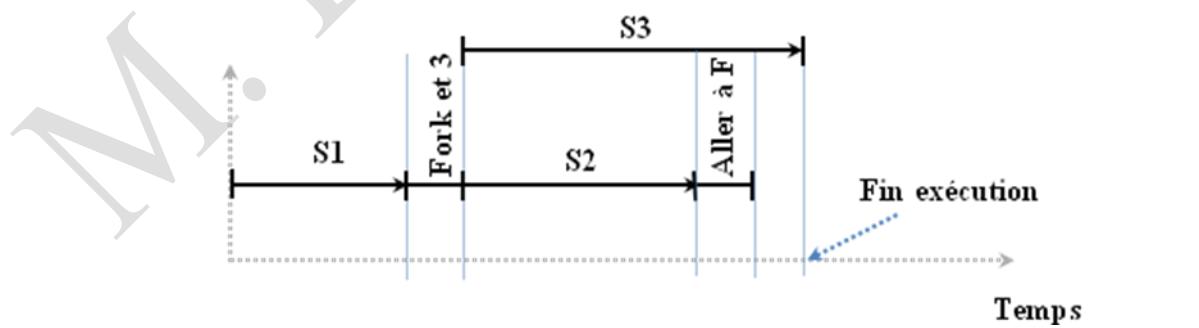
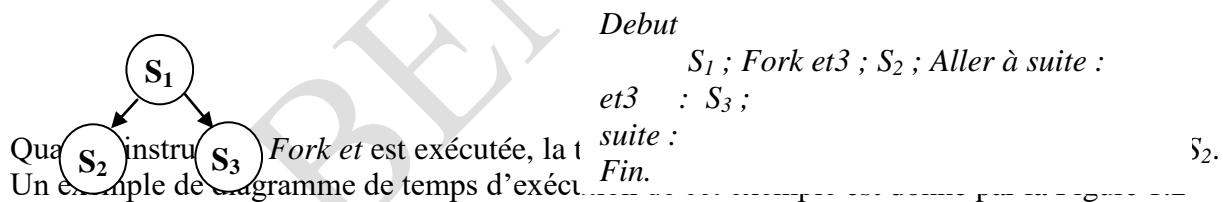
Dans ce cas, le temps de calcul de Lire (a) et Lire (b) est supposé être le même.

**5.2. Structure Fork/join**

La structure *Fork/join* est introduite par Conway (1963) et Dennis et Van Horn(1966).

a- Fork et: Cette instruction produit une nouvelle activité commençant à l'étiquette *et* parallèle à celle qui est en cours et qui continue après l'instruction Fork et :

Exemple 1.8: Exprimons le graphe de précédences suivant à l'aide de fork et join.



**Figure 1.2 :** Exemple de diagramme de temps d'exécution de l'exemple 1.8.

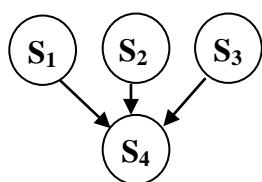
b- Join n : Cette structure est définie comme suit :

$n := n-1$ ; Si ( $n \neq 0$ ) Alors quit Fsi;

// L'instruction *quit* signifie arrêter l'exécution de cette séquence à ce point.

Elle permet de réduire deux ou plus d'activités en cours d'exécution à une seule. Si  $n$  est initialisée à une valeur  $k$  qui représente le nombre d'activités à réduire, les  $(k-1)$  premières activités qui exécutent *Join n* s'arrêtent (car  $n > 0$ ), seule la dernière dans le temps va poursuivre son exécution après cette instruction.

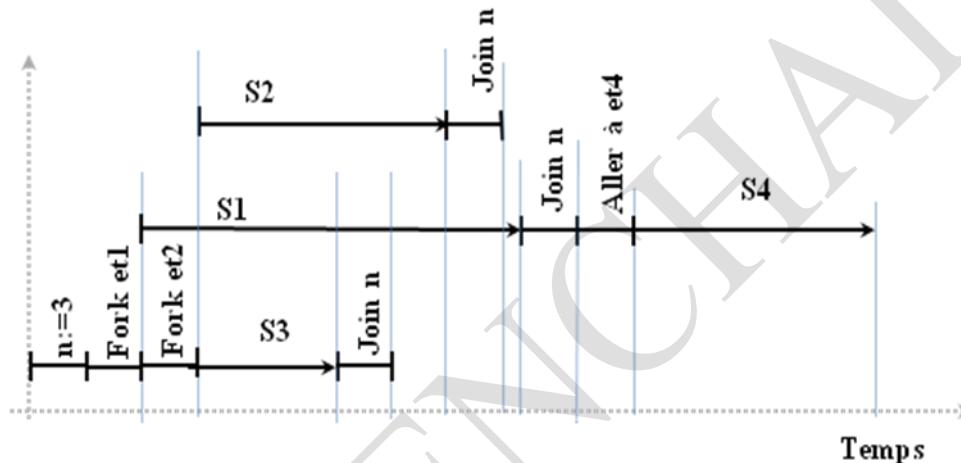
### Exemple 1.9:



*Debut*  
 $n := 3 ;$   
*Fork et<sub>1</sub> ;*  
*Fork et<sub>2</sub> ;*  
 $S_3 ; \text{Join } n ; \text{aller à et}_4 ;$       Ou      mieux  
*et<sub>1</sub> : S<sub>1</sub> ; Join n ;*  
*aller à et<sub>4</sub> ;*  
*et<sub>2</sub> : S<sub>2</sub> ; join n ;*  
*et<sub>4</sub> : S<sub>4</sub> ;*  
*Fin.*

*Debut*  
 $n := 3 ;$   
*Fork et<sub>1</sub> ; Fork et<sub>2</sub> ;*  
 $S_3 ; \text{Aller à et}_4 ;$   
*et<sub>1</sub> : S<sub>1</sub> ; Aller à et<sub>4</sub> ;*  
*et<sub>2</sub> : S<sub>2</sub> ;*  
*et<sub>4</sub> : Join n ; S<sub>4</sub> ;*  
*Fin.*

Un exemple de diagramme de temps d'exécution du programme de l'exemple 1.9 est donné par la Figure 1.3.



**Figure 1.3 :** Exemple de diagramme de temps d'exécution de l'exemple 1.9

- Cette instruction est exécutée de manière indivisible (si plusieurs activités tentent de l'exécuter en parallèle, une seule est admise à la fois).
- Elle est utilisée pour exprimer un rendez-vous entre plusieurs tâches. Une autre formulation de cette instruction existe :

*Join n, etiq :*

```

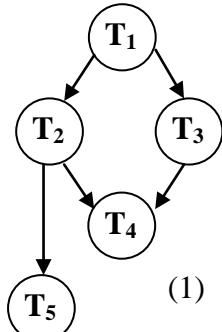
n := n-1 ;
Si (n = 0) Alors Aller à etiq
    [Sinon quit;]
Fsi ;
  
```

Les instructions de branchement du programme ci-dessus sont écrites comme suit (sans considérer l'option [sinon quit]) :

*Debut*  
 $n := 3 ; \text{Fork et}_1 ; \text{Fork et}_2 ;$   
 $S_1 ; \text{Join } n, \text{etiq} ; \text{quit} ;$   
*et<sub>1</sub> : S<sub>2</sub> ; Join n, etiq ; quit ;*  
*et<sub>2</sub> : S<sub>3</sub> ; Join n, etiq ; quit ;*  
*etiq : S<sub>4</sub> ;*  
*Fin.*

c- Relation entre Parbegin/Parend et Fork/Join : La structure *Fork/Join* est plus expressive que *Parbegin/Parend*. En effet, cette dernière ne peut exprimer que les graphes dits *proprement liés*, c.à.d dont les tâches peuvent être distinguées proprement comme parallèles ou séquentielles. Autrement dit, si on arrive à décomposer le graphe hiérarchiquement en sous graphes parfaitement parallèles ou séquentielles jusqu'aux tâches élémentaires, on déduit que ce graphe est proprement lié.

Exemple 1.10 :

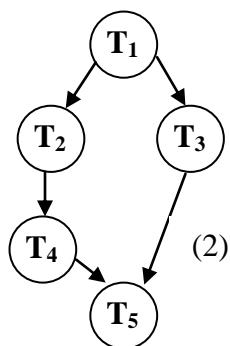


Essayons de décomposer G1:

- G1 est vu comme T1 séquentiel au bloc qui est le reste de G1.
- Ce bloc n'est pas décomposable.

Donc, G1 n'est pas proprement lié.

La suppression de l'arc (T<sub>3</sub>, T<sub>4</sub>) ou (T<sub>2</sub>, T<sub>5</sub>) ou (T<sub>2</sub>, T<sub>4</sub>) rend le graphe proprement lié.



Dans le graphe (2), on distingue que :

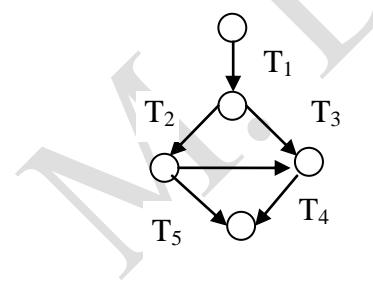
- T<sub>1</sub> est séquentielle au système [T<sub>2</sub>, T<sub>3</sub>, T<sub>4</sub>, T<sub>5</sub>],
- Le système [T<sub>2</sub>, T<sub>3</sub>, T<sub>4</sub>] séquentiel à T<sub>5</sub>,
- Le système [T<sub>2</sub>, T<sub>4</sub>] est parallèle à T<sub>3</sub>,
- T<sub>2</sub> est séquentielle à T<sub>4</sub>.

C'est alors une structure proprement liée (ou *blockage*).

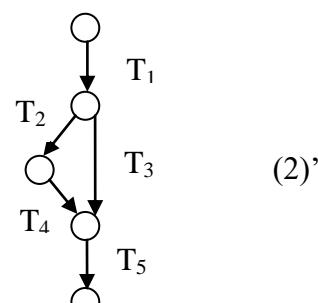
Remarque : La structure blockage du système de tâches est directement visible si on redéfinie le graphe comme suit :

- Les nœuds définissent les rendez-vous, et
- Les arcs les tâches.

Avec cette redéfinition, les deux graphes ci-dessus deviennent respectivement comme les graphes (1)' et (2)' ci-dessous :



(1)'



(2)'

Le graphe (1) ne peut alors être décrit par l'instruction *Parbegin/Parend* seule. Il est nécessaire de lui adjoindre des outils de synchronisation (voir Chapitre 2).

## 6. Déterminisme d'un système de tâches [2]

### 6.1. Dépendances entre tâches

Soit l'exemple suivant :

Exemple 1.11 :

P :

Lire (A) ;	T <sub>1</sub>
Lire(B) ;	T <sub>2</sub>
B := 2*B ;	T <sub>3</sub>
C := A+B ;	T <sub>4</sub>
Ecrire (C)	T <sub>5</sub>

Si les valeurs de A et B sont resp. 10 et 15,  
le résultat C=40

Pour augmenter le degré de parallélisme, il est nécessaire de tenter de supprimer certaines contraintes de précédences *inutiles* entre tâches. Divisons le processus P en deux processus parallèles P<sub>1</sub> et P<sub>2</sub>.



Une exécution possible du système de tâches P<sub>1</sub>//P<sub>2</sub> est :

- |            |                |   |
|------------|----------------|---|
| Lire (A) ; | T <sub>1</sub> | - Avec les mêmes valeurs de A et B que                            |
| Lire (B)   | T <sub>2</sub> | ci-dessus, le résultat est C=25                                   |
| C := A+B   | T <sub>4</sub> |   |
| B := 2*B   | T <sub>3</sub> | - Ce nouveau système (P <sub>1</sub> //P <sub>2</sub> ) n'est pas |
| Ecrire (C) | T <sub>5</sub> | "équivalent" à P.   |

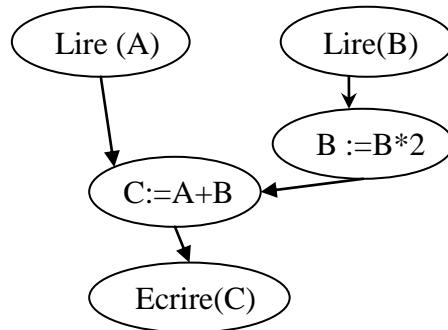
Une autre exécution de ce système est : T<sub>1</sub> ; T<sub>2</sub> ; T<sub>3</sub> ; T<sub>4</sub> ; T<sub>5</sub>. Elle donne le même résultat que celui de P.

En observant de près ces deux comportements, on déduit que la différence des résultats est due à l'ordre d'exécution des tâches T<sub>3</sub> et T<sub>4</sub>. Plus précisément, la variable B est modifiée par T<sub>3</sub> et utilisée en lecture par T<sub>4</sub>. Selon l'ordre d'exécution, la variable C prend une valeur ou une autre.

Procédons par intuition à la recherche d'un graphe qui donne le même résultat que celui de P mais avec un minimum de contraintes de précédence entre tâches.

- Les instructions Lire(A) et Lire(B) peuvent s'exécuter dans un ordre quelconque, on dit qu'elles sont *indépendantes* ou *n'interfèrent pas*.
  - De même que Lire(A) et B := 2\*B.
  - Par contre, Lire(B) et B:=2\*B sont séquentielles.
  - De même que B := 2\*B et C:=A+B.
  - De même que Lire(A) et C:=A+B.
  - De même que C :=A+B et Ecrire(C).

On obtient alors le graphe de la Figure 1.4



**Figure 1.4 :** Graphe de précédences de parallélisme maximal de l'exemple 1.11

## 6.2 Etats d'un système de tâches

Rappelons qu'un comportement d'un système de tâches est l'entrelacement de l'exécution des actions des différentes tâches en respectant les contraintes de précédences. Le système évolue donc de manière discrète. Il est observable après l'exécution de chaque action. Les actions significatives retenues sont soit d (initialisation), soit f (terminaison) d'une tâche.

L'état d'un système de tâches est défini par l'état des variables et/ou des périphériques après l'exécution d'une action dans un comportement donné.

Soit  $S$ , un système de tâches. Supposons que la suite des exécutions est :  $a_1 a_2 \dots a_i \dots a_n$ , la suite des états obtenus après exécution de cette séquence d'actions est donnée par :  $S_0 S_1 \dots S_i \dots S_n$  où  $S_i$  est l'état après l'exécution de l'action  $a_i$  et  $S_0$  est l'état initial. Considérons le système de tâches séquentiel  $P$  ci-dessus et supposons que les valeurs affectées en premier à  $A$  et  $B$  sont respectivement  $a$  et  $b$ . La suite des états du système est la suivante :

$$(-, -, -)^{d1}; (-, -, -)^{f1}; (a, -, -)^{d2}; (a, -, -)^{f2}; (a, b, -)^{d3}; (a, b, -)^{f3}; (a, 2b, -)^{d4}; (a, 2b, -)^{f4}; \\ (a, 2b, a+2b)^{d5}; (a, 2b, a+2b)^{f5}; (a, 2b, a+2b).$$

## 6.3. Déterminisme d'un système de tâches

Définition : Un système de tâches est dit *déterminé*, si pour chaque comportement possible de ce système, la suite de valeurs affectées à chaque variable est la même.

Exemple 1.12: Le système de tâches obtenu intuitivement à partir de  $P$  (voir section 6.1) admet certains comportements suivants :

$$C_1 = d_1 f_1 d_2 f_2 d_3 f_3 d_4 f_4 d_5 f_5$$

$$C_2 = d_2 f_2 d_1 f_1 d_3 f_3 d_4 f_4 d_5 f_5$$

$$C_3 = d_2 f_2 d_3 f_3 d_1 f_1 d_4 f_4 d_5 f_5$$

Il est clair que tous ces comportements engendrent la suite de valeurs suivantes pour chaque variable, c'est un système déterminé.

$$A : (a)$$

$$B : (b, 2b)$$

$$C : (a+2b)$$

Par contre, le système de tâches  $P_1//P_2$  n'est pas déterminé étant donné qu'au moins deux comportements engendrent deux résultats différents pour la variable  $C$ .

**Remarque :**

- Tout système de tâches séquentiel est déterminé car il admet un seul comportement.
- Un système de tâches faiblement déterminé se contente de la valeur finale de chaque variable qui doit être identique pour tous les comportements de ce système.

**6.4 Conditions de Bernstein**

Dans un système de tâches, chaque tâche utilise des variables pour lecture et d'autres pour écriture. On note :

$L_T$ : domaine de lecture d'une tâche  $T$ . Il est défini par l'ensemble des variables utilisées par la tâche  $T$  en lecture.

$E_T$  : domaine d'écriture d'une tâche  $T$ . Il est défini par l'ensemble des variables utilisées par la tâche  $T$  en écriture.

Exemple 1.13: Dans l'exemple de P.

$$\begin{array}{ll} L_{T1} = \emptyset & E_{T1} = \{A\}. \\ L_{T2} = \emptyset & E_{T2} = \{B\}. \\ L_{T3} = \{B\} & E_{T3} = \{B\}. \\ L_{T4} = \{A, B\} & E_{T4} = \{C\}. \\ L_{T5} = \{C\} & E_{T5} = \emptyset. \end{array}$$

Définition : Soit  $S$ , un système de tâches. Deux tâches  $T$  et  $T'$  de  $S$  sont dites *non-interférentes* par rapport à  $S$  si :

- Soit  $T$  est prédécesseur ou un successeur de  $T'$
- Soit  $E_T \cap L_{T'} = L_T \cap E_{T'} = E_T \cap E_{T'} = \emptyset$

Exemple 1.14: Dans le système de tâches  $P_1//P_2$ , les tâches  $T_3$  et  $T_4$  sont non non - interférentes (ou interférentes) car :

- $T_3$  n'est ni successeur ni prédécesseur de  $T_4$ , et
- $L_{T3} = \{B\}$      $E_{T3} = \{B\}$
- $L_{T4} = \{A, B\}$      $E_{T4} = \{C\}$

Donc,  $E_{T3} \cap L_{T4} = \{B\} \neq \emptyset$

Par contre,  $T_1$  et  $T_2$  sont non- interférentes.

Théorème 1 : Soit  $S$  un système de tâches. Si  $S$  est constitué de tâches deux à deux non-interférentes, alors il est déterminé quelque soit les fonctions calculées par les tâches.

Exemple 1.15 : Le système de tâches obtenu intuitivement à partir de P est déterminé car les tâches sont deux à deux non – interférentes par construction.

Remarque : Pour vérifier qu'un système de tâches est déterminé, il peut être intéressant de vérifier seulement si les tâches sont 2 à 2 non- interférentes. On échange, si au moins deux tâches sont interférentes, cela ne signifie pas que le système de tâches est non déterminé.

**7. Parallélisme maximal [2]**

Il est utile d'atteindre à partir d'un système de tâches un autre qui donne les "mêmes résultats" et avec un minimum de contraintes de précédences. La conséquence directe est l'amélioration du degré de parallélisme. Cette section permet de décrire une méthode de recherche d'un système de tâches de parallélisme maximal à partir d'un autre.

**7.1. Définitions**

Définition 1: Soit deux systèmes de tâches  $S$  et  $S'$  sur un même ensemble de tâches.  $S$  et  $S'$  sont dits équivalents :

- s'ils sont déterminés et
- si pour toute variable utilisée, les suites de valeurs affectées sont identiques pour les deux systèmes de tâches.

Exemple 1.16: Le système de tâches P vu précédemment et le système de tâches obtenu intuitivement à partir de P sont équivalents car ils sont déterminés et les suites de valeurs affectées à A, B et C sont identiques.

Définition 2: Un système de tâches est de parallélisme maximal s'il est déterminé et si la suppression de toute précédence (T, T') du graphe correspondant engendre l'interférence de tâches T et T'.

Exemple 1.17: Dans l'exemple du système de tâches obtenu par intuition de P, la suppression de tout arc engendre l'interférence de tâches, donc il est de parallélisme maximal.

## 7.2. Construction d'un système de tâches de parallélisme maximal

Soit  $S=(E, \leq)$ , un système de tâches déterminé. Il existe un système de tâches unique  $S'$  de parallélisme maximal équivalent à S. Il est obtenu en ne gardant que les précédences (T, T') tel que :

$$[(L_T \cap E_{T'}) \neq \emptyset \text{ ou } (L_{T'} \cap E_T) \neq \emptyset] \text{ ou } (E_T \cap E_{T'}) \neq \emptyset]$$

Pour construire ce système de tâches, il faudra examiner les tâches liées par des contraintes de précédences deux à deux pour décider de la suppression ou non de cette précédence. À l'issu de cette opération, le système de tâches  $S'$  obtenu doit être purgé des arcs redondants.

Applications 1: Construire le système de tâches de parallélisme maximal équivalent au système de tâches séquentiel suivant:

Lire(A);	A:=C*A;
Lire(B);	B:=B/D;
C:=A+B;	E:=A+B;
D:=C+A;	Ecrire(E);

### Solution

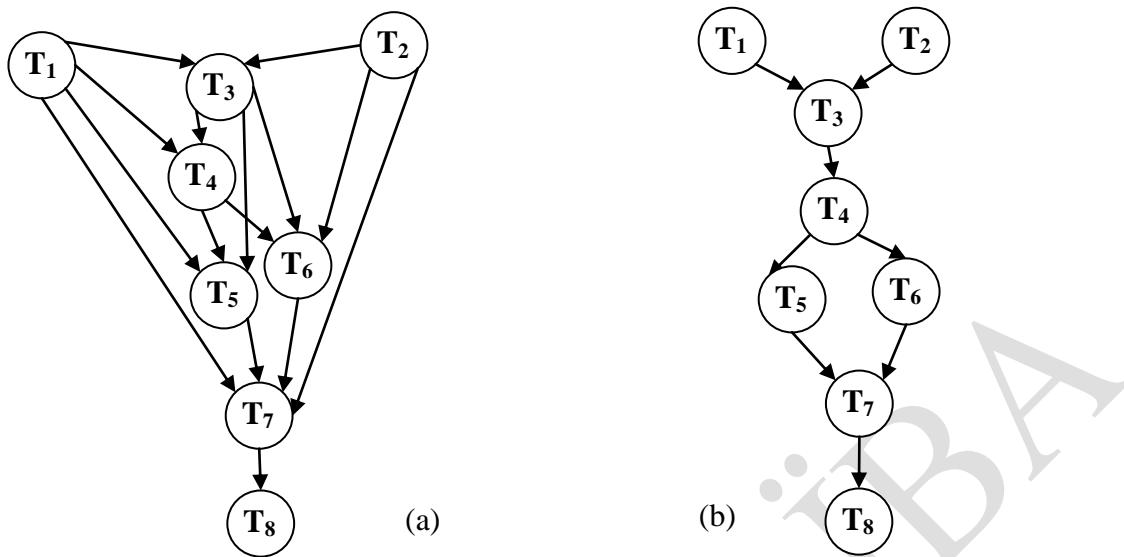
$L_{\text{Lire}(A)} = \emptyset$	$E_{\text{Lire}(A)} = \{A\}$
$L_{\text{Lire}(B)} = \emptyset$	$E_{\text{Lire}(B)} = \{B\}$
$L_{C := A+B} = \{A, B\}$	$E_{C := A+B} = \{C\}$
$L_{D := C+A} = \{C, A\}$	$E_{D := C+A} = \{D\}$
$L_{A := C*A} = \{C, A\}$	$E_{A := C*A} = \{A\}$
$L_{B := B/D} = \{B, D\}$	$E_{B := B/D} = \{B\}$
$L_{E := A+B} = \{A, B\}$	$E_{E := A+B} = \{E\}$
$L_{\text{Ecrire}(E)} = \{E\}$	$E_{\text{Ecrire}(E)} = \emptyset$

Examinons les liens de précédences existants (G: Garder; S: Supprimer)

T1 < T2 S	T2 < T3 G	T3 < T4 G	T4 < T5 G
T1 < T3 G	T2 < T4 S	T3 < T5 G	T4 < T6 G
T1 < T4 G	T2 < T5 S	T3 < T6 G	T4 < T7 S
T1 < T5 G	T2 < T6 G	T3 < T7 S	T4 < T8 S
T1 < T6 S	T2 < T7 G	T3 < T8 S	
T1 < T7 G	T2 < T8 S		
T1 < T8 S			

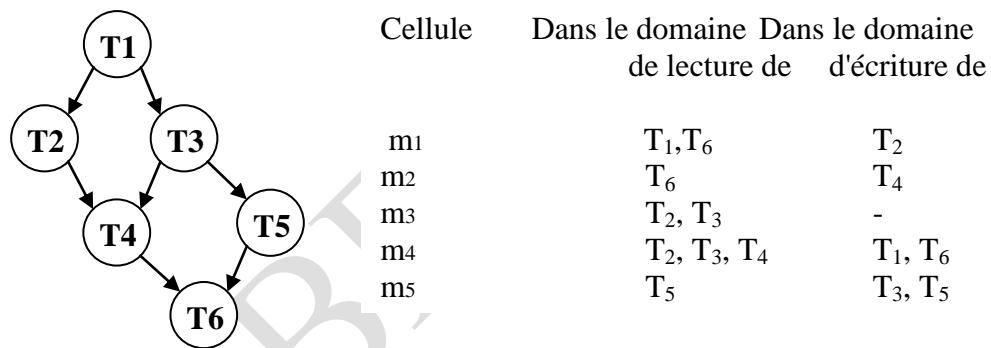
T5 < T6 S	T6 < T7 G	T7 < T8 G
T5 < T7 G	T6 < T8 S	
T5 < T8 S		

Le graphe obtenu à partir des liens restants est donné par la Figure 1.5(a). Le graphe résultats après élimination des arcs redondants est donné par la Figure 1.5(b).



**Figure 1.5 :** Graphes de précédences de l'application 1

Application 2: Même question que précédemment.



## Références

- [1] M. J. Bach, traduit par G. Fellah, "Conception du Système UNIX," Masson et Prentice Hall, 1990.
- [2] J. Beauquier, B. Berard, "Systèmes d'exploitation : concepts et algorithmes," McGraw-Hill, 1990.
- [3] Cornafion, "Systèmes informatiques répartis: concepts et techniques," Dunod informatique, 1981.
- [4] Crocus, "Systèmes d'exploitation des ordinateurs", Dunod 1975.
- [5] N. B. Fontaine, P. Hammes, "UNIX système V: Système et environnement" Masson 1989.
- [6] S. Krakowiak, "Principes des systèmes d'exploitation des ordinateurs," Dunod 1987.
- [7] J-L.Peterson, A.Silbershartz "Operating Systems Concepts," Addison-Wesly Publishing Company, Inc, 1983.
- [8] A. Silberschatz, P. B. Galvin "Principes des systèmes d'exploitation," 4 e Edition, Addison Wesley, 1994.
- [9] A. Tanenbaum, "Modern Operating Systems, " Second Edition Prentice Hall.
- [10] A. Tananbaun, "Systèmes d'exploitation," 3°édition, Pearson Edition. 2008.
- [11] J-P. Verjus et al, "Synchronisation des programmes parallèles : Expression et mise en oeuvre dans les systèmes centralisés et distribués," Bordas 1983.