CHAPITRE I : NOTIONS DE PARALLÉLISME

NOTION DE PROCESSUS

Définition

- Un processus est une entité dynamique qui représente l'exécution d'un programme sur un processeur.
- Le déroulement d'un processus est une suite d'actions élémentaires (instructions ou taches élémentaires).
- D'une manière informelle, un processus est un programme en cours d'exécution. Il est constitué de :
  - Un code exécutable du programme en exécution.
  - Un contexte (compteur ordinal, mot d'état PSW, registres, pile) qui est une image décrivant l'environnement du processus.

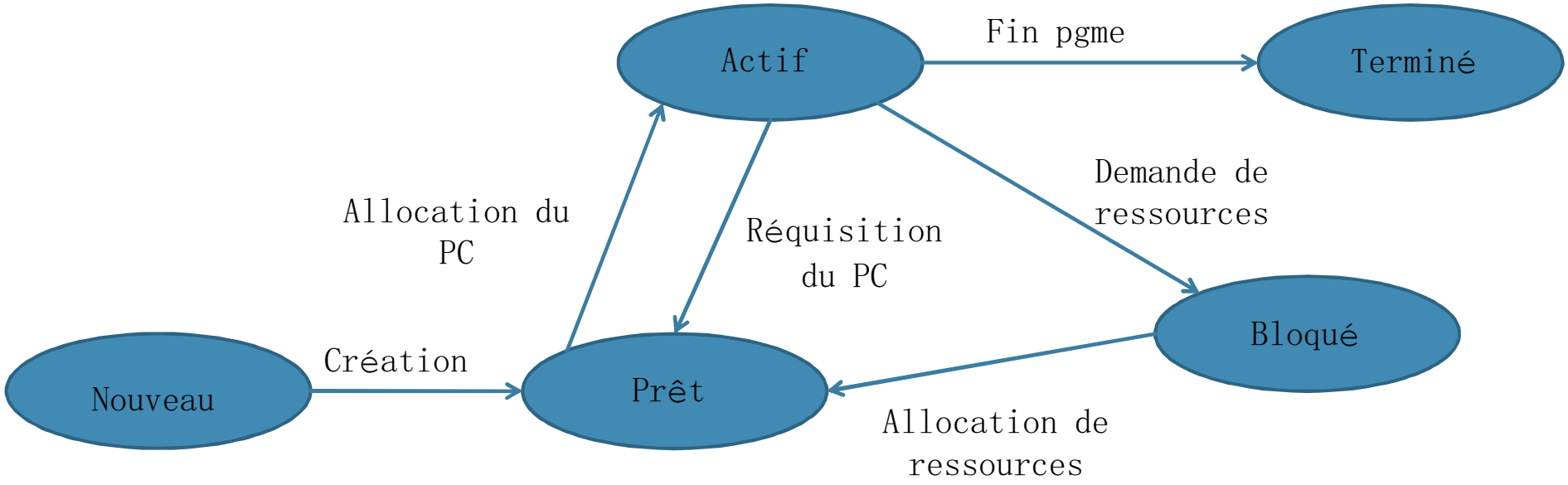
NOTION DE PROCESSUS

Etat d'un processus

- Trois principaux états d'un processus :
  - Prêt : le processus attend la libération du processeur pour s'exécuter.
  - Actif : le processus est en exécution.
  - Bloqué : le processus attend une ressource physique ou logique autre que le processeur pour s'exécuter (mémoire, fin d'E/S, …).

NOTION DE PROCESSUS

Transition des états d'un processus



NOTION DE TACHE

Définition

- Une tâche est une unité élémentaire de traitement ayant une cohérence logique.
- Si l'exécution du processus P est constituée de l'exécution séquentielle des tâches T_1, T_2,..., T_n, on écrit : $P = T_1\ T_2 \ldots T_n$.
- Exemple

P1 : Lire(X) T1

X ← X+1 T2 ⟹ Mov Ax, X T4

Ecrire(X) T3 Inc Ax T5

Mov X, Ax T6

T2 = T4 T5 T6

PROGRAMMES CONCURRENTS

- Un programme concurrent est un programme dont certaines instructions ne s'exécutent pas de manière séquentielle. → Parallélisme
- La concurrence implique deux modes d'exécution, le parallélisme et la séquentialité entre les tâches.
- Deux types de parallélisme:
  - Le parallélisme réel qui nécessite autant de processeurs physiques que de taches ou processus parallèles.
  - Le pseudo-parallélisme est un moyen de simuler le parallélisme réel sur un processeur unique, en entrelaçant les traces temporelles des différents processus(temps partagé).

PROGRAMMES CONCURRENTS

Exemple

Considérons l'évaluation de l'expression E=(A*B)/(C+D).

- Une évaluation séquentielle qui consiste à calculer dans l'ordre :

  A*B ; C+D puis (A*B)/(C+D)

- Une évaluation concurrente consiste à calculer simultanément :

  A*B et C+D puis (A*B)/(C+D)

- Une évaluation concurrente permet donc de réduire le temps total d'exécution.

SYSTÈMES DE TÂCHES

Définition

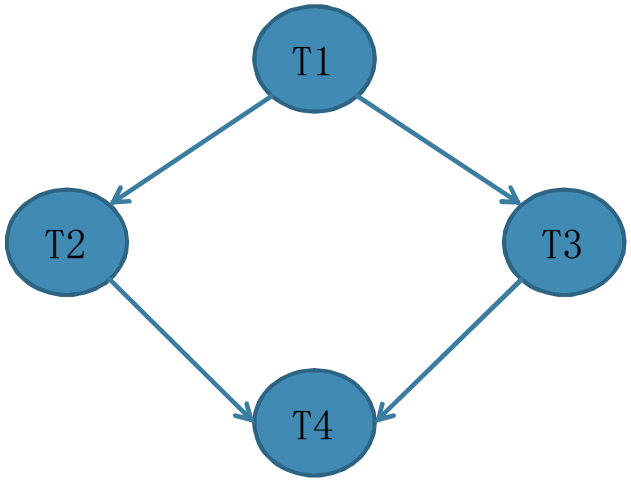
- On appelle système de tâches, qu'on note S = (E, <), un couple constitué d'un ensemble E de tâches et d'une relation de précédence "<" sur cet ensemble.
- Il définit l'ordre dans lequel les tâches du système doivent être exécutées.

COMPORTEMENT D'UN SYSTÈME DE TÂCHES

- L'exécution d'un système de tâches peut définir plusieurs comportements différents.
- Un comportement est le résultat d'un entrelacement (ou la linéarisation) lors de l'exécution des tâches du système.

COMPORTEMENT D'UN SYSTÈME DE TÂCHES

- Exemple




```
T1 : Lire(N);
T2 : N ← N + 2 ;
T3 : N ← N / 3 ;
T4 : Ecrire(N);
```

```
T2 : Mov Ax, N      (a)
     Add Ax, 2      (b)
     Mov N, Ax      (c)
T3:  Mov Ax, N      (a′)
     Div Ax, 3      (b′)
     Mov N, Ax      (c′)
```

- Comportement possibles:

```
T1; (a)(b)(c)(a′)(b′)(c′);T4
T1; (a′)(b′)(c′)(a)(b)(c);T4
T1;(a)(a′)(b)(b′)(c)(c′) ;T4
T1;(a′)(a)(b′)(b)(c′)(c) ;T4
. . .
```

DÉTERMINISME D'UN SYSTÈME DE TÂCHES

Définition

- Un système de tâches est dit déterministe si la suite des valeurs affectées à chaque variable est la même, pour chaque comportement possible.

CONDITIONS DE BERNSTEIN

- Deux tâches peuvent s'exécuter en parallèle sans risques d'incohérence (d'interférence) si et seulement si les règles de Bernstein suivantes sont respectées :

$$R(T1) \cap W(T2) = \varnothing$$
$$W(T1) \cap R(T2) = \varnothing$$
$$W(T1) \cap W(T2) = \varnothing$$

R : est l'ensemble de lecture (Read) qui définit l'ensemble des variables consultées par la tache.

W : est l'ensemble d'écriture (Write) qui définit l'ensemble des variables modifiées par la tache.

CONDITIONS DE BERNSTEIN : EXEMPLE

Soit le programme suivant:

T1 : Lire(N);
T2 : N ← N + 2 ;
T3 : N ← N / 3 ;
T4 : Ecrire(N);

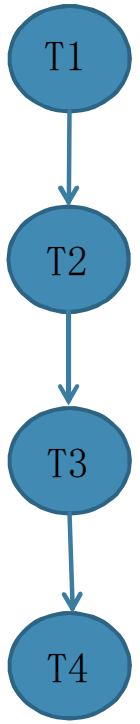
Déterminer les tâches qui peuvent être exécutées en parallèle.

Tâche	R	W
T1	∅	N
T2	N	N
T3	N	N
T4	N	∅

T1 doit précéder toutes les autres tâches car on ne peut pas utiliser la variable N avant son initialisation.

$R(T2) \cap W(T3) \neq \emptyset$
$W(T2) \cap R(T3) \neq \emptyset$ ➔ P2 précède P3
$W(T2) \cap W(T3) \neq \emptyset$
$R(T3) \cap W(T4) = \emptyset$
$W(T3) \cap R(T4) \neq \emptyset$ ➔ P3 précède P4
$W(T3) \cap W(T4) = \emptyset$

Graphe de précédence possible



EXERCICE

Supposons le système de taches suivants :

T1 : read(x) ; T2 : read(y) ; T3 : Z1= x+y ; T4 : Z2= x*y ; T5 : v= z1+z2 ; T6 : write (v)

Tracer le graphe de précédence (ou le graphe de parallélisme maximal) correspondant à ce système après avoir vérifier les conditions de Bernstein.

CORRIGÉ DE L'EXERCICE

T1 : read(x) ; T2 : read(y) ; T3 : Z1= x+y ; T4 : Z2= x*y ; T5 : v= z1+z2 ; T6 : write (v)

Tâche	R	W
T1	∅	x
T2	∅	y
T3	x, y	z1
T4	x, y	z2
T5	z1, z2	v
T6	v	∅

- T1 et T2 peuvent s'exécuter en parallèle car elles n'utilisent pas les mêmes variables (les conditions de Bernstein sont vérifiées).
- T3 et T4 peuvent s'exécuter en parallèle car elles utilisent pas les mêmes variables mais en lecture seule (les conditions de Bernstein sont vérifiées).
- T1 et T2 doivent précéder T3 et T4 car ces dernières ne peuvent pas utiliser les valeurs de variables x et y avant leur initialisation. Donc, T3 et T4 ne peuvent pas commencer leurs exécutions avant la terminaison de T1 et T2.
- T3 et T4 doivent précéder T5 car cette dernière utilise les résultats d'exécution de T3 et T4 (Une variable ne peut pas être utilisée avant son initialisation).
- T5 doit précéder T6 car cette dernière affiche le résultat de T5 (Une variable ne peut pas être utilisée avant son initialisation).

CORRIGÉ DE L'EXERCICE

Système de Tâches

S = { (T1, T2, T3, T4, T5, T6), T1 < T3

T1 < T4

T2 < T3

T2 < T4

T3 < T5

T4 < T5

T5 < T6}

Graphe de précédence