Chapitre 1 : Notions de parallélisme, de coopération et de compétition

Introduction

L'apparition des systèmes multiprogrammés vers la fin des années 1960 a révolutionné les méthodes de conception des applications. Les performances de celle-ci ont été renforcées par l'introduction, dans les programmes, de mécanismes de contrôle d'exécution permettant d'exploiter toutes les possibilités *d'activités parallèles*. Cela a donné naissance à de nouveaux modes d'exploitation comme *le temps partagé* ou *le temps réel.*

L'objectif du parallélisme est d'avoir des machines performantes soit en temps d'exécution soit en complexité de programmes.

1. Notion de Processus

La notion de processus constitue la pierre angulaire des systèmes d'exploitation. Un processus est décrit par une séquence d'actions produite par l'exécution d'une suite d'instructions. C'est une entité dynamique qui nait, qui vit et qui meurt. Pendant le fonctionnement du système d'exploitation, plusieurs processus s'exécutent en parallèle, en simultanéité réelle (sur une machine multiprocesseurs) ou apparente (exécution entrelacée de plusieurs processus sur un même processeur).

1.1. Notion de tâche

Une tâche est une unité élémentaire de traitement ayant une cohérence logique. Si ***P*** est le nom d'un processus, ***P= T_1 T_2 … T_n ;*** signifie que ***P*** est constitué de la suite des exécutions des tâches ***T_1 T_2 … T_n*** .

A chaque tâche ***T_i*** nous associons deux événements ***d_i*** et ***f_i*** :

- ***d_i*** : la tâche commence son exécution
- ***f_i*** : la tâche termine son exécution

L'initialisation d'une tâche est associée à la lecture des paramètres d'entrée, acquisition des ressources nécessaires à l'exécution de la tâche, etc.

La terminaison correspond à l'écriture des résultats, la libération des ressources, etc.

Exemple 1

Considérons la tâche **T : X=X/2**

L'initialisation (d) de ta tâche représente la lecture de la valeur de X. La terminaison (f) représente l'écriture de la valeur de X.

2. Système de tâches et graphe de précédence

Une relation de précédence sur un ensemble E, est une relation notée : **'<'**, qui vérifie :

1) Pour tout élément ***T*** de ***E***, on a pas ***T < T***
2) Pour tout couple ***(T_i , T_j)*** de ***E***, on a pas ***<u>simultanément</u> $T_i < T_j$ <u>et</u> $T_j < T_i$***
3) La relation **'<'** est transitive ; c-à-d ***si T_i , T_j, $T_k \in E$***. tel que $T_i < T_j$, **<u>et</u>** $T_i < T_k \rightarrow T_i < T_k$

On appel système de tâches le couple ***S = (E, <)*** constitué d'un ensemble ***E*** de tâches et d'une relation de précédence ***'<'*** sur cet ensemble.

Remarques

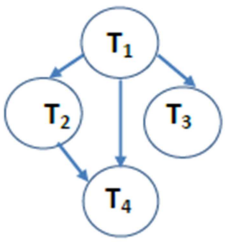
1. $T_i < T_j$: la terminaison de T_i doit obligatoirement intervenir avant l'initialisation de T_j
2. Si $T_i \not< T_j$ **et** $T_j \not< T_i$ → l'ordre dans lequel on effectue T_i et T_j n'est pas important. Par définition nous dirons que T_i et T_j sont ***parallèles ou indépendantes.***

Exemple 2

E= $\{T_1, T_2, T_3, T_4\}$
$T_1 < T_2$, $T_2 < T_4$, $T_1 < T_3$, $T_1 < T_4$

Graphe de précédence

L'ordre d'exécution des tâches d'un système de tâches peut être représenté schématiquement par un graphe orienté où les sommets représentent les tâches, et les arcs les relations entre elles.

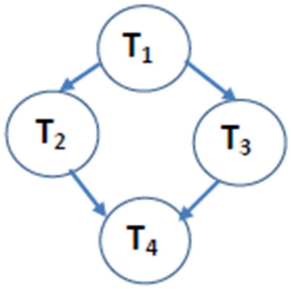


Représentation graphique d'un système de tâches

→ Le graphe est dit redondant à cause de l'arc (T_1, T_4),
→ ***Le graphe de précédence d'un système de tâches ne doit pas contenir d'arc redondant.***
→ T_1 est une tâche initiale.
→ T_4 est une tâche finale.
→ T_2 est le successeur immédiat de T_1.
→ T_2 et T_3 sont indépendantes.

3. Comportement d'un système de tâches

Soit le système de tâches représenté par le graphe de précédence suivant.



Les suites des exécutions ***d1f1d2d3f3f2d4f4, d1f1d3d2f2f3d4f4, d1f1d3f3d2f2d4f4*** satisfont bien aux contraintes du graphe de précédence ci-dessus. On appelle chacune des suites un comportement du système de tâches.

Définition

Un comportement d'un système de tâches est le résultat d'un entrelacement de l'exécution des différentes tâches qui le composent, toute en respectant les contraintes de précédences.

4. Déterminisme d'un système de tâches- Les conditions de Bernstein :

a. Domaine de lecture, domaine d'écriture d'une tâche

Dans un système de tâches, chaque tâche **T** utilise des variables en lecture et d'autres en écriture.
A chaque tâche est associée

- **Un domaine de lecture (L_T) :** ensemble des variables accédées en lecture par la tâche.
- **Un domaine d'écriture (E_T):** ensemble des variables accédées en écriture (modifiées) par la tâche.

Exemple 3

Soit le processus P suivant :
T1 : Lire (A) ;
T2 : Lire (B) ;
T3 : B=3*B ;
T4 : C=A+B ;
T5 : Ecrire (C) ;

$L_{T1} = \emptyset$ $E_{T1} = \{A\}$

$L_{T2} = \emptyset$ $E_{T2} = \{B\}$

$L_{T3} = \{B\}$ $E_{T3} = \{B\}$

$L_{T4} = \{A, B\}$ $E_{T4} = \{C\}$

$L_{T5} = \{C\}$ $E_{T5} = \emptyset$

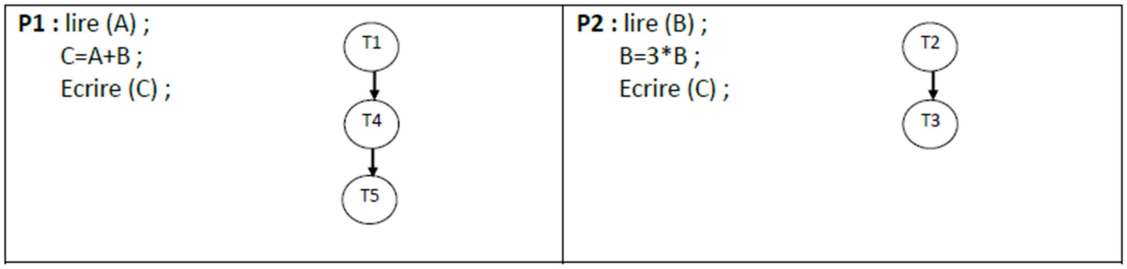
b. Définition

Soit S, un système de tâches, deux tâches T1 et T2 de S sont dites non interférentes par rapport à S, si:

1. Soit T1 est prédécesseur ou successeur de T2
2. Soit $L_{T1} \cap E_{T2} = L_{T2} \cap E_{T1} = E_{T1} \cap E_{T2} = \emptyset$

Exemple 4

Décomposons le processus P de l'exemple 3 en deux processus P1 et P2 avec P1 et P2 sont parallèles (P1 // P2).



Dans le système P1 // P2, les tâches T3 et T4 sont interférentes.

Théorème

Soit S un système de tâches. Si S est constitué de tâches **deux à deux non interférentes**, alors il est **déterministe** quelque soient les fonctions calculées par les tâches.

5. Le multithreading

Un thread (appelé aussi processus léger ou fil d'exécution), est similaire à un processus car tous deux représentent d'exécution d'un ensemble d'instructions du langage machine d'un processeur. Toutefois, là où chaque processus lourd (créé par l'appel à la primitive *fork()*) possède sa propre mémoire virtuelle, les threads appartenant à un même processus père se partagent sa mémoire virtuelle, sa section de code, sa section de données et d'autres ressources systèmes. Par contre, tous les processus légers possèdent leur propre pile système.

Un thread comprend :

- Un identificateur
- Un compteur de programme
- Un ensemble de registre
- Et une pile

Un thread partage avec le processus créateur les variables globales et les descripteurs de fichiers.

Utilisation des threads

Les threads sont typiquement utilisés avec l'interface graphique (GUI) d'un programme. En fait, les interactions de l'utilisateur avec le programme ou l'application sont gérées par les threads, tandis que les calculs lourds (en termes de temps de calcul) sont gérés par un ou plusieurs processus. Cette technique de conception de logiciel est avantageuse dans ce cas, car l'utilisateur peut continuer d'interagir avec le programme même lorsque celui-ci est en train d'exécuter une tâche. Une application pratique se retrouve dans les traitements de texte où la correction orthographique est exécutée tout en permettant à l'utilisateur de continuer à enter son texte.

L'utilisation des threads permet donc de rendre l'utilisation d'une application plus fluide car il n'y a pas de blocage durant les phases de traitements intenses.