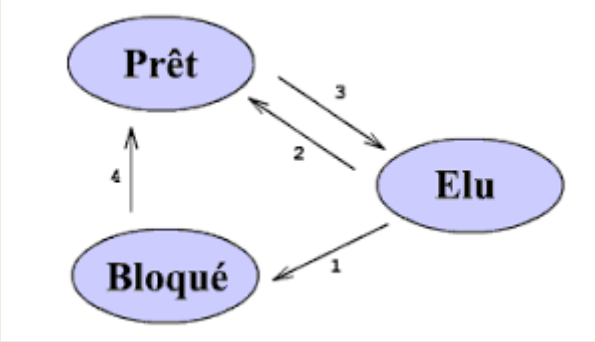
PROGRAMMATION CONCURRENTE

DEFINITIONS ET CONCEPTS

- ✓ La **programmation concurrente** est un paradigme de programmation tenant compte, dans un programme, de l'existence de plusieurs piles sémantiques qui peuvent être appelées threads, processus ou tâches. Elles sont matérialisées en machine par une pile d'exécution et un ensemble de données privées.

Concurrent	Non concurrent
X= a*b*c Y= 3*a+7	X= 5 Y= 2*X+4

- ✓ **Processus**, correspond à un traitement (calcul) séquentiel, muni de son thread de contrôl



DEFINITIONS ET CONCEPTS

✓ **Classification**

On distingue trois types de concurrence :

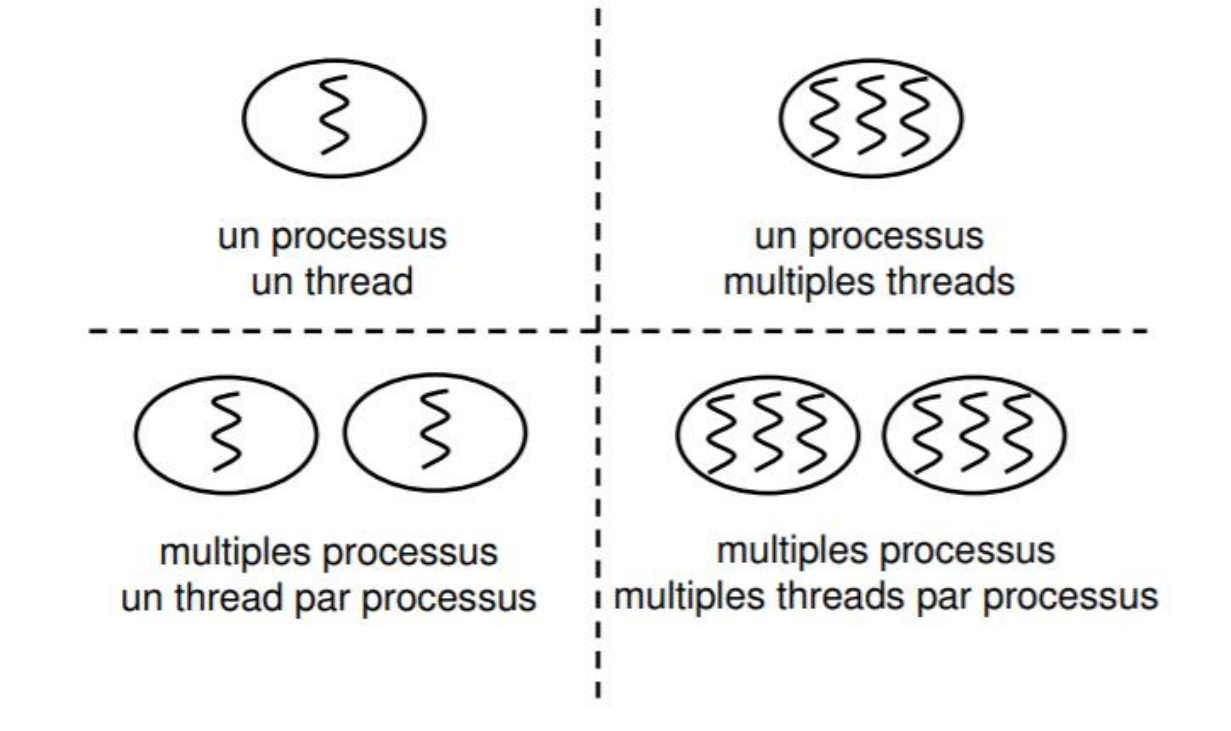
disjointe : les entités concurrentes ne communiquent pas et n'interagissent pas entre elles ;

compétitive : un ensemble d'entités concurrentes en compétition pour l'accès à certaines ressources partagées (par exemple le temps CPU, un port d'entrées/sorties, une zone mémoire) ;

coopérative : un ensemble d'entités concurrentes qui coopèrent pour atteindre un objectif commun. Des échanges ont lieu entre les processus. La coopération est un élément primordial de la programmation concurrente.

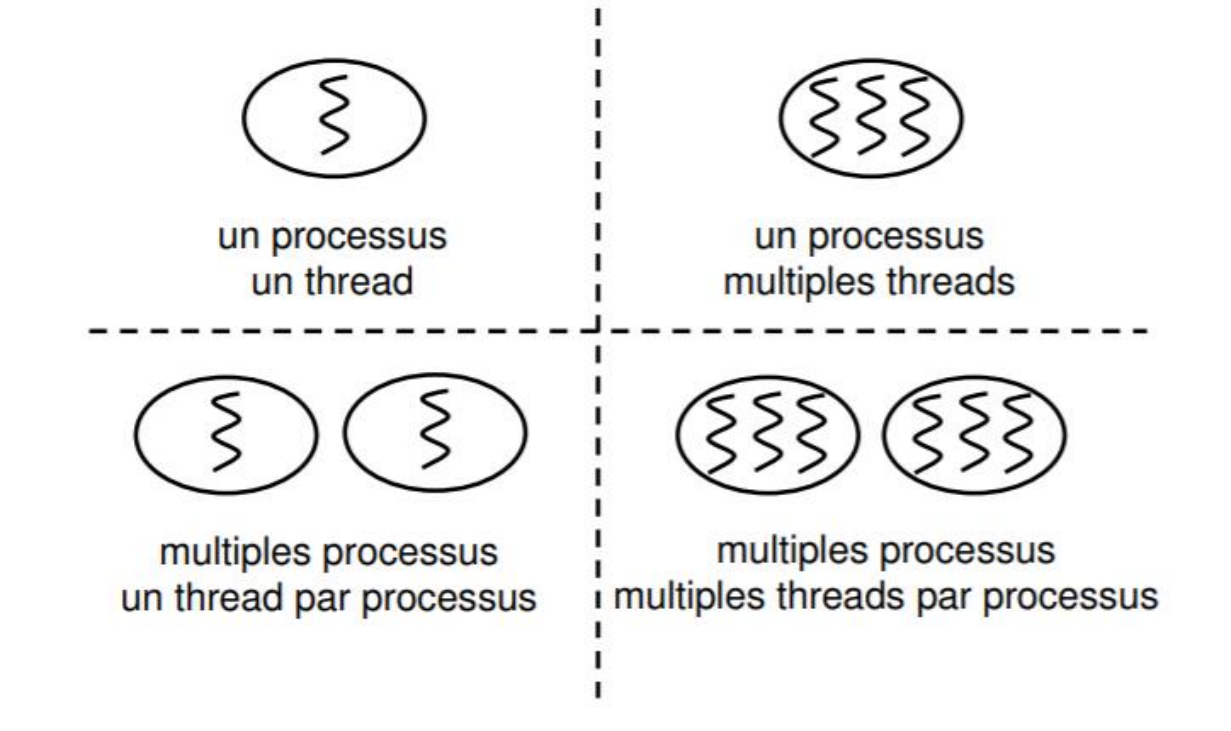
DEFINITIONS ET CONCEPTS

Le mot « ***thread*** » est un terme anglais qui peut se traduire par « *fil d'exécution* ». L'appellation de « *processus léger* » est également utilisée.



DEFINITIONS ET CONCEPTS

Le mot « ***thread*** » est un terme anglais qui peut se traduire par « *fil d'exécution* ». L'appellation de « *processus léger* » est également utilisée.



DEFINITIONS ET CONCEPTS

❖ **Système monoprocesseur**

✓ un processus pour chaque organe d'entrée ou sortie : lecteur de cassette, imprimante, clavier, écran

✓ un ou plusieurs processus pour le traitement de base

❖ **Système multiprocesseur ou repartis**

✓ un processus au moins par processeur

✓ sites clients et sites serveurs ou gestionnaires d'objets

PROGRAMMATION ET ORDONNANCEMENT DES ACTIONS

- **Solution par programmation séquentielle impérative**

On détermine dès la programmation l'ordre total de l'exécution

- **Solution séquentielle impérative avec non-déterminisme**

On laisse des choix pour l'exécution

- **Programmation séquentielle déclarative**

On déclare les contraintes de causalité ou de précédence

- **Programmation concurrente**

On déclare des processus séquentiels (programmes séquentiels impératifs) et des relations de synchronisation entre eux (ordre partiel)

EXEMPLES D'APPLICATION CONCURRENTES

- Centre de calcul ou centre de ressources accessibles aux filiales ou aux agences, par télétraitement ou via un réseau
- Station de travail avec processeurs parallèles (CAO de circuits, traitement d'images, Systèmes d'Information Géographique, serveur de Bases de Données pour transactions)
- Systèmes de réservation de places (hôtels, avions, train, concerts)
- SGBD avec accès multiple par des guichets ou des GAB (banque, PTT, assurances, mutuelles)
- Systèmes embarqués de conduite de mobiles (avion, voiture, train)
- Centraux téléphoniques interconnectés, avec valeur ajoutée
- Services coopératifs : annuaires, mirroirs, caches Web
- Diffusion Multimedia ("multicast"), synchronisation voix et image

Concepts de base de la concurrence et du parallélisme

❖ **Processus vs. Thread**

Tant les **processus** que les **threads** représentent des instances de code en cours d'exécution — donc une notion dynamique. Toutefois, on établit souvent les distinctions suivantes entre **processus** et **thread:**

Processus :

– Un processus représente un **programme** en cours d'exécution.

– Un processus possède un espace mémoire **privé**, indépendant de celui des autres processus.

– Parce qu'ils ne partagent aucun espace mémoire commun, deux processus qui veulent collaborer doivent le faire en utilisant un mécanisme **d'échange de messages**

— par exemple, pour des processus Unix, en utilisant des pipes.

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

✓ **Programme séquentiel**

= Un programme qui comporte un seul ***fil d'exécution*** — un seul «***thread***»

⇒ Un seul doigt suffit pour indiquer l'instruction en cours d'exécution

✓ **Programme concurrent**

= Un programme qui contient ***deux ou plusieurs threads*** qui coopèrent

⇒ Plusieurs doigts sont nécessaires pour indiquer les instructions en cours d'exécution

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

Coopération ⇒ Communication, échange d'information

Deux principales façons de communiquer :

– Par l'intermédiaire de variables partagées — principalement dans le cas de threads

– Par l'échange de messages et de signaux — par ex., via des pipes dans le cas de processus Unix

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

– **Différents types d'applications concurrentes**

- Application **multi-contextes** (**multi-threaded**) = contient deux ou plusieurs *threads*, qui peuvent ou non s'exécuter en même temps, et qui sont utilisés pour mieux organiser et structurer l'application (meilleure modularité)

Exemples : Système d'exploitation multi-tâches, fureteurs multi-tâches, interface personne-machine vs. traitement de la logique d'affaire, serveur Web

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

– Différents types d'applications concurrentes

- Application **parallèle** = chaque thread s'exécute sur son propre processeur (ou coeur), dans le but de résoudre plus rapidement un problème — ou pour résoudre un problème plus gros

Exemples : Prévisions météorologiques, modélisation du climat, simulations physiques, bio-informatique, traitement graphique, etc.

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

– Différents types d'applications concurrentes

- Application **distribuée** = contient deux ou plusieurs processus, qui communiquent par l'intermédiaire d'un réseau (⇒ délais plus longs), et ce pour répartir, géographiquement, des données et des traitements

Exemples : Serveurs de fichiers, accès à distance à des banques de données

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

Autre caractérisation pour distinguer concurrence et parallélisme : Programme CPU-bound vs. Programme IO-bound

Une caractéristique intéressante qui permet de comprendre la différence entre exécution concurrente et exécution parallèle est celle de programme ***CPU-bound vs. IO-bound :***

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

CPU-bound : un programme est *CPU-bound* si son temps d'exécution est limité (contraint) par la vitesse du CPU.

On peut donc supposer qu'un programme CPU-bound passe la majeure partie de son temps d'exécution à utiliser le CPU — il est gourmand en CPU — et **donc il s'exécuterait plus rapidement si le CPU était plus rapide.**

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

IO-bound : un programme est **IO-bound** si son temps d'exécution est limité (contraint) par la vitesse du système d'entrées/sorties (accès disques, accès réseaux, etc.).

Un programme **IO-bound** passe donc la majeure partie de son temps d'exécution à utiliser les E/S — il est gourmand en E/S — et donc ***il s'exécuterait plus rapidement si le système d'E/S était plus rapide.***

Concepts de base de la concurrence et du parallélisme

- Concurrence vs. Parallélisme

Soit deux machines multi-coeurs M_1 et M_8, ayant des configurations semblables sauf pour le nombre de coeurs : un seul (1) coeur pour M_1 et huit (8) coeurs pour M_8.

1. Soit un programme P_1 qui est *IO-bound* et qui s'exécute sur M_1 en 1.6 secondes.

   Si on exécute P_1 sur M_8, quel temps d'exécution peut-on s'attendre à obtenir?

2. Soit un programme P_2 qui est *CPU-bound* et qui s'exécute sur M_1 est 1.6 secondes.

   Si on exécute P_2 sur M_8, quel temps d'exécution peut-on s'attendre à obtenir?

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

Remarques additionnelles :

- Les catégories qui précèdent ne sont pas mutuellement exclusives. Par exemple, de nombreuses machines parallèles modernes, qu'on utilise essentiellement pour développer des applications parallèles, sont des multi-ordinateurs, donc des machines composées d'un ensemble de processeurs (avec mémoire distribuée) interconnectés par un réseau. On programme souvent ces machines avec des langages de programmation où l'on doit tenir compte de la distribution (principalement des données) et où les échanges d'information se font par l'envoi de messages.

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

Remarques additionnelles :

❖ Dans le cadre du cours, nous traiterons principalement d'applications parallèles, puis d'applications concurrentes, mais pas d'applications distribuées. De plus, nous traiterons principalement de la programmation concurrente et parallèle avec mémoire partagée

Concepts de base de la concurrence et du parallélisme

❖ Concurrence vs. Parallélisme

Remarques additionnelles :

❖ Dans le cours, nous utiliserons les termes suivants :

– ***Programmation parallèle*** = Programmation pour développer une application parallèle et l'exécuter sur machine parallèle. Dans cette forme de programmation, l'objectif sera d'obtenir un programme qui, si possible, s'exécute plus rapidement.

– ***Programmation concurrente*** = Programmation pour développer une application concurrente, plus généralement une application multi-contextes, et l'exécuter sur machine arbitraire — séquentielle, multi-coeurs, multiprocesseurs, etc.

Dans ce cas, l'objectif sera typiquement de développer un programme comportant plusieurs threads qui ***partagent des ressources de façon correcte et efficace***.

Concepts de base de la concurrence et du parallélisme

- **Tâches, dépendances entre tâches et graphes de dépendances**

Pour paralléliser un algorithme, il faut identifier toutes les tâches possibles, mais aussi identifier **leurs dépendances**, pour déterminer **ce qui peut, ou non, se faire en parallèle.**

Au niveau algorithmique, on commence par identifier les tâches les plus fines possibles (granularité aussi fine que nécessaire en fonction du problème : voir plus bas), sans tenir compte des ressources ou contraintes de la machine — en d'autres mots, on suppose une machine «idéale» avec des ressources illimitées.

Concepts de base de la concurrence et du parallélisme

- **Tâches, dépendances entre tâches et graphes de dépendances**

Ensuite, on détermine les dépendances entre ces tâches : si une tâche T_1 doit s'exécuter avant une tâche T_2, par exemple parce que T_1 produit un résultat utilisé par T_2, alors on dit qu'il y a une dépendance (de données) de T_1 vers T_2. On peut ainsi construire un graphe des dépendances de tâches. Ce graphe ne permet ensuite de mieux comprendre le comportement de l'algorithme parallèle.

Concepts de base de la concurrence et du parallélisme

- **Tâches, dépendances entre tâches et graphes de dépendances**

C'est souvent aussi à partir de ce graphe qu'on pourra développer ***un programme parallèle efficace***, qui tiendra compte des ressources matérielles — par exemple, pour diminuer les coûts de synchronisation et communication, on pourra décider de combiner ensemble des petites tâches pour obtenir des tâches plus grosses. On traitera de ces questions dans un chapitre ultérieur.

Concepts de base de la concurrence et du parallélisme

- **Tâches, dépendances entre tâches et graphes de dépendances**

Un exemple de graphes de dépendances des tâches : le calcul des racines d'un polynôme de 2e degré:

$$p(x) = ax^2 + bx + c$$

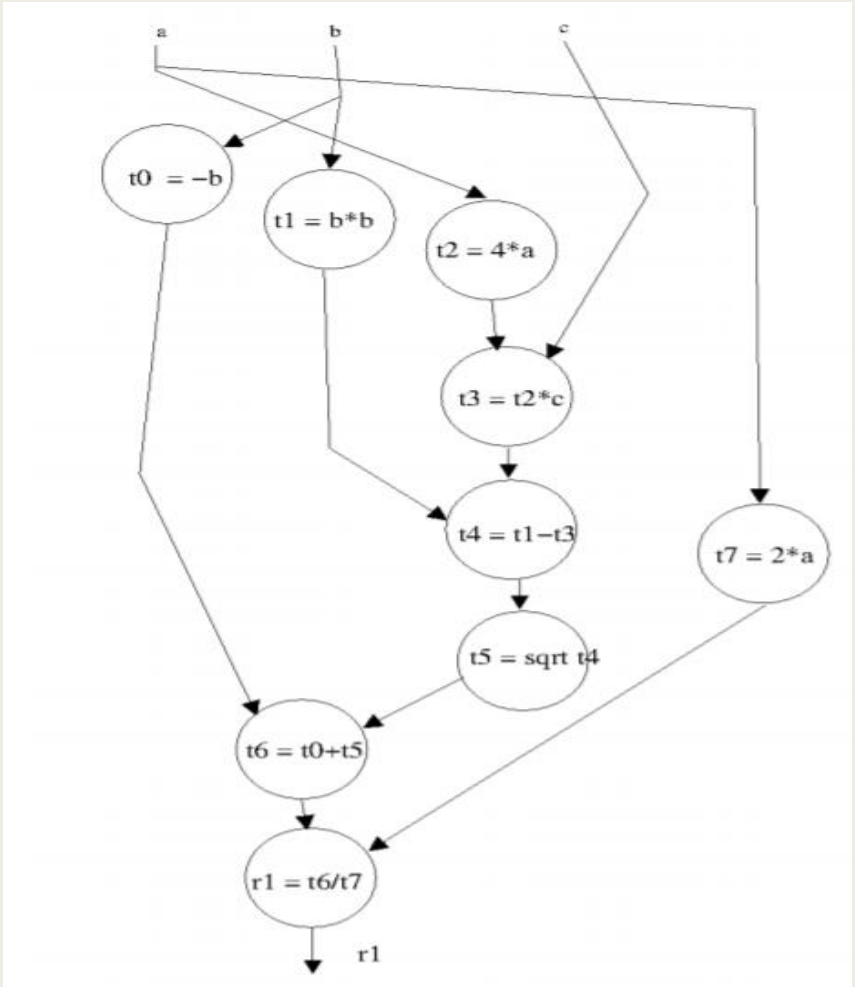
Concepts de base de la concurrence et du parallélisme

- **Tâches, dépendances entre tâches et graphes de dépendances**

Pour cet exemple, nous allons décomposer le calcul de r_1 en une suite d'instructions simples dites «à trois adresses», soit deux opérandes, une destination pour le résultat et un opérateur — donc des pseudo-instructions machines ou du pseudo code-octet. Ce seront alors ces instructions qui représenteront nos tâches à paralléliser. Remarque : En pratique, on ne procèdera généralement pas à une décomposition aussi fine des instructions. Ici, on le fait pour illustrer plus clairement les notions de dépendances et de graphes de dépendances

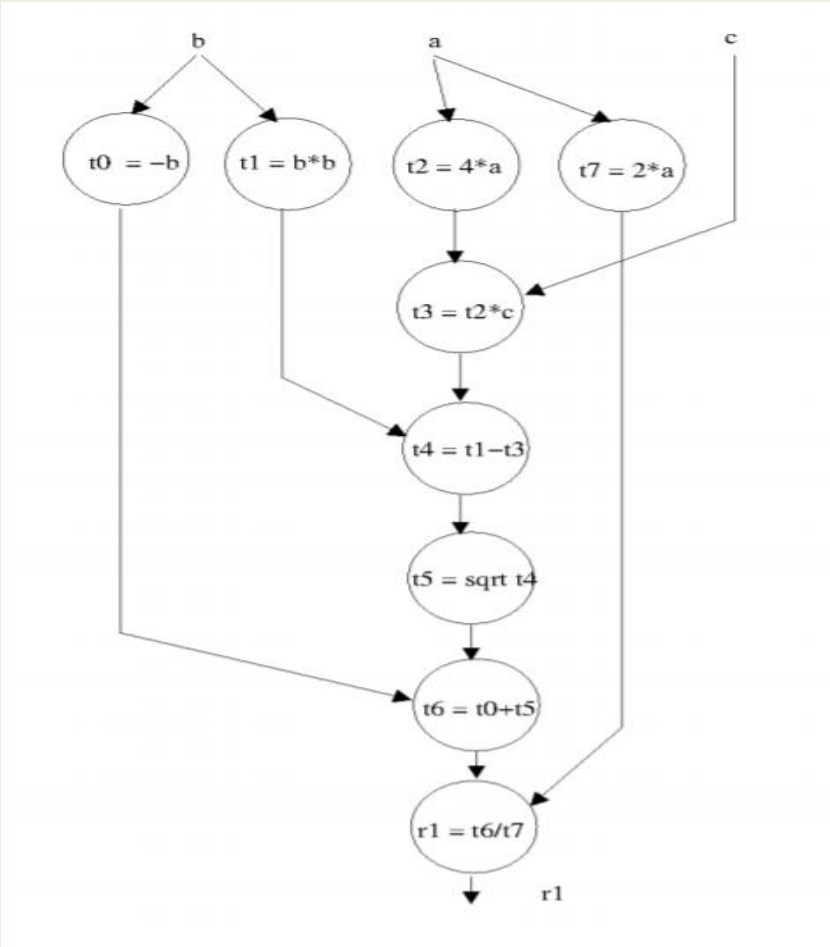
Concepts de base de la concurrence et du parallélisme

- Tâches, dépendances entre tâches et graphes de dépendances



Concepts de base de la concurrence et du parallélisme

- Tâches, dépendances entre tâches et graphes de dépendances



Concepts de base de la concurrence et du parallélisme

❖ Degré de parallélisme

Le **degré de parallélisme** d'un algorithme parallèle à un instant donné est le nombre de tâches qui peuvent être exécutées en même temps à cet instant. Ce nombre peut évidemment varier en cours d'exécution.

La performance d'un programme parallèle dépend de son degré de parallélisme et du nombre de processeurs de la machine : si la machine possède plus de processeurs que le degré de parallélisme, alors le programme pourra s'exécuter à sa plus grande vitesse parallèle.

Concepts de base de la concurrence et du parallélisme

- Degré de parallélisme

Degré de parallélisme du graphe de dépendance de calcul de r1

Temps	Degré
0	4
1	1
2	1
3	1
4 5	1 1

Concepts de base de la concurrence et du parallélisme

❖ Degré de parallélisme

Une autre notion intéressante pour comprendre le comportement d'un programme parallèle est celle de «***degré moyen de parallélisme***» = nombre moyen de tâches qui peuvent être exécutées en parallèle tout au long des différents moments de l'exécution. Ainsi, dans notre exemple, le degré moyen de parallélisme serait le suivant:

$$\frac{4+1+1+1+1+1}{6} = \frac{9}{6} = 1.5$$

Concepts de base de la concurrence et du parallélisme

❖ Degré de parallélisme

Lorsque ce degré moyen de parallélisme est ***semblable*** au degré maximum, alors cela implique que les ressources de la machine pourront être utilisées de façon **efficiente** tout au long de l'exécution du programme. Inversement, si l'écart est grand, alors à certains instants on aura besoin d'un grand nombre de processeurs, alors qu'à d'autres moments on aura besoin de peu de processeurs — donc soit on «***acquiert***» un grand nombre de processeurs et plusieurs resteront parfois ou souvent inutilisés, soit on utilise moins de processeurs, ce qui augmentera le temps d'exécution puisque certaines tâches indépendantes ne pourront pas être exécutées de façon parallèle.

Concepts de base de la concurrence et du parallélisme

❖ Longueur du chemin critique et temps d'exécution parallèle idéal

Un **chemin** dans un graphe entre deux sommets **S1** et **S2** est une série de sommets et d'arcs qui permettent d'aller de **S1** à **S2**.
La **longueur** d'un chemin est le nombre d'arcs traversés par ce chemin.

Dans un graphe de dépendances des tâches, le plus long chemin allant de la tâche initiale à la tâche finale est appelé le ***chemin critique***. C'est cette longueur — dite **longueur du chemin critique** (*critical path length*) — qui détermine le **meilleur temps d'exécution possible**.

Concepts de base de la concurrence et du parallélisme

❖Temps d'exécution parallèle idéal

Le ***temps d'exécution parallèle minimum*** — ou ***idéal*** — d'un algorithme parallèle est obtenu pouvant être obtenu en utilisant autant d'unités d'exécution que nécessaire — donc en supposant une machine idéale, sans limite sur le nombre d'UE. Ce temps minimum idéal est simplement égal à ***1+la longueur du chemin critique*** du graphe de dépendance des tâches.

Concepts de base de la concurrence et du parallélisme

❖ Indépendance entre threads

L'ensemble de lecture (*read set*) d'une partie de programme est l'ensemble des variables lues, mais non modifiées, par cette partie de programme.

L'ensemble d'écriture (*write set*) d'une partie de programme est l'ensemble des variables modifiées par cette partie de programme.

Deux parties de programme sont **indépendantes** si l'ensemble d'écriture de chaque partie est indépendante (*l'intersection est vide*) tant de l'ensemble de lecture que de celui d'écriture de l'autre partie.