Plan

- Introduction & définitions
- Mécanismes de base
- **Notions complémentaires**

Synchronisation : Pistes d'approfondissement

- Autres mécanismes de base
- Problèmes liés à la synchronisation
- Mémoires transactionnelles
- Retour sur les critères de progrès

Autres mécanismes élémentaires de synchronisation

- **Verrous lecteurs-rédacteurs (reader-writer locks)**
  - Observation : des accès concurrents à une donnée en lecture seule n'introduisent aucun conflit (= aucun risque)
  - Idée : différencier les demandes d'accès
    - Accès en lecture : possible malgré les accès concurrents (éventuels) d'autres lecteurs
    - Accès en écriture : un rédacteur doit manipuler les données en exclusion mutuelle (autre lecteurs ou rédacteurs interdits)
- **Barrières de synchronisation**
  - Scénario : N flots (threads/processus) concurrents (par exemple pour diviser une phase de calcul en N tâches.
  - Lorsqu'un flot a terminé sa tâche, il appelle la fonction *barrière*. Les (N-1) premiers flots sont bloqués lors de l'appel, le dernier flot débloque les autres et permet ainsi de passer à la phase suivante.
- **Ces mécanismes seront étudiés en TD/TP**

Problèmes liés à la synchronisation (1)
Inversion de priorité

- **Un processus P1 de haute priorité peut se retrouver bloqué en attente d'une ressource détenue (exclusivement) par un processus P2 de plus basse priorité**
- **Pire : P2 peut être préempté par un processus P3 de priorité intermédiaire (et ainsi de suite) => la durée du blocage de P1 est augmentée (voir infinie)**
- **Conséquences possibles**
  - Mauvaises performances, mauvaise réactivité du système
  - Dysfonctionnement (problèmes en cascades) – Exemple : robot Mars Pathfinder
  - Problème important dans les systèmes temps-réel
- **Solutions possibles**
  - N'utiliser que 2 niveaux de priorités
  - Elévation (temporaire) de la priorité d'un processus (ici P2)
    - Par héritage (en fonction des processus en attente)
    - Par tirage aléatoire (parmi le processus qui détiennent des verrous)

Problèmes liés à la synchronisation (2) Interblocages (deadlocks)

- **Définition du problème déjà vu dans le cours de bases de données**
- **Deux types d'approches pour gérer le problème**
  - Prévention
  - Détection et réaction
- **Prévention**
  - Technique 1 : Réservation globale des ressources à l'avance
  - Technique 2 : Respecter un ordre global sur les demandes de ressources
  - Technique 3 : Suivre les dépendances et empêcher la formation de cycles (refuser la demande fatale)
- **Détection**
  - Laisser l'interblocage survenir, le détecter et tuer un processus impliqué
  - Attention : contrairement aux SGBD, les systèmes d'exploitation généralistes n'ont pas de support intégré pour le « rollback » d'une tâche
- **Problème : ces solutions ne sont pas toujours applicables dans un système généraliste avec des applications arbitraires**
- **Variante du problème : livelocks**

Mémoires transactionnelles

- **Idée inspirée de la notion de *transaction* dans le domaine des bases de données**
  - Le programmeur doit simplement indiquer les frontières (début/fin) des sections critiques (= transactions)
  - Le programmeur n'a pas à déclarer les mécanismes utilisés (verrous …)
  - Le « système » sous-jacent se charge de gérer les problèmes de concurrence (généralement de manière optimiste)
- **Différentes niveaux d'implémentation possibles**
  - Logiciel, Matériel, Hybride
  - Support encore limité en production
- **Bénéfices**
  - Simplicité, composition aisée et correcte de code
- **Limitations**
  - Performances (pour certaines configurations)
  - Gestion des blocs de code ayant des effets de bord non annulables

Compléments sur la notion de *progrès*

- **Lors de la définition de la notion de section critique, nous avons vu défini la notion de *progrès*.**
- **En fait, selon les garanties fournies par un algorithme de synchronisation donné, on peut distinguer différents critères de progrès.**

- **Algorithmes bloquants (*blocking*):**
  - (à ne pas confondre avec les notions d'attente passive/active)
  - Un problème/retard/délai d'un thread peut éventuellement empêcher les autres threads de progresser
  - Les primitives de base que base que nous avons étudiées rentrent dans cette catégorie

- **Algorithmes non bloquants (*non-blocking*)**
  - Il existe différentes sous-catégories :
    - *Obstruction-free*, *Lock-free*, *Wait-free*

Compléments sur la notion de *progrès (suite)*

- ***Obstruction freedom*:** l'algorithme permet à un thread exécuté en isolation (c'est-à-dire pendant que tous les autres sont suspendus) pendant un nombre fini d'étapes de mener à bien son opération.

- ***Lock freedom*:** l'algorithme permet à au moins un thread de mener à bien son opération en un nombre fini d'étapes. Il est possible que certains threads doivent abandonner/réessayer leur opération.

- ***Wait freedom*:** l'algorithme garantit que chaque opération effectuée par chaque thread sera menée à bien en un nombre fini d'étapes.

Compléments sur la notion de *progrès (suite)*

- **Les algorithmes non bloquants ne sont pas sujet à des risques d'interblocages (deadlocks/livelocks) ou d'inversion de priorité.**

- **En matière de garanties de progrès, on a la hiérarchie suivante : blocking < obstruction free < lock free < wait free**
  - Cependant, en pratique, cette hiérarchie ne se transpose pas forcément au niveau des performances observées. Certains algorithmes non bloquants (en particulier *wait free*) peuvent être inefficaces.