Threads et programmation concurrente

IFT 3880-IFT 6835 APPLICATIONS DISTRIBUÉES

Par Laurent Magnin

Université de Montréal

Applications distribuées ?

- **Définition : application (logiciel) fonctionnant en parallèle sur plusieurs ordinateurs**
- **Pourquoi ?**
  - Problème de fait distribué
    - Client / serveur, Internet
  - Optimisation
    - Calcul réparti
  - Sécurité & robustesse
    - « FireWall », redondance, etc.
  - Partage de ressources

Programmation concurrente

- **Classiquement : un programme s'exécute de façon linéaire**
- **Programmation concurrente : plusieurs blocs d'instructions peuvent s'exécuter en parallèle**
  - Processus (*Process*) : lancement en parallèle de plusieurs logiciels sur la même machine (gestion par le système d'exploitation). Nécessite son propre espace d'adresse mémoire.
  - « *Thread* » : exécution en parallèle de plusieurs blocs d'instruction au sein du même logiciel. Beaucoup plus léger qu'un processus

Pourquoi une programmation // ?

- **Permet de faire plusieurs choses en même temps**
  - Ex : écouter de la musique tout en utilisant un traitement de texte
  - Interface réactive avec traitements en arrière plan
- **Programmation plus facile**
  - Ex : un serveur dans un schéma « client - serveur »
- **Exécution plus efficace**
  - Calcul scientifique sur machine multi-processeur
- **Applications réparties**
  - Certains principes restent identiques
  - Nécessite une gestion multi-thread

Coopératif ou préemptif ?

- **Sur une machine mono-processeur, une seule instruction peut être exécutée à la fois…**
- **A quel bloc d'instruction doit appartenir la prochaine instruction ?**
  - Au même bloc, sauf s'il décide de « rendre la main »
    - Système coopératif
  - C'est le système de gestion qui décide
    - Système préemptif (*time-slicing*)
- **Prise en compte des interruptions**
  - Fonctions de bas niveau générées au niveau matériel

« Sauvegarder le contexte » ?

- **Lorsqu'un programme s'exécute, le processeur à accès à :**
  - Un compteur ordinal (pointe vers la prochaine instruction à exécuter)
  - Une pile (sauvegarde de tous les appels récursifs de fonctions)
  - Un tas (sauvegarde des valeurs des variables globales)
- **Chaque *thread* dispose des mêmes éléments (appelé contexte)**
- **Lorsque l'on permute de *thread***
  - Sauvegarde du contexte du thread mis en sommeil
  - Rétablissement du contexte du nouveau thread

Système « réentrant » ?

- **La même fonction ou méthode peut être exécutée plusieurs fois en parallèle**
- **Sinon, il faut attendre que la première exécution de la méthode se termine pour en relancer une autre (avec un autre ensemble de paramètres)**
  - Arrive si des méthodes utilisent des variables globales (communes)
  - N'interdit pas que d'autres méthodes indépendantes s'exécutent en parallèle

Processus et systèmes d'exploitation

- **Processus en Unix**
  - Bloquant
    - >commande
  - Détaché
    - >commande&
- **Sous DOS**
  - Plusieurs processus possibles, mais un seul actif à la fois
- **Les processus sous Windows NT/2000/XP**
  - http://www.inoculer.com/processus.php3
- **Macintosh « classique » (OS 9)**
  - Système coopératif

Les *threads* en Java

- **De base, Java supporte les threads**
  - Soit, gestion par la machine virtuelle (cas le plus courant)
  - Soit, un thread = un processus du système d'exploitation
- **Coopératif ou préemptif ?**
  - Cela dépend…
  - Considérer un système coopératif !

Thread & Runnable

- **La classe `Thread`**
  - Permet d'instancier (créer) un *thread* Java
- **L'interface `Runnable`**
  - Permet à la méthode run() d'un objet d'être intégrée à un *thread* Java

Les méthodes associées à Thread

- **Constructor Thread(String nom)**
- **getName()**
- **Abstract public void Run()**
- **Start()**
- **Stop()**
- **Suspend() & Resume()**
- **SetPriority() & GetPriority()**
- **Sleep(long millisecondes)**
- **getThreadGroup()**

ThreadGroup

- **Permet de regrouper un ensemble de threads**

Référence(s)

- ***JAVA Threads*, Scott Oaks & Henry Wong, ed. O'Reilly**

Démo

- **http://www.iro.umontreal.ca/~pift3880/sources/02-thread/**