Programmation orientée objet en langage JAVA

Les processus légers : *thread*

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/∼duvallet/

Les processus légers

Introduction aux Thread Java (1/2)

- Les threads Java peuvent être créés par :
  - Extension (`extends`) de la classe Thread
  - Implémentation (`implements`) l'interface Runnable
- Exemple 1 en étendant la classe `Thread` :

```
public class Thread1 extends Thread {
    public void run() {
        System.out.println("Je suis un Thread JAVA");
    }
}

public class Principale {
    public static void main(String args[]) {
        Thread1 th = new Thread1();
        th.start();
        System.out.println("Je suis dans la fonction principale");
    }
}
```

Introduction aux Thread Java (2/2)

- Exemple 2 en utilisant l'interface `Runnable` :

```
public class Thread2 implements Runnable {
    public void run() {
        System.out.println("Je suis un Thread JAVA");
    }
}

public class Principale {
    public static void main(String args[]) {
        Runnable run = new Thread2();

        Thread th = new Thread (run);
        th.start();

        System.out.println("Je suis dans la fonction principale");
    }
}
```

- Pourquoi 2 méthodes ? Absence d'héritages multiples

```
public class ThreadApplet extends JApplet implements Runnable { ... ... ... }
```

Les outils des Thread Java

- `suspend()` suspend l'exécution du thread en train d'être éxecuté. Cette méthode a été dépréciée et son utilisation est à prohiber.
- `resume()` relance l'exécution d'un thread suspendu. Cette méthode est aussi dépréciée.
- `sleep()` endort pour un certain temps le thread en train d'être exécuté.
- `join()` permet d'attendre la fin de l'exécution d'un Thread. Il peut servir à positionner des points de synchronisation.
- `interrupt()` arête l'exécution du thread.

Producteurs/Consommateurs (1/4)

Le serveur :

```
public class Serveur {
    public Serveur() {
        MessageQueue boiteMail = new MessageQueue();
        Producteur threadProducteur = new Producteur(boiteMail);
        Consommateur threadConsommateur = new Consommateur(boiteMail);
        threadProducteur.start();
        threadConsommateur.start();
    }

    public static void main(String args[]) {
        new Serveur();
    }
}
```

Producteurs/Consommateurs (2/4)

Le producteur :

```
import java.util.Date;

class Producteur extends Thread {
    private MessageQueue boitec;

    public Producteur(MessageQueue m) {
        boitec = m;
    }

    public void run() {
        while (true) {
            // produit un élément et le place dans la boite
            Date message = new Date();
            System.out.println ("Message envoyé : "+message);
            boitec.send(message);
        }
    }
}
```

Producteurs/Consommateurs (3/4)

Le consommateur :

```
import java.util.Date;

class Consommateur extends Thread {
    private MessageQueue boitec;

    public Consommateur(MessageQueue m) {
        boitec = m;
    }

    public void run() {
        while (true) {
            Date message = (Date)boitec.receive();
            if (message != null)
                // consomme les message qui arrivent dans la boîte
                System.out.println ("Message reçu : "+message);
        }
    }
}
```

Producteurs/Consommateurs (4/4)

La file de messages :

```
import java.util.Vector;

public class MessageQueue{
    private Vector queue;

    public MessageQueue() {
        queue = new Vector();
    }

    public void send(Object I) {
        queue.addElement(I);
    }

    public Object receive() {
        Object item;
        if( queue.size() == 0)
            return null;
        else {
            item = queue.firstElement();
            queue.removeElementAt(0);
            return item;
        }
    }
}
```

Conclusion/Bilan (1/2)

- Les processus poids-léger et les threads sont des nouvelles abstractions (de nouveaux modèles).
- Ils permettent de distinguer plusieurs chaînes de contrôle à l'intérieur d'un même processus et d'en augmenter le degré de concurrence apparent.
- On conserve l'équivalence application/processus.
- Mais un processus compte plusieurs threads :
  - Ils partagent le même espace d'adressage.
  - Leurs exécutions s'effectuent indépendamment les uns des autres.
- En particulier, un thread peut être actif pendant que les autres sont bloqués.

Conclusion/Bilan (2/2)

- Dans certains systèmes d'exploitation, le kernel comporte plusieurs threads concurrents :
  - Windows NT, Solaris 2.
- On peut distinguer :
  - les threads usagers
    - Implantés par des librairies,
    - Le kernel n'en a pas connaissance.
  - de ceux du kernel
    - Qui ne sont pas accessibles à l'utilisateur,
    - Mais gérer par le répartiteur du kernel.
- L'inconvénient majeur des threads est qu'ils introduisent des problèmes de synchronisation complexes (en particulier dans le kernel).