Threads/Processus légers

Définition

- Un thread est un processus léger
- Un thread permet de partager l'environnement d'un processus
  - les variables
  - les objets
  - les descripteurs de fichier
  - ...
- Un processus peut être composer de plusieurs threads

Avantages/Inconvénients

Avantages

- Facilite la communication

Inconvénients

- Attention aux modifications de valeur...

Création

Une activité peut être créée de deux manières :

- Héritage de la classe Thread

```
class X extends Thread {
    ...
    public void run () {
        ... code de l'activité ...
    }
}
```

- Implantation de l'interface Runnable

```
class X implements Runnable {
    ...
    public void run () {
        ... code de l'activité ...
    }
}
```

Utilisation

- Héritage de la classe Thread

```
foo() {
  X x = new X();
  x.start();
  ...
  x.join();
 }
```

- Implantation de l'interface Runnable

```
foo() {
   X x = new X(...);
   Thread t = new Thread(x);
   t.start();
   :
   t.join();
}
```

Exemple extends

```
import java.util.*;

public class PingPong extends Thread {
    String mot;
    Random rnd;
    int delai;

    PingPong(String chaine, int del) {
        mot = chaine ;
        delai = del;//new Random();
    }

    public void run(){
        try {
            for (;;) {
                System.out.print(mot+ "␣");
                sleep(delai);
            }
        } catch (InterruptedException e) { }}


    public static void main(String[] args){
        new PingPong("ping", 33).start();
        new PingPong("PONG", 100).start();
    }
}
```

Exemple implements

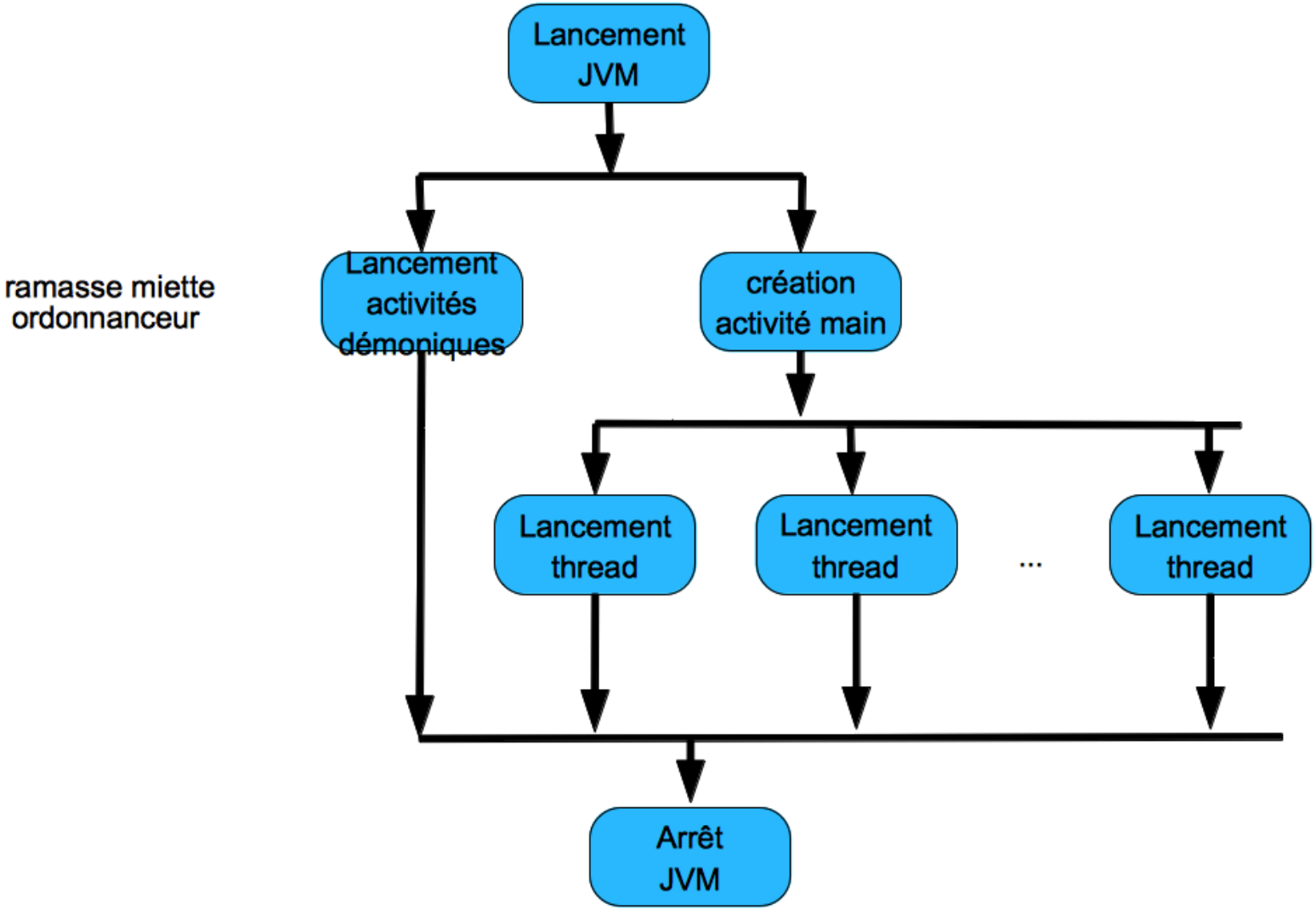
```
public class RunPingPong implements Runnable{
    String mot;
    int attente;

    RunPingPong(String chaine, int delai) {
        mot = chaine ;
        attente = delai ;
    }

    public void run(){
        try {
            for (;;) {
                System.out.print(mot+ "␣");
                Thread.sleep(attente);
            }
        } catch (InterruptedException e) { }}

    public static void main(String[] args){
        Runnable Ping = new RunPingPong("ping", 33);
        Runnable Pong = new RunPingPong("PONG", 100);
        new Thread(Ping).start();
        new Thread(Pong).start();
    }
}
```

La JVM et les threads



Modèle de thread (un peu d'histoire...)

Green thread

- Gestion des threads au niveau de la machine virtuelle
- non préemptif, mais assurant la commutation sur entrée-sortie bloquante

native thread

- Gestion des threads au niveau du noyau
- préemptif, utilisable sur système multi-processeurs

Avantages/Inconvénients

Thread dans l'espace utilisateur

- Implantable sur tous les systèmes, même ceux dépourvus de threads
- Basculement entre thread plus rapide
- Chaque processus peut avoir son propre algo d'ordonnancement
- Problème sur les appels systèmes bloquants
- Pas de préemption au niveau thread

Thread dans le noyau

- Création d'un thread plus lourd (mécanisme de recyclage)
- Préemption possible au sein d'un processus

Verrou et Java : synchronised

Utilité : permet de déclarer qu'une méthode ou un bloc d'instructions est critique : un seul thread à la fois peut se trouver dans une partie synchronisée sur un objet.

Fonctionnement : Chaque objet JAVA possède un verrou. Pour exécuter une section de code synchronisée (bloc ou méthode), il faut posséder le verrou.

synchronised (suite)

Pour un bloc de commande

```
synchronized (unObj) {
       < Région critique >
   }
```

Pour une méthode

```
synchronized T uneMethode(...) { ... }
```

Ce qui est équivalent à

```
T uneMethode(...) {
     synchronized (this) { ... }
}
```

Exemple : Compte

```
public class Compte {
    private int valeur;

    Compte(int val) {valeur = val;}

    public int solde() {return valeur;}

    public void depot(int somme) {
        if (somme > 0) valeur+=somme;
    }

    synchronized public  boolean retirer(int somme)
        throws InterruptedException {
        if (somme > 0)
            if (somme <= valeur) {
                Thread.currentThread().sleep(50);
                valeur -= somme;
                Thread.currentThread().sleep(50);
                return true;
            }
        return false;
    } }
```

Exemple : Compte

```
import java.util.*;

public class useCompte extends Thread {
    Compte tc;
    useCompte(Compte c){tc=c;}

    public void run(){
        System.out.println(tc.solde());
        try
            {
                tc.retirer(700);
            }
        catch(InterruptedException e){System.out.println("error");}
        System.out.println(tc.solde());
    }


    public static void main(String[] args){
        Compte lc = new Compte(1000);
        Thread t1 = new useCompte(lc);
        Thread t2 = new useCompte(lc);
        t1.start();
        t2.start();
    }
}
```

La synchronisation par objet

Les primitives :

- unObj.wait() libère l'accès exclusif à l'objet et bloque l'activité appelante en attente d'un réveil via une opération unObj.notify ;
- unObj.notify() réveille une seule activité bloquée sur l'objet (si aucune activité n'est bloquée, l'appel ne fait rien) ;
- unObj.notifyAll() réveille toutes les activités bloquées sur l'objet.

Exemple

```
public class Evenement
{
    private Boolean Etat; // etat de l'evenement
    public Evenement() {
        Etat = Boolean.FALSE;
    }
    public synchronized void set() {
        Etat = Boolean.TRUE;
        // debloque les threads qui attendent cet evenement:
        //notifyAll();
                notify();
    }
    public synchronized void reset() {
        Etat = Boolean.FALSE;
    }
    public synchronized void attente() {
        if(Etat==Boolean.FALSE) {
            try {
                wait(); // bloque jusqu'a un notify()
            }
            catch(InterruptedException e) {};
        }
    }// fin attente
}// fin classe
```

Les sémaphores

```
public class Semaphore {

    private int cpt = 0;
    Semaphore (int c) {
        cpt = c;
    }

    public void P() throws InterruptedException {
        synchronized (this) {
            while (cpt == 0) {
                this.wait ();
            }
            cpt--;
        }
    }

    public void V() {
        synchronized (this) {
            cpt++;
            this.notify ();
        }
    }
}
```