QUALITÉ DE DÉVELOPPEMENT

LA PROGRAMMATION CONCURRENTE EN JAVA

🎓 2A - Bachelor Universitaire de Technologie
🏛 IUT d'Orsay - Université Paris-Saclay - 2024/2025

Idir AIT SADOUNE
idir.ait-sadoune@universite-paris-saclay.fr

PLAN

Retour au plan - Retour à l'accueil

PLAN

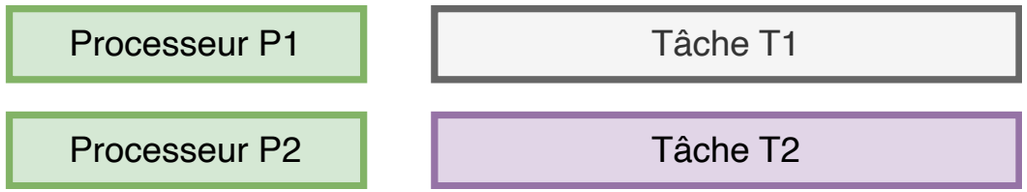
Retour au plan - Retour à l'accueil

SYSTÈMES MULTITÂCHES

Un système **multitâches** permet d'exécuter plusieurs programmes informatiques en parallèle.

SYSTÈMES MULTIPROCESSEURS

- **Vrai-parallélisme** $\rightarrow$
  un système multiprocesseurs permet un parallélisme, où chaque tâche est exécutée par un processeur.



SYSTÈMES MONOPROCESSEUR

- **Pseudo-parallélisme** →
  les systèmes monoprocesseur permettent un parallélisme, où toutes les tâches s'exécutent sur le même processeur.

- **Systèmes à traitement par lots**

- **Systèmes à temps partagé**

LA PROGRAMMATION CONCURRENTE

Un **paradigme de programmation** qui tient compte de l'existence de plusieurs unités d'exécution de base dans un programme :

- **les processus** et **les threads**.

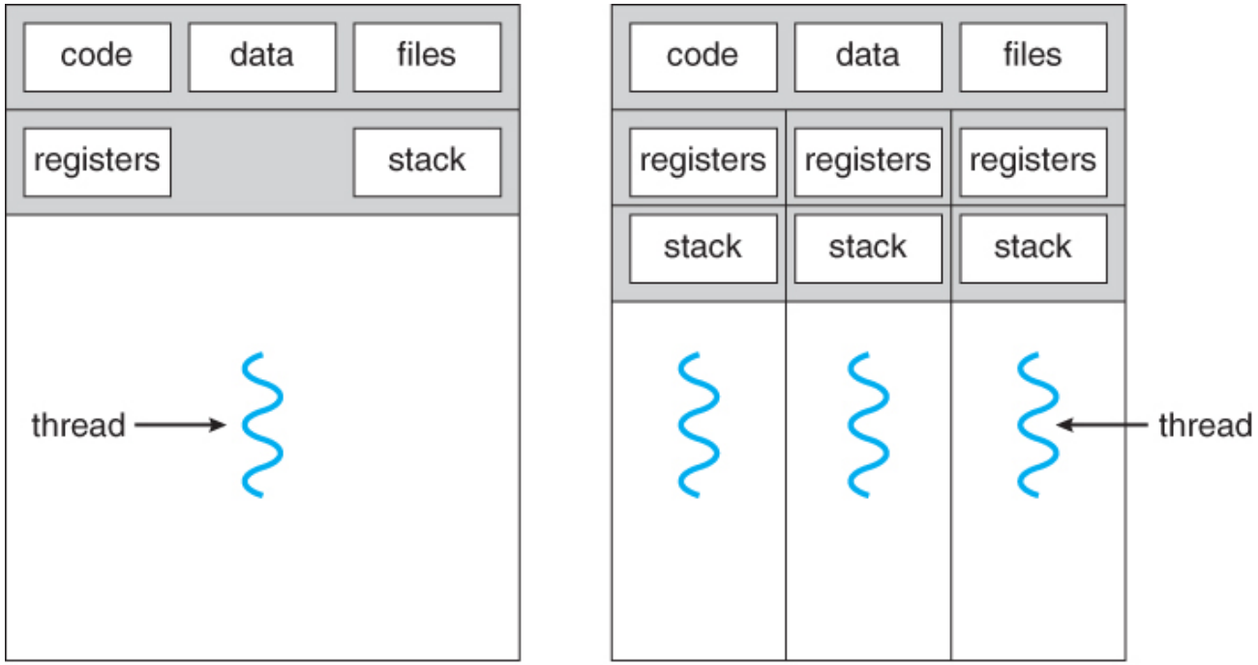
LA NOTION DE PROCESSUS

- **Les processus** sont considérés comme synonymes de programmes.
  - ➠ un processus est un programme en cours d'exécution.
- Un processus possède un environnement d'exécution autonome.
  - ➠ chaque processus a son propre espace mémoire.
- Pour gérer la communication entre les processus, la plupart des OS utilisent des **IPC** (**Inter Process Communication**) :
  - les pipes et les sockets (à étudier dans le cours suivant)

LA NOTION DE THREAD

- **Les threads** sont aussi appelés processus légers
  - ➠ la création d'un thread nécessite moins de ressources qu'un processus.
- Les threads sont créés par des processus. Les threads partagent les ressources du processus, y compris la mémoire et les fichiers ouverts.
- Cela rend la communication efficace mais potentiellement problématique
  - ➠ synchronisations et gestion des ressources partagées

PROCESSUS VS THREAD



single-threaded process

multithreaded process

PLAN

Retour au plan - Retour à l'accueil

LES PROCESSUS EN JAVA

- La description du comportement d'un processus en **Java** se fait en implémentant la méthode `main()` dans une classe.

```java
public class ProcessInstanciation {
  public static void main(String[] args) {
    System.out.println("Run with main method");
  }
}
```

- La création d'un processus se fait en exécutant la méthode `main()`.

```
Run with main method
```

LES THREADS EN JAVA

- La création d'un thread se fait de deux manières:
  1. en dérivant la classe `Thread` et en surchargeant la méthode `run()`.
  2. en implémentant l'interface `Runnable` contenant la méthode `run()`.

- L'exécution d'un thread se fait en utilisant la méthode `start()`.
  Le code métier (la méthode `run()`) sera exécuté en tâche de fond.

ETENDRE LA CLASSE Thread

```java
public class FromThreadClass extends Thread {
  public FromThreadClass(String id) {
    super(id);
  }

  public void run() {
    System.out.println("[Thread "+ this.getName() +"] Run");
  }
}
```

```java
public class FromProcessClass {
  public static void main(String[] args) {
    FromThreadClass th1 = new FromThreadClass("TH1");
    th1.start();
  }
}
```

```
[Thread TH1] Run
```

IMPLÉMENTER L'INTERFACE Runnable

```java
public class FromRunnableInterface implements Runnable {
  private String name ;

  public FromRunnableInterface(String n){
    this.name = n;
  }

  public void run() {
    System.out.println("[Thread "+ this.name +"] Run");
  }
}
```

```java
public class FromProcessClass {
  public static void main(String[] args) {
    Thread th1 = new Thread(new FromRunnableInterface("TH1"));
    th1.start();
  }
}
```

```
[Thread TH1] Run
```

PLAN

Retour au plan - Retour à l'accueil

LES MÉTHODES sleep() ET join()

```
1  public class ThreadFils extends Thread {
2    public ThreadFils(String name) {
3      super(name);
4    }
5
6    public void run() {
7      try {
8        System.out.println("[Thread "+this.getName()+"] Debut");
9        System.out.println("[Thread "+this.getName()+"] Sleep ...");
10       this.sleep(2000);
11       System.out.println("[Thread "+this.getName()+"] Fin");
12     } catch (InterruptedException e) {
13       e.printStackTrace();
14     }
15   }
16 }
```

LES MÉTHODES sleep() ET join()

```java
public class Main {
  public static void main(String[] args) {
    try {
      System.out.println("[Main Process] Debut");
      ThreadFils f1 = new ThreadFils("f1");
      f1.start();
      f1.join();
      System.out.println("[Main Process] Fin");
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

```
[Main Process] Debut
[Thread f1] Debut
[Thread f1] Sleep ...
[Thread f1] Fin
[Main Process] Fin
```

LES MÉTHODES sleep() ET join()

```java
public class Main {
  public static void main(String[] args) {
    try {
      System.out.println("[Main Process] Debut");
      ThreadFils f1 = new ThreadFils("f1");
      f1.start();
      f1.join(1000);
      System.out.println("[Main Process] Fin");
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

```
[Main Process] Debut
[Thread f1] Debut
[Thread f1] Sleep ...
[Main Process] Fin
[Thread f1] Fin
```

LA MÉTHODE interrupt()

```java
public class Main {
  public static void main(String[] args) {
    try {
      System.out.println("[Main Process] Debut");
      ThreadFils f1 = new ThreadFils("f1");
      f1.start();
      f1.join(1000);
      f1.interrupt();
      System.out.println("[Main Process] Fin");
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }
}
```

```
[Main Process] Debut
[Thread f1] Debut
[Thread f1] Sleep ...
[Main Process] Fin
    java.lang.InterruptedException: sleep interrupted
        at java.lang.Thread.sleep(Native Method)
        at ThreadFils.run(ThreadFils.java:15)
```

MERCI

Version PDF des slides

Retour à l'accueil - Retour au plan