Thread

**1. Process là gì?**

**Process (quá trình)** là một chương trình đang chạy trên máy tính. Mỗi process có không gian địa chỉ của riêng nó, các tài nguyên hệ thống (như bộ nhớ và CPU) và ít nhất một thread hoạt động.

**Đặc điểm của Process**:

* Mỗi process có không gian bộ nhớ độc lập, nên chúng không chia sẻ dữ liệu trực tiếp với nhau.
* Mỗi process có thể có nhiều thread bên trong nó.
* Hệ điều hành quản lý quá trình thông qua cơ chế "quản lý tiến trình" (process management).

Ví dụ: Mỗi ứng dụng mà bạn mở trên máy tính (như trình duyệt web, trình biên dịch, trò chơi) đều là một process.

**2. Thread là gì?**

**Thread (luồng)** là đơn vị nhỏ nhất của một quá trình (process) có thể được lên lịch để thực hiện. Một process có thể có một hoặc nhiều thread (được gọi là multi-threaded).

**Đặc điểm của Thread**:

* Nhiều thread trong cùng một process chia sẻ không gian bộ nhớ và tài nguyên của process đó.
* Các thread hoạt động đồng thời, và mỗi thread thực hiện một phần công việc cụ thể.
* Một thread có thể thực hiện công việc trong khi các thread khác cũng đang chạy trong cùng process.

**3. Có bao nhiêu cách để tạo 1 thread trong Java? Khác biệt giữa các cách đó?**

Trong Java, có hai cách chính để tạo một thread:

**1. Kế thừa từ lớp Thread:**

* Bạn có thể tạo một class mới và kế thừa từ lớp Thread, sau đó ghi đè phương thức run().

Vd:

class MyThread extends Thread {

public void run() {

System.out.println("Thread is running.");

}

}

public class Main {

public static void main(String[] args) {

MyThread t1 = new MyThread();

t1.start(); // Start the thread

}

}

**2. Triển khai interface Runnable:**

* Bạn cũng có thể tạo một class triển khai interface Runnable và truyền đối tượng đó vào lớp Thread.

Vd:

class MyRunnable implements Runnable {

public void run() {

System.out.println("Runnable is running.");

}

}

public class Main {

public static void main(String[] args) {

Thread t1 = new Thread(new MyRunnable());

t1.start(); // Start the thread

}

}

**Khác biệt giữa hai cách**:

* **Kế thừa từ Thread**: Class chỉ có thể kế thừa một lớp duy nhất trong Java (vì Java không hỗ trợ đa kế thừa), nên khi kế thừa từ Thread, bạn không thể kế thừa từ lớp khác.
* **Triển khai Runnable**: Ưu điểm lớn nhất là bạn có thể triển khai interface này mà không bị hạn chế việc kế thừa lớp khác. Điều này linh hoạt hơn, do đó thường được khuyến khích hơn.

**4. Multi-thread là gì? Sử dụng multi-thread mang lại ưu nhược điểm gì?**

**Multi-thread:**

* **Multi-threading** là kỹ thuật cho phép một chương trình thực hiện nhiều thread cùng lúc. Mỗi thread chạy đồng thời và có thể xử lý một phần công việc cụ thể.
* Multi-threading thường được sử dụng trong các ứng dụng yêu cầu xử lý nhiều tác vụ đồng thời, như xử lý song song, giao tiếp mạng, hoặc UI.

**Ưu điểm của multi-threading:**

* **Tăng hiệu suất**: Các thread có thể chạy đồng thời trên các nhân CPU khác nhau, giúp tăng tốc độ xử lý.
* **Tối ưu hóa tài nguyên**: Trong một chương trình đơn luồng, nếu chương trình bị chặn (ví dụ, khi đợi dữ liệu I/O), hệ thống sẽ không thực hiện bất kỳ công việc nào khác. Với multi-threading, các thread khác có thể tiếp tục công việc trong khi một thread bị chặn.
* **Cải thiện phản hồi**: Ứng dụng sẽ phản hồi nhanh hơn, ví dụ như khi một phần của ứng dụng chờ dữ liệu, phần còn lại vẫn có thể hoạt động.

**Nhược điểm của multi-threading:**

* **Phức tạp trong quản lý**: Multi-threading có thể gây ra các vấn đề phức tạp như "race condition", "deadlock", và cần phải đồng bộ hóa giữa các thread.
* **Overhead**: Việc tạo và quản lý nhiều thread có thể tốn tài nguyên hệ thống. Nếu không quản lý tốt, việc sử dụng quá nhiều thread có thể làm giảm hiệu suất.

**5. Làm thế nào để biết được một thread hoặc multi-thread đã hoàn thành hay chưa?**

Để kiểm tra xem một thread đã hoàn thành hay chưa, có thể sử dụng một trong những cách sau:

**Phương thức isAlive()**:

* Phương thức này kiểm tra xem thread có đang hoạt động hay không. Nếu thread đã kết thúc, phương thức sẽ trả về false.

Vd:

Thread t = new Thread(new MyRunnable());

t.start();

if (!t.isAlive()) {

System.out.println("Thread has finished.");

}

**Phương thức join()**:

* Phương thức join() chờ cho đến khi thread hoàn thành trước khi tiếp tục thực hiện các lệnh sau nó.

Vd:

Thread t = new Thread(new MyRunnable());

t.start();

t.join(); // Chờ cho đến khi thread t kết thúc

System.out.println("Thread has finished.");

**6. Có giới hạn việc tạo ra bao nhiêu thread trong một ứng dụng Java hay không?**

Có, việc tạo ra quá nhiều thread trong một ứng dụng Java sẽ bị giới hạn bởi một số yếu tố:

1. **Tài nguyên hệ thống**:
   * Mỗi thread tiêu tốn một lượng bộ nhớ nhất định cho ngăn xếp (stack), và CPU cần phải phân phối thời gian cho mỗi thread. Nếu bạn tạo quá nhiều thread, hệ thống có thể hết tài nguyên hoặc hiệu suất bị suy giảm nghiêm trọng.
2. **Giới hạn cấu hình JVM**:
   * JVM có thể giới hạn số lượng thread có thể được tạo ra dựa trên cấu hình của hệ thống và thiết lập JVM (như kích thước ngăn xếp mặc định cho mỗi thread). Bạn có thể điều chỉnh giới hạn này bằng cách sử dụng tham số -Xss để thay đổi kích thước ngăn xếp của thread.
3. **Giới hạn hệ điều hành**:
   * Hệ điều hành cũng có thể áp đặt giới hạn số lượng thread mà một ứng dụng có thể tạo ra.