Ministry of SMEs and Startups × KOSME Korea SMEs and Startups Agency × LIKELION

Single-Threaded, Multi-threaded

Nguyễn Anh Tuấn

K TECH COLLEGE

Nội dung bài giảng

1. Đơn luồng - Single-threaded
2. Đa luồng - Multi-threaded

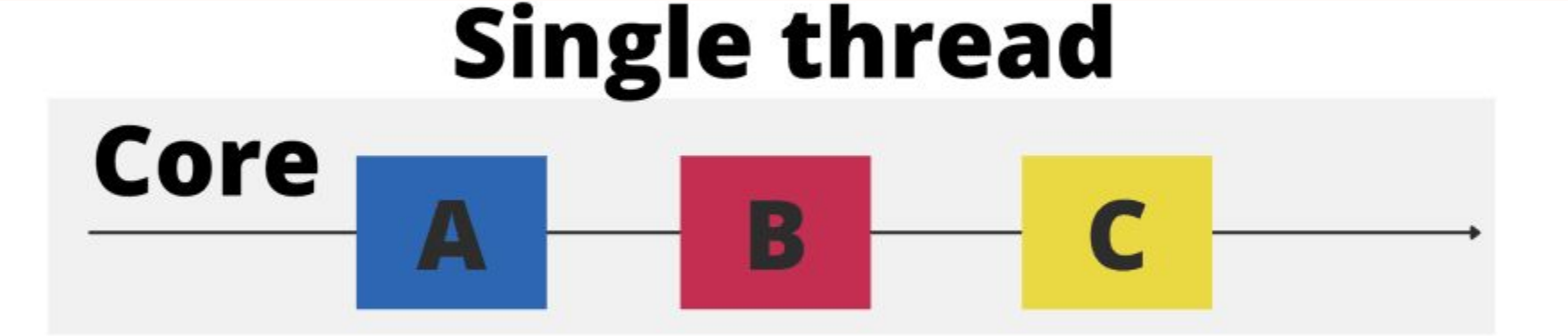
Đơn luồng - Single thread

Đa luồng - Multi thread

Đơn luồng - Single-threaded

Đơn luồng - Single-threaded

- **Định nghĩa**:
  - Mỗi nhiệm vụ (task) được cấp phát CPU để thực hiện độc lập gọi là 1 luồng **(thread)**.
  - 1 **thread** là 1 luồng thực hiện từ khi bắt đầu tới khi kết thúc 1 task.
- **Lưu ý**:
  - Hàm main được gọi là "**main thread**". JVM sẽ tạo ra main thread và sử dụng nó để thực thi nội dung của hàm main trong một luồng riêng biệt.



Đơn luồng - Single-threaded

```
public class Application implements Runnable {
        private String name;
        public Application(String name) { this.name = name; }

        @Override
        public void run() { // Phương thức run() chứa những công việc cần làm trong thread đó
                for (int i = 0; i<10; i++) { System.out.println(name); }

        public static void main(String[] args) {
                Runnable task1 = new Application("Tấn");
                Thread thread1 = new Thread(task1);
                thread1.start();
        }
}
```

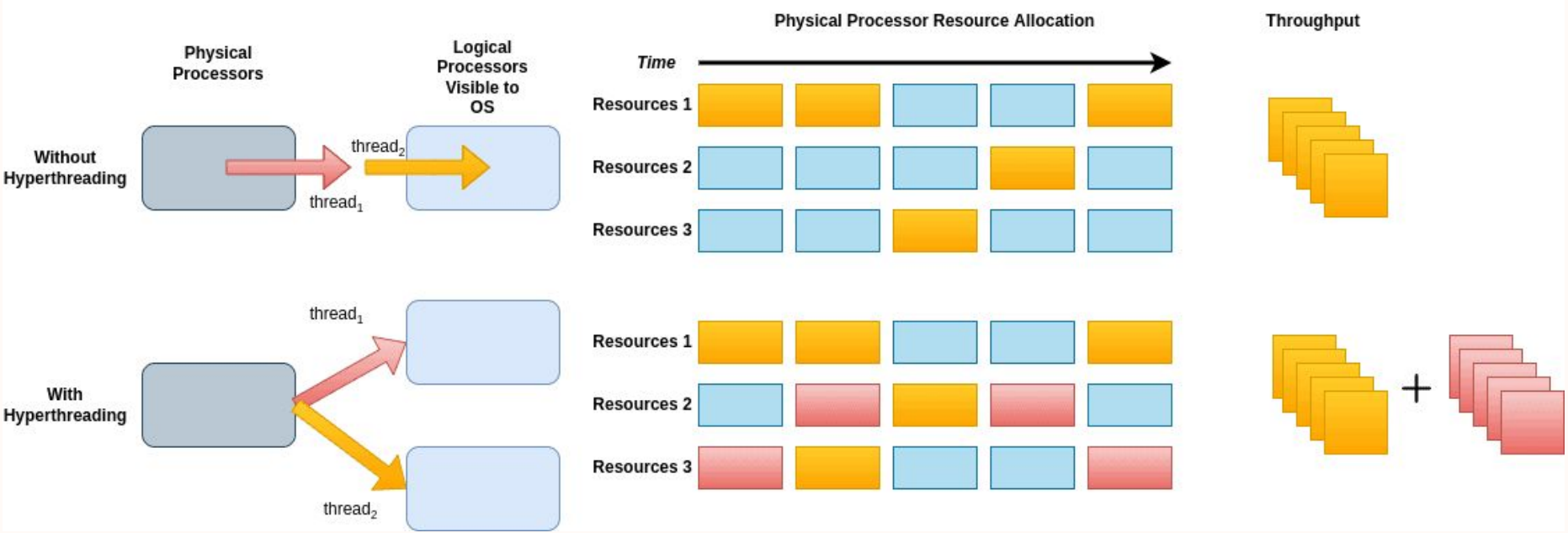
Đơn luồng - Single-threaded

Tại sao lại ghi đè phương thức **run()** nhưng lại chạy bằng phương thức **start()** ?

- Khởi tạo thread mới **start()**: JVM sẽ tạo một thread mới và bắt đầu chạy thread đó.
- Gọi phương thức **run()**: Thread mới sẽ gọi phương thức run() để thực thi chuỗi lệnh đã định nghĩa trong run().
- Lưu ý:
  - Nếu chỉ gọi phương thức **run()** mà không gọi phương thức **start()**, nó sẽ không tạo ra một thread mới. Thay vào đó, nó sẽ chạy phương thức run() trong thread hiện tại.

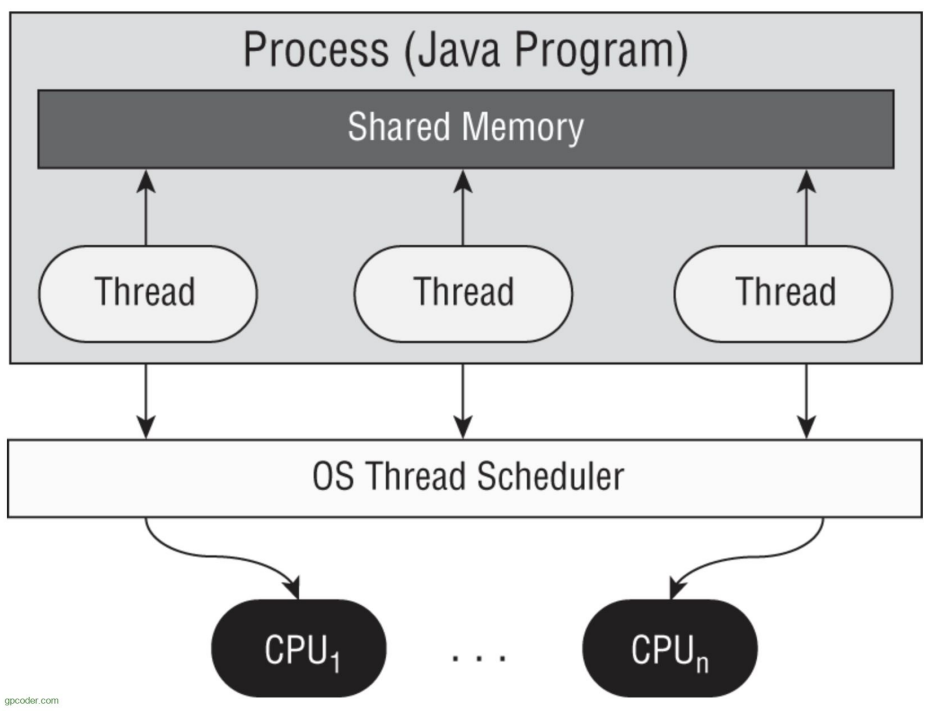
Đa luồng - Multi-threaded

Đa luồng - Multi-threaded



Đa luồng - Multi-threaded

- **Định nghĩa**:
  - Đa luồng cho phép nhiều task cùng thực thi đồng thời trong 1 tiến trình. Hay còn gọi là 1 tiến trình có thể thực hiện nhiều luồng đồng thời.
  - Đa luồng có thể chạy trên 1 hoặc nhiều CPU, điều này sẽ do hệ điều hành quyết định về việc chia sẻ tài nguyên.



Đa luồng - Multi-threaded

```
public class Application implements Runnable {
        private String name;
        public Application(String name) { this.name = name; }

        @Override
        public void run() { // Phương thức run() chứa những công việc cần làm trong thread đó
                for (int i = 0; i<10; i++) { System.out.println(name); }

        public static void main(String[] args) {
                Runnable task1 = new Application("Xem diu tu be");
                Thread thread1 = new Thread(task1);
                Runnable task1 = new Application("Xem tốp tốp");
                Thread thread2 = new Thread(task2);
                thread1.start();
                thread2.start();
        }
}
```

Đa luồng - Multi-threaded

```
// Bổ sung thêm phương thức sau để kiểm tra thông tin về CPU hiện tại
// Từ đây có thể thấy các thread chạy trên các core khác nhau nếu OS cho phép
private static Long getCurrentCPU() {
        return java.lang.management
                .ManagementFactory
                .getThreadMXBean()
                .getThreadCpuTime(Thread.currentThread().getId());
    }

public static void main(String[] args) {
        System.out.println("Thread " + Thread.currentThread().getName()
        + " is running on CPU core: " + getCurrentCPU());
}
```

Đa luồng - Multi-threaded

- **Ưu điểm của Multi-threaded**:
  - Mỗi luồng có thể dùng chung và chia sẻ nguồn tài nguyên trong quá trình chạy.
  - Các luồng là độc lập, có thể thực hiện nhiều công việc cùng một lúc.
  - Các luồng là độc lập, vì vậy nếu 1 luồng xảy ra exception thì nó không ảnh hưởng đến luồng khác trong ***(trường hợp được try-catch)***.

- **Nhược điểm của Multi-threaded**:
  - Càng nhiều luồng thì xử lý càng phức tạp.
  - Xử lý vấn đề về tranh chấp bộ nhớ, đồng bộ dữ liệu khá phức tạp.
  - Cần phát hiện tránh các luồng chết (deadlock). Deadlock xảy ra khi 2 tiến trình đợi nhau hoàn thành trước khi chạy. Kết quả của quá trình là cả 2 tiến trình không bao giờ kết thúc.

Đa luồng - Multi-threaded

- **Vòng đời (trạng thái) của 1 luồng**:
  - **NEW**: Luồng vừa được khởi tạo nhưng chưa được start(). Ở trạng thái này, luồng được tạo ra nhưng chưa được cấp phát tài nguyên.
  - **RUNNABLE**: Sau khi gọi phương thức start() thì luồng đã được cấp phát tài nguyên và các lịch điều phối CPU cho luồng cũng bắt đầu có hiệu lực.
  - **RUNNING**: Khi lịch điều phối của JVM chọn thread để thực thi, nó chuyển sang trạng thái running. Thread đang thực sự chạy và thực thi mã trong phương thức run() của nó
  - **WAITING**: Thread chờ không giới hạn cho đến khi một luồng khác đánh thức nó.
  - **TIMED_WAITING**: Thread chờ trong một thời gian nhất định, hoặc là có một luồng khác đánh thức nó.
  - **BLOCKED**: Đây là 1 dạng của trạng thái “Not Runnable”, là trạng thái khi Thread vẫn còn sống, nhưng hiện tại không được chọn để chạy.
  - **TERMINATED**: Một thread ở trong trạng thái terminated hoặc dead khi phương thức run() của nó bị thoát.

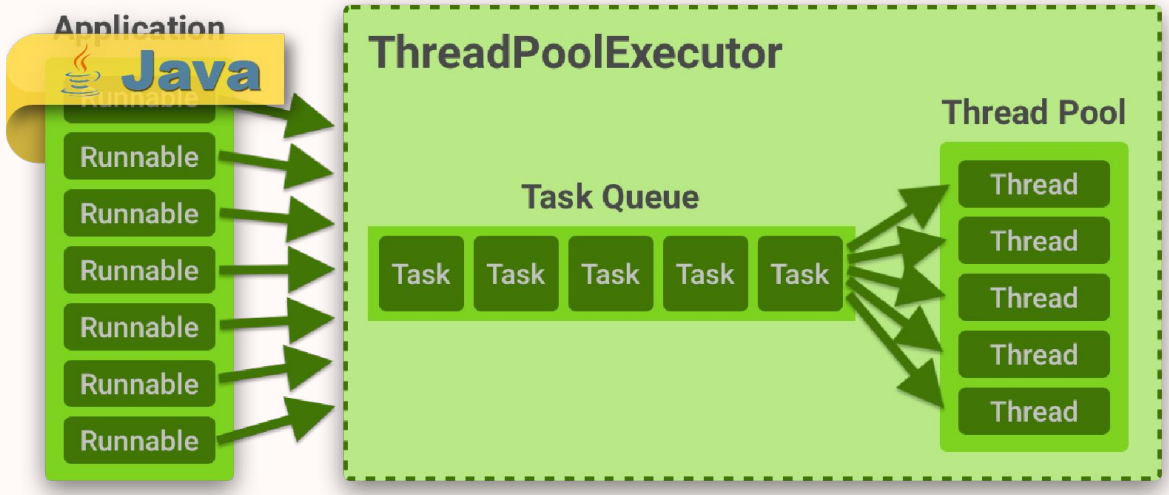
Đa luồng - Multi-threaded

- **Các phương thức thường dùng**:
  - **.sleep()**: Tạm dừng thread hiện tại trong một khoảng thời gian xác định (đơn vị: milliseconds).
  - **.join()**: Chờ cho đến khi thread kết thúc (đơn vị: milliseconds).
  - **.wait(), .notify(), .notifyAll()**: Đợi cho đến khi được đánh thức (**wait**). Đánh thức một thread đang đợi (**notify**). Đánh thức tất cả các thread đang đợi (**notifyAll**).
  - **.setName(), .getName()**: Đặt tên và lấy tên của thread.
  - **.setPriority(), .getPriority()**: Mức độ ưu tiên của thread (từ 1 đến 10).
  - **.yield()**: Tạm giải phóng các thread khác, nhường quyền ưu tiên cho thread có độ ưu tiên cao hơn.

Thread pool

Thread pool

- **Định nghĩa**:
  - Là 1 cơ chế để lưu đệm các thread vào 1 **tập hợp** (hoặc là bể chứa - pool).
  - **Thread pool** tận dụng những thread đã tạo mà đang nhàn rỗi để tái sử dụng.
  - 1 cơ chế khác của **Thread pool** là mở thêm thread mới nhằm tránh hiện tượng quá tải.



1 ví dụ về Hội trường hiến máu:

- **Hội trường - Thread pool**
- **Ghế ngồi - Thread**
- **Thanh niên làm tình nguyện hết mình - Task**

=> Trong hội trường cứ có ghế nào trống thì thanh niên nhào vào ngồi, không thì phải chờ.

Thread pool

- **Cơ chế tạo pool với số thread cố định**:

```
public static void main(String[] args) {

    // Tạo pool với 2 thread cố định
    ExecutorService es = Executors.newFixedThreadPool(2);
    // Executors để thực thi các task trong thread pool
    // ExecutorService để quản lý và điều khiển các task

    // Đặt các task vào trong pool để thực thi
    es.execute(new Application("Tấn"));
    es.execute(new Application("Kiên"));
    es.execute(new Application("Khang"));

    // Đóng cửa bể bơi
    es.shutdown();
}
```

Thread pool

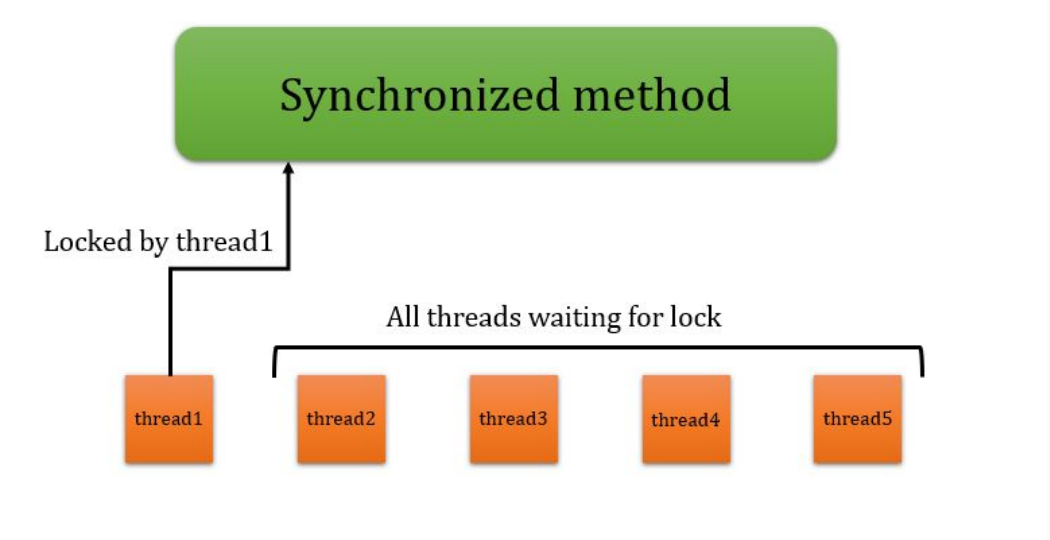
- **Cơ chế tạo pool với số thread linh hoạt**:

```
public static void main(String[] args) {

    // Tạo pool với số thread linh hoạt
    ExecutorService es = Executors.newCachedThreadPool();

    // Đặt các task vào trong pool để thực thi
    es.execute(new Application("Tấn"));
    es.execute(new Application("Kiên"));
    es.execute(new Application("Khang"));
    // Nếu 2 thread trên ko có cái nào trống thì pool sẽ tạo ra 1 thread thứ 3 để chạy

    // Đóng cửa bể bơi
    es.shutdown();
}
```

Thread synchronization

Thread synchronization

- **Định nghĩa**:
  - **Synchronized** là cơ chế đồng bộ phương thức, nó sẽ khoá (lock) phương thức mỗi khi thực thi. 1 thời điểm phương thức đó chỉ đc 1 thread truy cập.
  - **Synchronized** giúp tránh việc các thread tranh chấp tài nguyên khiến cho việc thực thi đa luồng có thể dẫn đến sai sót.



Thread synchronization

```
public class Account extends Thread {
        private double amount;
        public Account(double amount) { this.amount = amount; }

        public synchronized void withdraw(double money) throws InterruptedException {
                if (this.amount >= money) {
                        Thread.sleep(1000);
                        this.amount = this.amount - money;
                        System.out.println(this.amount);
                } else System.out.println("Hết tiền");
        }

        @Override
        public void run() { this.withdraw(10); }
}
```

ROAD TO KOREA

Nếu có bất kỳ thắc mắc nào, hãy đặt câu hỏi qua

mail@mail.com hoặc Zalo 0xxx xxx xxx