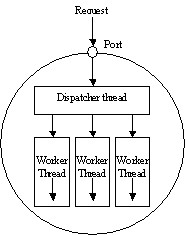
**Luồng (Thread)**

Luồng là một cách thông dụng để nâng cao năng lực xử lý của các ứng dụng nhờ vào cơ chế song song. Trong một hệ điều hành cổ điển, đơn vị cơ bản sử dụng CPU là một quá trình. Mỗi quá trình có một Thanh ghi bộ đếm chương trình (PC-Program Counter), Thanh ghi trạng thái (Status Register), ngăn xếp (Stack) và không gian địa chỉ riêng (Address Space).

Ngược lại, trong một hệ điều hành có hỗ trợ tiện ích luồng, đơn vị cơ bản sử dụng CPU là một luồng. Trong những hệ điều hành này, một quá trình bao gồm một không gian địa chỉ và nhiều luồng điều khiển. Mỗi luồng có bộ đếm chương trình, trạng thái thanh ghi và ngăn xếp riêng. Nhưng tất cả các luồng của một quá trình cùng chia sẻ nhau một không gian địa chỉ. Nhờ đó các luồng có thể sử dụng các biến toàn cục, chia sẻ các tài nguyên như tập tin, hiệu báo một cách dễ dàng...

Cách thức các luồng chia sẻ CPU cũng giống như cách thức của các quá trình. Một luồng cũng có những trạng thái: đang chạy (running), sẵn sàng (ready), nghẽn (blocked) và kết thúc (Dead). Một luồng thì được xem như là một quá trình nhẹ.

Nhờ vào luồng, người ta thiết kế các server có thể đáp ứng nhiều yêu cầu một cách đồng thời.

Sử dụng luồng cho các server

Trong mô hình này, Server có một **Luồng phân phát**(Dispatcher thread) và nhiều **Luồng thực hiện**(Worker thread). Luồng phân phát tiếp nhận các yêu cầu nối kết từ các Client, rồi chuyển chúng đến các luồng thực hiện còn rảnh để xử lý. Những luồng thực hiện hoạt động song song nhau và song song với cả luồng phân phát,  nhờ đó,  Server có thể phục vụ nhiều Client một cách đồng thời.

## Các mức cài đặt luồng

Nhìn từ góc độ hệ điều hành , Luồng có thể được cài đặt ở một trong hai mức:

* Trong không gian người dùng (user space)
* Trong không gian nhân (kernel mode)

## Luồng trong java

Trong Java, luồng là 1 đối tượng thuộc lớp java.lang.Thread. Một chương trình trong java có thể cài đặt luồng bằng cách tạo ra một lớp con của lớp java.lang.Thread.

Lớp này có 3 phương thức cơ bản để điều khiển luồng là:

* public native synchronized void start()
* public void run()
* public final void stop()

### Phương thức start()

Chuẩn bị mọi thứ để thực hiện luồng.

### Phương thức run()

Thực hiện công việc thực sự của luồng, () sẽ được kích hoạt một cách tự động bởi phương thức start().

### Phương thức stop()

Kết thúc luồng.

Luồng sẽ "chết" khi tất cả các công việc trong  phương thức run() được thực hiện hoặc khi phương thức stop() được kích hoạt.

## Đồng bộ hóa giữa các luồng

Tất cả các luồng của một quá trình thì được thực thi song song và độc lập nhau nhưng lại cùng chia sẻ nhau một không gian địa chỉ của quá trình. Chính vì vậy có thể dẫn đến khả năng đụng độ trong việc cập nhật các dữ liệu dùng chung của chương trình (biến, các tập tin được mở) khi một luồng ghi lên một dữ liệu trong khi một luồng khác đang đọc dữ liệu này.

Trong trường hợp đó, cần phải sử dụng cơ chế đồng bộ hóa của Java. Có nhiều mức đồng bộ hóa như: trên một biến, trên một câu lệnh, trên một khối lệnh hay trên một phương thức.

# Stream:

Nhập dữ liệu - input (nhập dữ liệu từ bàn phím, đọc dữ liệu từ tập tin, …) và xuất dữ liệu - output (xuất dữ liệu ra màn hình, ghi dữ liệu ra tập tin, đĩa, máy in), gọi tắt là I/O, là 2 thao tác được thực hiện thường xuyên khi lập trình. Java đã cung cấp gói java.io chứa gần như toàn bộ các lớp mà chúng ta cần để thực hiện 2 thao tác này.

Bên cạnh đó, Java còn đưa ra khái niệm **Stream** (luồng) để thực hiện I/O nhanh hơn.

Stream là một chuỗi các dữ liệu tuần tự, đọc dữ liệu từ một nguồn và ghi dữ liệu ra một đích.

Stream hỗ trợ nhiều kiểu dữ liệu: nguyên thủy, đối tượng, các ký tự đặc biệt của ngôn ngữ các quốc gia*,*

Các đặc điểm của Stream bao gồm:

* Stream hỗ trợ hoàn hảo cho Lambda Expression.
* Stream không chứa các element của collection hay array.
* Stream là immutable object.
* Stream không dùng lại được, nghĩa là một khi đã sử dụng nó xong, chúng ta không thể gọi nó lại để sử dụng lần nữa.
* Chúng ta không thể dùng index để access các element trong Stream.
* Stream hỗ trợ thao tác song song các element trong collection hay array.
* Stream hỗ trợ thao tác lazy, khi cần thì thao tác mới được thực hiện. Để làm được điều này, hầu hết các thao tác với Stream đều return lại một Stream mới, giúp tạo một mắc xích bao gồm một loạt các thao tác nhằm thực thi các thao tác đó một cách tối ưu nhất. Mắc xích này còn được gọi là pipeline.

# Mô hình client-sever

**Mô hình client-server** là một mô hình nổi tiếng trong [mạng máy tính](https://vi.wikipedia.org/wiki/M%E1%BA%A1ng_m%C3%A1y_t%C3%ADnh" \o "Mạng máy tính), được áp dụng rất rộng rãi và là mô hình của mọi trang web hiện có. Ý tưởng của mô hình này là máy con (đóng vài trò là máy khách) gửi một yêu cầu (request) để máy chủ (đóng vai trò người cung ứng dịch vụ), máy chủ sẽ xử lý và trả kết quả về cho máy khách.