# Process (Tiến trình)

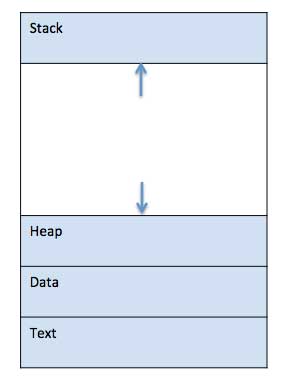
## Khái niệm:

Việc thực hiện công việc được mô tả thông qua các chương trình. Tiến trình nói 1 cách dễ hiểu là chương trình đang thực thi. Việc thực hiện một quy trình phải tiến triển theo một cách tuần tự.

Ví dụ như khi chúng ta viết 1 chương trình trong một trình soạn thảo văn bản và chạy chương trình thì nó sẽ trở thành tiến trình và bắt đầu làm các công việc được viết trong chương trình.

Khi một chương trình được load vào trong bộ nhớ, nó trở thành tiến trình, chúng ta có thể chia nó ra làm 4 section:

* Stack: ngăn xếp,
* Heap: Vùng
* Text: chữ
* Data: Kí tự



## Cấu trúc:

Stack: Tiến trình Stack bao gồm dữ liệu tạm thời như phương thức, tham số trong hàm, địa chỉ trả về và biến cục bộ.

Heap: vùng nhớ cấp phát động trong khi tiến trình đang chạy.

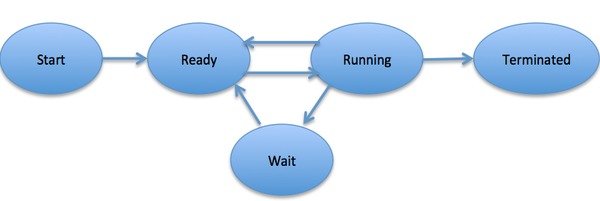
Text: bao gồm hoạt động hiện tại được thể hiện bằng giá trị của bộ đếm chương trình và nọi dung các thanh ghi của bộ xử lý.

Data: chứa biến toàn cục và biến static

Khi một tiến trình thực thi, nó đi qua nhiều trạng thái khác nhau. Các tầng trạng thái này có thể khác nhau tùy thuộc vào hệ điều hành, Và tên của những trạng thái này cũng không theo một chuẩn nào cả.

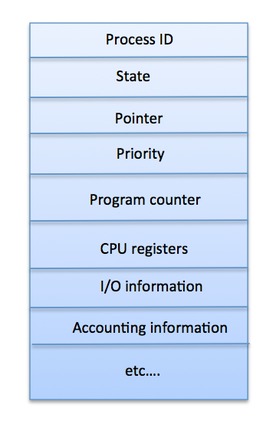
Tổng quát thì trong một thời điểm tiến trình có thể có một trong năm trạng thái sau:

* Star: Đây là trạng thái ban đầu khi một tiến trình được bắt đầu hay tạo lần đầu tiên.
* Ready: Trạng thái tiến trình đợi để được giao cho bộ xử lý. Một tiến trình sẵn sàng là tiến trình đợi để được hệ điều hành giao cho bộ xử lý để chúng có thể chạy. Tiến trình có thể có trạng thái này sau khi qua trạng thái Start, hay khi đang chạy nhưng bị ngắt bởi CPU hay các tiến trình khác.
* Running: một khi tiến trình đã được hệ điều hành giao cho bộ xử lý, trạng thái của tiến trình sẽ thành Running và bộ xử lý sẽ thực thi các lệnh của tiến trình.
* Waiting: Tiến trình vào trạng thái waiting nếu như nó cần đợi 1 tài nguyên, như là đợi input của người dung, hay file để sử dụng.
* Terminated or Exit:Khi một tiến trình thực thi xong, nó sẽ bị Terminated bởi hệ điều hành, Nó sẽ vào trạng thái Terminated và đợi để được xóa khỏi bộ nhớ chính.



## Process Control Block (PCB):

Khối quản lý tiến trình là một cấu trúc dữ liệu được duy trì bởi hệ điều hành cho mỗi tiến trình. PCB được định danh vởi một số nguyên PID (process ID). PCB giữ tất cả thông tin cần thiết để theo dõi quá trình được liệt kê dưới đây:

* Trạng thái tiến trình: Trạng thái hiện tại của tiến trình, như là ready, running, waiting,…
* Process privilege: Các phạm vi được truy cập, không được truy cập, hay các quyền tới tài nguyên hệ thống.
* PID: Số nguyên duy nhất cho tiến trình trong hệ điều hành.
* Pointer: Con trỏ trỏ tới các tiến trình cha.
* Program Counter: Là con trỏ trỏ tới địa chỉ của lệnh tiếp theo sẽ được thực thi trong tiến trình.
* CPU registers:Các thanh ghi CPU nơi tiến trình cần lưu trữ để thực thi các lệnh trong trạn thái running.
* CPU Scheduling Information: Mức độ ưu tiên của tiến trình và các thông tin lặp lịch khác cần để lập lịch tiến trình.
* Memory managerment Information: Bao gồm các thông tin như bảng trang, bộ nhớ giới hạn, bảng Segment của bộ nhớ được sử dụng bởi hệ điều hành.
* Accounting information: Bao gồm số lượng CPU được dung trong khi chạy tiến trình, giới hạn bộ nhớ, ID thực thi.
* IO status information: Bao gồm danh sách thiết bị I/O được phân phối trong tiến trình.

Khối PCB được duy trì trong suốt quá trình tiến trình thực thi và bị xóa khi tiến trình kết thúc.

## Cách thức Window quản lý process

# Thread (Luồng)

## Khái niệm:

Thread là một đơn vị cơ bản trong CPU, là một bước điều hành bên trong một process. Một luồng sẽ chia sẻ với các luồng khác trong cùng process về thông tin data, các dữ liệu của mình. Việc tạo ra thread giúp cho các chương trình có thể chạy được nhiều công việc cùng một lúc.

Trong một process có thể có nhiều thread, ví dụ khi chúng ta chạy ứng dụng Word, hệ điều hành tạo ra một process và bắt đầu chạy các thread chính của process đó.

## Phân loại:

Single-threaded: Đơn luồng: Trong một thời điểm chỉ cho phép 1 luồng chạy trên 1 tiến trình.

Multithreaded: Đa luồng: Trong một thời điểm cho phép nhiều luồng cùng chạy trên 1 tiến trình.

VD: một trang web, một thread sẽ đảm nhiệm việc chạy hình ảnh và bài viết, và một thread khác cùng lúc sẽ có nhiệm vụ nhận thêm các dữ liệu vào web.

Bên cạnh đó, thread cũng rất quan trọng đối với hệ thống RPC (Remote Procedure Call – hệ thống cho phép quá trình truyền tin giữa các tiến trình IPC – interprocess communication được diễn ra). Và hiển nhiên, RPC servers cũng là một dạng multithreaded. Khi một server PRC nhận được một tin nhắn, nó sẽ tạo ra một thread để giải quyết tin nhắn đó. Multithreaded giúp RPC có thể giải quyết nhiều yêu cầu cùng một lúc.

Multithreaded cũng được sử dụng rộng rãi trong nhân hệ hiều hành (operating system kernels). Có nhiều luồng hoạt động trong một kernel, và mỗi luồng đảm nhiệm một công việc riêng biệt, như quản lý thiết bị, quản lý bộ nhớ, quản lý ngắt… Một vài ví dụ có thể đưa ra như là: Solaris có một set những thread chuyên quản lý bộ xử lý ngắt; hay Linux có một kernel thread chuyên quản lý những vùng bộ nhớ trống trong hệ thống.

Phần lớn các phần mềm trong máy tính hiện đại đều có dạng multithreaded, tức đa luồng. Các ứng dụng trong máy tính đa phần đều chạy một process nhất định cùng với đó là nhiều luồng chạy bên trong.

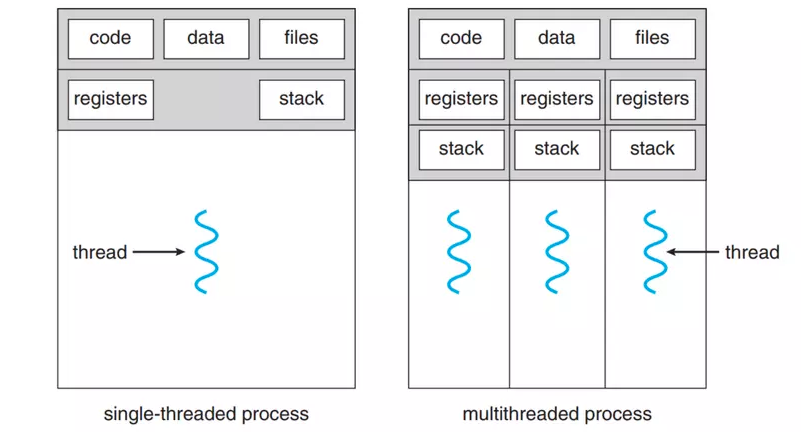
Lợi ích của Multithreaded: có 4 lợi ích chính:

**Khả năng đáp ứng:** Multithread giúp các ứng dụng tương tác có thể hoạt động tốt hơn vì ngay cả khi một phần chương trình bị block hoặc cần một thời gian dài để hoạt động, chương trình nhìn chung vẫn có thể chạy. Và điều này giúp người dùng hài lòng hơn vì khả năng đáp ứng cao của ứng dụng. Điều này đặc biệt đúng với người dùng là các designer. Bạn có thể hình dung: khi một chương trình đang chạy, và người dùng nhấn vào một nút lệnh mà cần rất nhiều thời gian để process, thì một hệ thống dạng single-threaded sẽ không kích hoạt bất kì hoạt động nào khác cho tới khi hoàn thành bước lệnh vừa rồi. Ngược lại, ứng dụng dạng multithread sẽ không làm gián đoạn quá nhiều quá trình của người dùng vì trong khi một thread được kích hoạt để thực hiện bước lệnh kia, một thread khác sẽ được kích hoạt để thực hiện bất kì bước lệnh ít tốn thời gian hơn mà người dùng yêu cầu.

**Khả năng chia sẻ tài nguyên:**các tiến trình chỉ có thể chia sẻ dữ liệu thông qua các kĩ thuật như shared memory (vùng bộ nhớ chung) và message sharing (chia sẻ tin). Các kĩ thuật này chỉ có thể được thiết lập bởi lập trình viên. Tuy nhiên, các luồng chia sẻ thông tin hoặc tài nguyên theo hệ thống được mặc định. Lợi ích của việc chia sẻ code và dữ liệu là nó giúp ứng dụng có nhiều threads hoạt động trong một vùng địa chỉ chung.

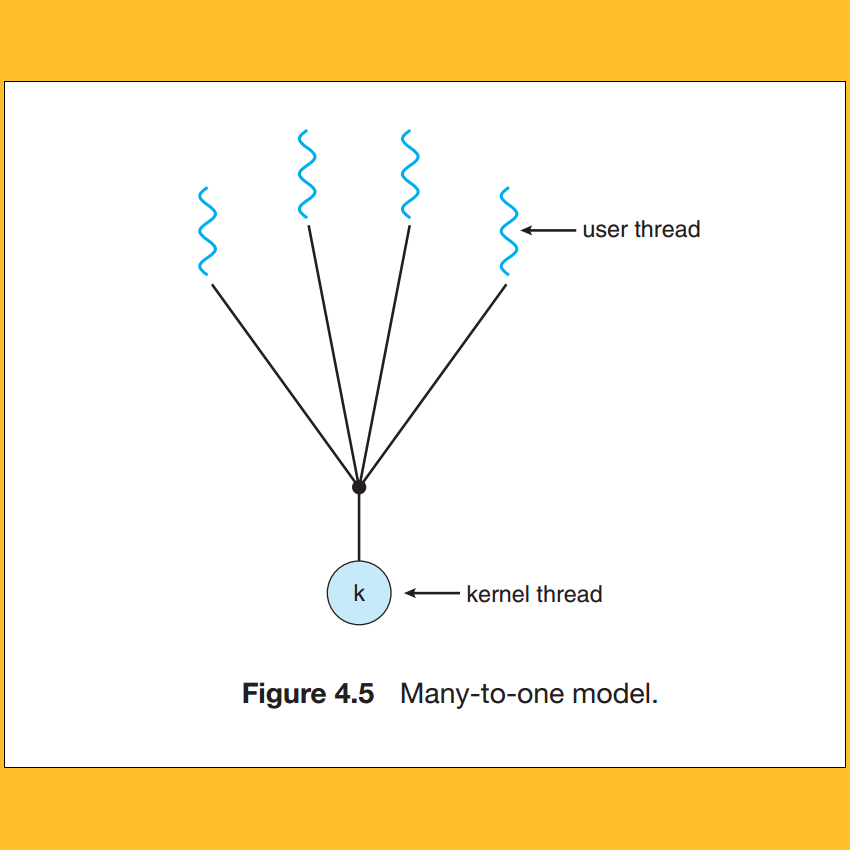
**Tiết kiệm:** việc cung cấp tài nguyên và dữ liệu cho quá trình tạo process rất tốn kém. Và vì threads tự động chia sẻ data cho process mà nó thuộc về, việc tạo các thread cho việc context-switch sẽ giúp tiết kiệm chi phí rất nhiều. Không chỉ chi phí mà còn là thời gian, vì việc tạo một process mới sẽ lâu hơn nhiều so với tạo một thread mới. Như trong Solaris, tạo ra một process lâu hơn 30 lần so với tạo ra một thread trong process đó, và lâu hơn 5 lần so với tạo một context-switch.

**Scalability:** Lợi ích của multithreaded thể hiện rõ hơn trong kiến trúc đa xử lý (multiprocessor architecture), vì multithread giúp các threads hoạt động song song trong các lõi xử lý khác nhau, trong khi đối với tiến trình dạng single-threaded, một thread chỉ có thể chạy trên một bộ xử lý, không quan trọng việc có bao nhiêu thread trong hệ thống hiện tại.

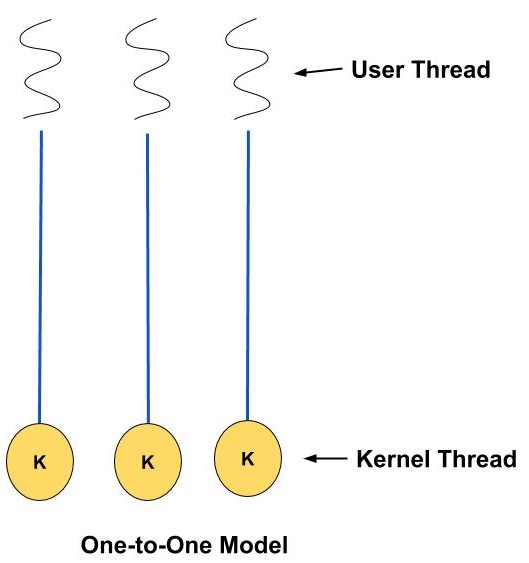


**Các mô hình trong multithreading**

**Mô hình many-to-one:** mô hình nhiều user threads nối vào một kernel thread.



**Mô hình one-to-one: là mô hình 1-1, một user thread kết nối với một kernel thread.**



**Mô hình many-to-many** chia các user-level threads cho một lượng nhỏ hơn hoặc bằng các kernel threads. Lượng kernel threads này tùy thuộc vào yêu cầu của ứng dụng sử dụng hoặc bộ máy sử dụng (một ứng dung thường dùng nhiều kernel threads trên multiprocessors hơn là trên single processor).

