**Hệ điều hành là gì?**

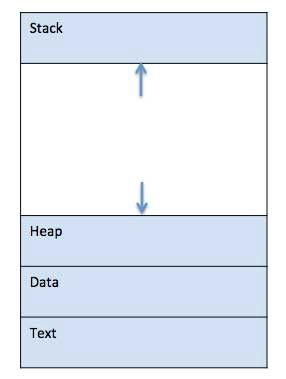
Trước khi nói về hoạt động, mình sẽ giải thích cơ bản một chút về hệ điều hành. Nhìn chung, hệ điều hành là một phần mềm máy tính, nhưng không giống như các phần mềm quen thuộc chúng ta hay dùng hàng ngày như trình duyệt, trình soạn thảo văn bản hay trình biên tập file. Hệ điều hành là tập hợp của các chương trình có chức năng quản lý phần cứng cũng như cung cấp các dịch vụ cần thiết cho các phần mềm khác hoạt động. Ngoài ra, hệ điều hành hầu hết sẽ che giấu đi những xử lý tính toán phức tạp, tính toán việc sử dụng tài nguyên cũng như cung cấp các chức năng về cách ly (để tránh tối đa xung đột giữa phần cứng với phần mềm) và tạo một vài lớp bảo vệ tránh tấn công từ bên ngoài (điển hình nhất là Firewall). Hệ điều hành bao gồm 3 thành phần, 3 yếu tố cấu thành và 2 nguyên tắc thiết kế

* 3 thành phần chính là File system (các tập tin hệ thống ), Scheduler(bộ lập lịch) và Device driver(trình điều khiển thiết bị ).
* 3 yếu tố cấu thành bao gồm Abstraction(yếu tố trừu tượng)(gồm có process, thread, file, socket và memory), Mechanisms(yếu tố cơ chế)(gồm có create, schedule, open, write, allocate) và Policies(yếu tố chính sách)(gồm có [LRU](https://en.wikipedia.org/wiki/Cache_replacement_policies#Least_recently_used_(LRU)) và [EDF](https://en.wikipedia.org/wiki/Earliest_deadline_first_scheduling))
* 2 nguyên tắc thiết kế bao gồm: Separation of mechanism and policy (đảm bảo thiết kế các cơ chế dễ đáp ứng việc bổ sung chính sách) và Optimization for common case (Tối ưu cho các trường hợp phổ biến)

Ngay sau đây, mình sẽ trình bày một số hoạt động chính của hệ điều hành mà mình tìm hiểu được, do số lượng nội dung khá nhiều nên mình sẽ chia bài viết làm 3 phần. Trong phần 1 này, mình sẽ nói về 3 vấn đề: (1) Quản lý tiến trình, (2) Luồng và xử lý đồng bộ và (3) Cơ chế lập lịch.

**1. Quản lý tiến trình**

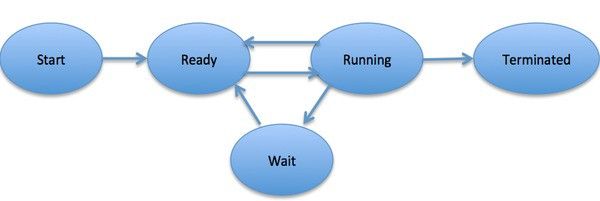
Một tiến trình về cơ bản là một chương trình đang ở trong trạng thái thực thi. Việc thực thi của các tiến trình buộc phải diễn ra theo kiểu tuần tự. Đơn giản là, danh sách các chương trình muốn thực thi sẽ được điền vào một tệp văn bản và khi máy tính hoạt động, nội dung của tệp trở thành thứ tự thực thi các tiến trình tương ứng, nội dung của tệp sẽ được thay đổi theo thời gian. Mặc dù đã từng có nhiều nhận định cho rằng máy tính có thể xử lý song song, tuy nhiên đó là với các vi xử lý hiện đại ngày nay có chứa nhiều nhân (core) hoạt động độc lập, còn thực tế các tiến trình được quản lý bởi một nhân xử lý sẽ luôn hoạt động tuần tự, chẳng qua việc xử lý diễn ra nhanh đến mức mắt người không thể thấy được. Các chương trình sẽ luân phiên hoạt động để đánh lừa mắt người rằng chúng đang hoạt động song song ^^. Khi một chương trình được nạp vào bộ nhớ và trở thành một tiến trình, nó có thể được chia thành bốn phần - Stack, Heap, Data và Text. Hình ảnh sau đây cho thấy bố cục đơn giản của một tiến trình bên trong bộ nhớ chính:



Trong đó,

* **Stack**: Chứa dữ liệu tạm thời, thường chỉ chứa dữ liệu dùng trong thời gian rất ngắn hoặc dùng một lần.
* **Heap**: Đây là vùng bộ nhớ được cấp phát động cho một tiến trình trong thời gian thực thi.
* **Data**: Chứa các biến global (biến dùng chung ở cấp toàn bộ chương trình) và static (loại biến dùng chung ở cấp struct).
* **Text**: Bao gồm hoạt động hiện tại được biểu thị bằng giá trị của Program Counter và nội dung của các thanh ghi trên vi xử lý.

Khi một tiến trình thực thi, nó có thể đi qua các trạng thái khác nhau. Các giai đoạn này có thể khác nhau đối với các hệ điều hành khác nhau và tên của các trạng thái này cũng không được chuẩn hóa cụ thể. Nhưng nhìn chung, một tiến trình có thể có một trong năm trạng thái sau đây tại một thời điểm:



* **Start**: Trạng thái khi một tiến trình được tạo hoặc được khởi động lần đầu tiên.
* **Ready**: Tiến trình đang chờ để được chuyển cho vi xử lý. Thường thì khi một tiến trình khác đang chạy, **CPU** sẽ không cho phép một tiến trình khác xen vào mà sẽ phải đợi tới lượt để đảm bảo tiến trình hiện tại đã xử lý xong (lưu ý, tiến trình xử lý xong không có nghĩa là nó sẽ kết thúc). Trạng thái này thường xuất hiện ngay sau trạng thái **Start**.
* **Running**: Trạng thái thực thi của tiến trình, khi bộ lập lịch *Scheduler* đã chuyển nó cho vi xử lý, vi xử lý sẽ tiến hành thực hiện các tính toán hoặc chỉ dẫn trên nó.
* **Wait**: Trạng thái này xảy ra nếu tiến trình cần chờ một tài nguyên để có thể sử dụng, ví dụ như chờ người dùng nhập liệu hoặc chờ đợi tệp tin đang được sử dụng bởi tiến trình khác. (Vấn đề này có thể được thảo luận liên quan đến xử lý I/O)
* **Terminated**: Xảy ra khi một tiến trình đã xử lý xong hoặc bị ngắt bởi yêu cầu từ hệ thống hoặc người dùng, trạng thái này cũng có thể được gọi là **Exit**.

Để lưu dữ liệu hoạt động của các tiến trình, hệ điều hành sẽ sử dụng một cấu trúc dữ liệu gọi là Process Control Block (PCB). Mỗi PCB được xác định bởi một số nguyên dương (bắt đầu từ 1), gọi là Process ID (PID). PCB thường sẽ bao gồm các thông tin như sau:

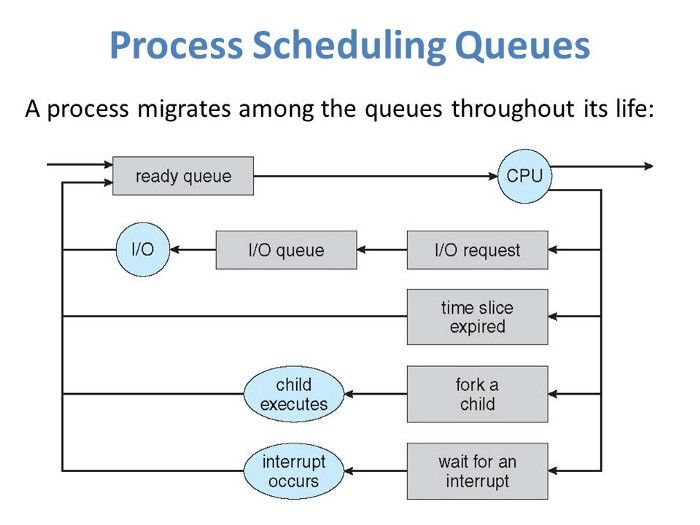


* **Process ID (PID)**: Số nguyên xác định định danh của tiến trình.
* **State**: Trạng thái hiện tại của tiến trình (là một trong các trạng thái ở phần trên)
* **Pointer**: Con trỏ lưu địa chỉ của tiến trình cha.
* **Priority**: Độ ưu tiên của tiến trình, giúp cho vi xử lý xác định thứ tự thực hiện.
* **Program Counter**: Con trỏ lưu địa chỉ của chỉ dẫn tiếp theo giúp cho tiến trình thực thi (là một phép tính hoặc tương tự).
* **CPU Registers**: Các thanh ghi tiến trình cần sử dụng để thực thi.
* **I/O Information**: Thông tin về các thiết bị đọc - ghi mà tiến trình cần sử dụng
* **Accounting Information**: Chứa các thông tin về việc sử dụng CPU như thời gian sử dụng, định danh ....

**2. Cơ chế lập lịch**

**Scheduling** (lập lịch) là nhiệm vụ của bộ xử lý, nó sẽ tiến hành loại bỏ các tiến trình đang chạy khỏi CPU và lựa chọn một danh sách tiến trình khác trên cơ sở một chiến lược cụ thể. (Có thể danh sách mới sẽ bao gồm một số tiến trình trong danh sách cũ). Lập lịch là một trong các tiêu chí của việc thiết kế hệ điều hành đa nhiệm (multi-task). Các hệ điều hành này cho phép nhiều hơn một tiến trình được nạp vào bộ nhớ để thực thi tại một thời điểm và tiến trình được nạp sẽ chia sẻ thời gian CPU bằng cách sử dụng ghép kênh thời gian (multiplexing). Hệ điều hành duy trì tất cả các **PCB** (đã nhắc đến ở cuối phần 1) trong Process Scheduling Queues (hàng đợi lập lịch tiến trình). Đồng thời, mỗi trạng thái của tiến trình sẽ có một hàng đợi riêng được duy trì (1 hàng đợi riêng cho ready, 1 hàng đợi riêng cho running, ...) và PCB của tất cả các tiến trình trong cùng trạng thái thực thi được đặt trong cùng một hàng đợi. Khi trạng thái của một tiến trình được thay đổi, PCB của nó sẽ bị xóa liên kết khỏi hàng đợi hiện tại và được chuyển sang hàng đợi trạng thái mới (áp dụng thuật toán danh sách liên kết cơ bản và cơ chế truy xuất theo địa chỉ). Ngoài các hàng đợi theo trạng thái, hệ điều hành còn duy trì một số hàng đợi như:

* **Job queue**: Hàng đợi lưu trữ mọi tiến trình đang ở trong hệ thống
* **Ready queue**: Chứa các tiến trình mới được khởi tạo.
* **Device queue**: Nếu một tiến trình cần chờ để có thể sử dụng thiết bị I/O thì được đưa vào đây. Dưới đây là hình ảnh mô ta một tiến trình từ khi bắt đầu cho đến khi rời khỏi CPU:



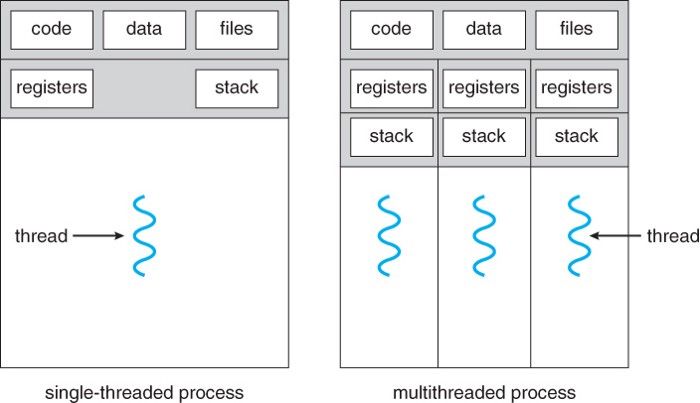
Một số chính sách thường sẽ được sử dụng để lập lịch bao gồm [FIFO](https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)), [Round Robin](https://en.wikipedia.org/wiki/Round-robin_scheduling) hoặc [Priority](https://en.wikipedia.org/wiki/Priority_queue) ... Theo như trên hình, khi một tiến trình rời khỏi **Ready queue** để bắt đầu thực thi, nó có thể có 4 khả năng:

* Request I/O: tiến trình cần xử lý liên quan đến đọc - ghi => Sẽ được chuyển vào **Device queue**.
* Timeout: tiến trình đã dùng hết thời gian được cấp phép bởi **CPU**, sẽ được đưa trở lại vào **Ready queue** để đợi đến lượt tiếp theo.
* Fork child: tiến trình tạo ra tiến trình con để tiếp tục xử lý, thường là các tiến trình phức tạp hoặc cần xử lý đồng thời nhiều công việc.
* Interrupt: tiến trình bị gián đoạn do một vấn đề nào đó đột ngột phát sinh.

Một khái niệm khác có thể được nhắc tới là **Context switch** (kỹ thuật chuyển ngữ cảnh). Đây là cơ chế lưu trữ và khôi phục trạng thái hoặc ngữ cảnh của CPU trong PCB. Nó cho phép thực hiện tiến trình được bắt đầu lại từ cùng một trạng thái kết thúc trước đó. Giống như khi ta ghi dữ liệu lên một file, sau đó khi sửa file này ta sẽ tiếp tục với nội dung được lưu trước đó thay vì bắt đầu lại từ đầu. Kỹ thuật này cho phép nhiều tiến trình cùng chia sẻ một CPU. **Context switch** là một tính năng thiết yếu của hệ điều hành đa nhiệm. Khi bộ lập lịch chuyển CPU từ việc thực thi một tiến trình này sang một tiến trình khác, trạng thái từ tiến trình đang chạy hiện tại được lưu trữ vào PCB. Với lần thực thi tiếp theo, trạng thái cho tiến trình đó được tải từ PCB của chính nó và được sử dụng để thiết lập bộ đếm chương trình, các thanh ghi, v.v. Tại thời điểm đó, tiến trình thứ hai có thể bắt đầu thực thi. Việc tính toán **Context switch** rất chuyên sâu, vì thanh ghi và trạng thái bộ nhớ phải được lưu và khôi phục lại đúng như giai đoạn thực thi trước đó. Để tránh thời gian chuyển đổi quá lâu, một số hệ thống phần cứng sử dụng hai hoặc nhiều thanh ghi của bộ xử lý để tính toán. Khi tiến trình được chuyển đổi bởi **Context switch**, các thông tin sau sẽ được lưu trữ để sử dụng: bộ đếm chương trình, thông tin lập lịch, giá trị thanh ghi cơ sở và giới hạn, thanh ghi hiện được sử dụng, trạng thái thay đổi, thông tin I/O và thông tin kế toán.

**3. Luồng và xử lý đồng bộ**

Luồng (hay Thread) là một quy trình thực hiện thông qua mã tiến trình, một tiến trình có thể có nhiều luồng thực hiện song song. Nó có bộ đếm chương trình (Program Counter) riêng để theo dõi chỉ dẫn nào sẽ thực hiện tiếp theo. Nó cũng có các thanh ghi hệ thống chứa các biến làm việc hiện tại và một ngăn xếp chứa lịch sử thực hiện các chỉ dẫn. Một luồng chia sẻ với các luồng ngang hàng của nó các thông tin khác nhau như code segment, data segment và các file đang mở. Khi một luồng làm thay đổi bất cứ thông tin nào kể trên, tất cả các luồng khác sẽ thấy. Luồng cũng có thể được gọi là một tiểu tiến trình. Luồng cung cấp một cách để cải thiện hiệu suất ứng dụng thông qua việc xử lý song song. Luồng đại diện cho một cách tiếp cận phần mềm để cải thiện hiệu suất của hệ điều hành bằng cách giảm chi phí xử lý. Một luồng tương đương với một tiến trình cổ điển. Mỗi luồng thuộc về chính xác một tiến trình và không có luồng nào có thể tồn tại bên ngoài một tiến trình (nói cách khác, không có tiến trình thì không có luồng). Luồng thường được áp dụng trong các máy chủ network hoặc máy chủ web, đáp ứng thực thi song song trong một hệ thống dùng chung bộ nhớ.



Hình ảnh trên cho thấy trừ là các thanh ghi và ngăn xếp ra thì mọi dữ liệu khác đều được dùng chung bởi luồng, khác với tiến trình ở việc tất cả đều được dùng riêng.

Các ưu điểm của luồng bao gồm:

* Giảm thời gian chuyển ngữ cảnh (\*\*Context switch)
* Các tài nguyên được cung cấp đồng thời giữa các luồng của cùng một tiến trình.
* Việc giao tiếp giữa các luồng luôn đạt hiệu quả cao (thay vì là tạo tiến trình con)
* Tiết kiệm chi phí hơn việc dùng đa tiến trình.
* Cho phép tận dụng hiệu quả hơn với vi xử lý hiện đại.

**Kết luận**

Vậy là mình đã xong phần thứ nhất của bài tìm hiểu cơ bản về hệ điều hành cũng như các hoạt động của nó. Ở phần thứ hai của bài viết, mình sẽ tiếp tục tìm hiểu và trình bày về các vấn đề như: **Memory Management** (Quản lý bộ nhớ), **Input/Output Management** (Quản lý vào/ra) và **Virtualization** (Công nghệ ảo hóa phần cứng). Cảm ơn mọi người đã đọc bài viết ^^.

**Tài liệu tham khảo**

* <https://medium.com/cracking-the-data-science-interview/how-operating-systems-work-10-concepts-you-should-know-as-a-developer-8d63bb38331f>
* <https://edu.gcfglobal.org/en/computerbasics/understanding-operating-systems/1/>
* <https://computer.howstuffworks.com/operating-system.htm>
* <https://en.wikipedia.org/wiki/Operating_system>

v