**KNOWLEDGE ABOUT THREAD**

1. Luồng CPU là gì?

Thread là một đơn vị cơ bản trong CPU. 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.

2. Sự khác nhau giữa single – threaded và multithreaded

Có hai khái niệm ta cần xem qua đó là single – threaded và multithreaded.

- 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. Bạn có thể hình dung thế này: trong 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.

- Các ứng dụng cũng có thể được thiết kế để tận dụng khả năng xử lý trên các hệ thống multicore, giúp thực hiện nhiều CPU task song song.

- Trong nhiều trường hợp nhất định, một ứng dụng có thể được yêu cầu thực hiện (request) nhiều nhiệm vụ giống nhau. Ví dụ: một web server nhận lệnh từ khác hàng nhấn vào trang web, hình ảnh, âm thanh… và tất nhiên, một web server có thể phải nhận rất nhiều (hàng nghìn, hàng triệu cho đến hàng trăm triệu) yêu cầu cùng một lúc. Vì thế nếu web server đó chạy theo dạng **single-threaded**, tức là chỉ một khách hàng được giải quyết yêu cầu trong một khoảng thười gian, thì những khách hàng khác sẽ phải đợi rất lâu để mình có thể access vào trang web. Trước khi có **multithreaded**, một cách giải quyết cho vấn đề này đó là web server sẽ chạy một process nhận nhiều requset cùng lúc, và với một request được tiếp nhận, nó sẽ tạo ra một process khác để giải quyết request đó. Điều này sẽ tốn nhiều thời gian và nguồn lúc. Multithreads giúp giải quyết vấn đề này. Thay vì tạo ra một process mới y chang process đang có, chúng ta chỉ cần một process duy nhất có nhiều luồng cùng chạy với nhau. **Khi server nhận được một yêu cầu từ khách hàng, nó sẽ tạo ra một luồng mới để luồng đó giải quyết yêu cầu nhận được, trong khi đó, server sẽ quay lại với những yêu cầu tiếp theo**. (giống kiểu download nhiều tệp).

- 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 server 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.

- Và cuối cùng, multithreaded cũng được sử dụng rộng rãi trong nhân hệ điều hành (operating system kernels). Có nhiều luồng hoạt động trong một kernel, và mỗi luồng đảm nhiệm một công việc riêng, như là 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.

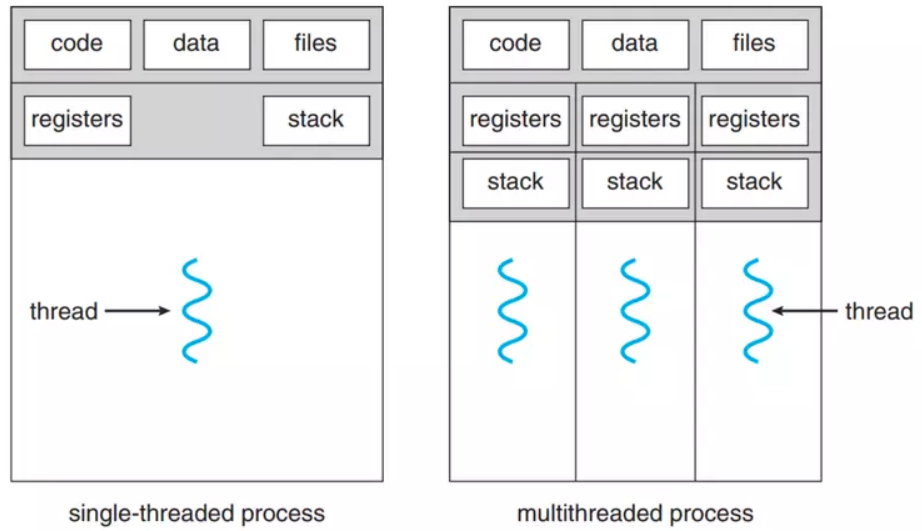
- Lợi ích của Miltithreaded: có 4 lợi ích chính:

+ Khả năng đáp ứng: Multithread giúp các ứng dụng tương tác có thể hoạt động tốt hơn vì ngay cả khi môt phần chương trình bị block hoặc cần một thời gian dài 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 kỳ 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 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 kỳ 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 kỹ thuật như shared memory (vùng bộ nhớ chung) và message sharing (chia sẻ tin). Các kỹ 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**

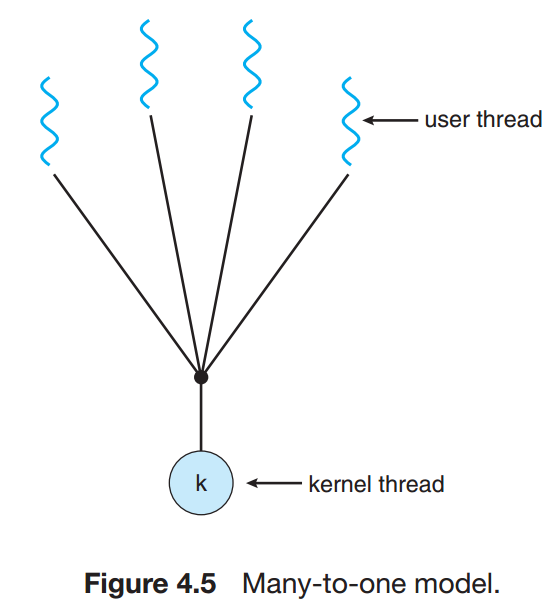
Sau phần bài tổng quan về luồng – threads, bài viết này sẽ nói về các mô hình đa luồng (multithreading models). Trước hết, threads có thể được hỗ trợ qua hai cách sau: thông qua người dùng, để thành lập user threads, và thông qua nhân máy tính, để hình thành kernel threads. User threads được hỗ trợ trên kernel và được quản lý không cần sự hỗ trợ từ kernel, trong khi đó kernel threads được hỗ trợ và quản lý trực tiếp từ hệ điều hành. Các hệ điều hành hiện nay như Window, Linux, Mac OS X, Solaris đều hỗ trợ kernel threads.

Dù là hai đơn vị riêng biệt, user threads và kernel threads có mối liên hệ không thể tách rời. Ba mô hình sau cũng là ba cách để hình thành mối quan hệ giữa user threads và kernel threads: mô hình Many-to-one, mô hình one-to-one, và mô hình one-to-many.

Mỗi hoặc nhiều user thread phải được map sang một hoặc nhiều kernel thread tương ứng để được xử lý bởi hệ điều hành.

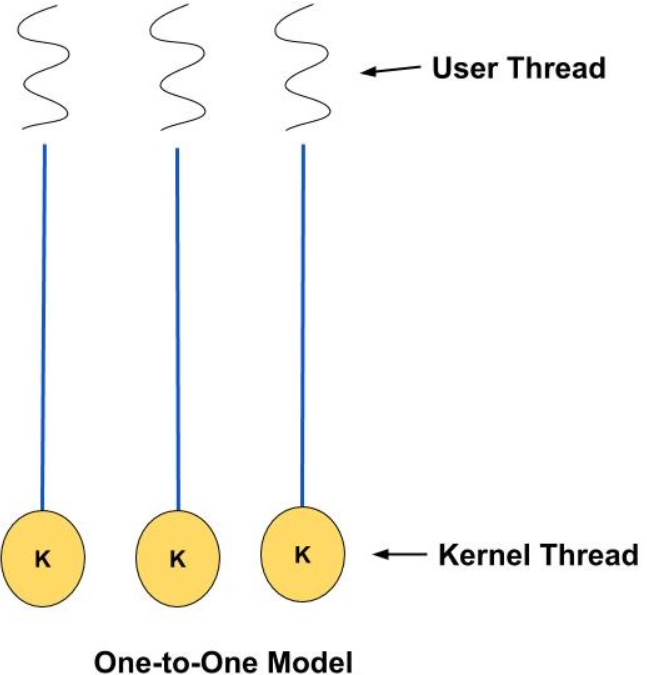
**Mô hình many-to-one**

**Mô hình many-to-one** là mô hình nhiều user threads nối vào một kernel thread. Việc quản lý các luồng này dựa vào thư viện luồng trong không gian người dùng. Tuy nhiên, cả hệ thống sẽ bị chặn nếu một luồng nào đó thực hiện một blocking system call. Và cũng vì chỉ một luồng được tiếp cận kernel trong một lần, các threads khác không thể chạy song song trong hệ thống đa lõi. Green threads – tên gọi của một thư viện luồng của hệ thống Solaris và đã được sử dụng trong những versions cũ của Java – sử dụng mô hình many-to-one. Không có quá nhiều hệ thống sử dụng mô hình này vì nó không sử dụng được lợi thế của multiple processing cores.



### ****Mô hình one-to-one****

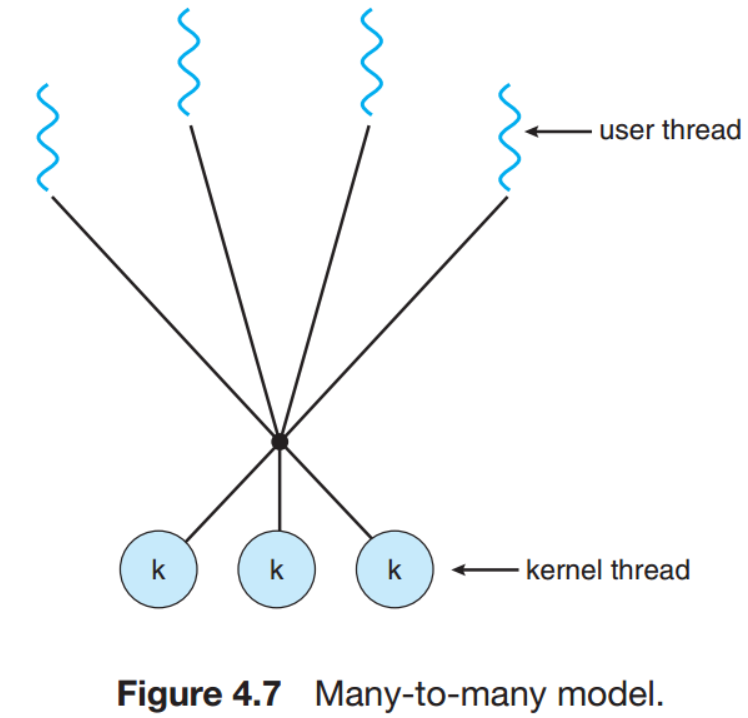
**Mô hình one-to-one**là mô hình 1-1, một user thread kết nối với một kernel thread. Mô hình 1-1 này đảm bảo được tính liên tục vì nếu một thread bị block thì một thread khác sẽ vẫn kết nối được với kernel. Nó cũng đảm bảo được nhiều luồng có thể hoạt động cùng một lúc trong bộ đa xử lý. Khuyết điểm duy nhất của mô hình này là nó đòi hỏi khi một user thread hoạt động thì một kernel thread phải được kích hoạt theo. Và vì quá nhiều kernel threads sẽ gây nên sự quá tải trong ứng dụng, những app sử dụng model này đều giới hạn số lượng thread được tạo ra trong hệ thống. Linux và Windows là những hệ điều hành sử dụng model one-to-one.



### ****Mô hình many-to-many****

**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). Mô hình này khác mô hình many-to-one ở tính liên lục. Trong mô hình many-to-one, người dùng có thể tạo bao nhiêu user threads tùy thích, nhưng nó không đảm bảo được tính liên tục vì một kernel chỉ kết nối được với một user thread, như đã nói ở trên. Mô hình one-to-one cho phép tính liên tục cao hơn, tuy nhiên số lượng threads được tạo ra rất quan trọng nếu bạn không muốn ứng dụng của mình bị quá tải.

Mô hình many-to-many sẽ giải quyết được vấn đề của hai mô hình trước: số lượng user threads tạo ra là tùy thích, và kernel thread tương ứng có thể chạy song song trong hệ đa xử lý. Và, khi một thread đang thực hiện blocking system call, kernel threads tương ứng có thể chuyển sang một user thread khác và giải quyết user thread đó. Một biến thể của many-to-many model là two-level model. Model này như là sự kết hợp giữa many-to-many model và one-to-one model, vì nó vừa chia các user-level threads cho một lượng nhỏ hơn hoặc bằng các kernel threads tương ứng, và vừa cho phép một user thread kết nối riêng với một kernel thread. Trước version Solaris 9, hệ điều hành Solaris đã sử dụng two-level model. Tuy nhiên, từ version 9 trở đi, Solaris sử dụng one-to-one model.



## ****Hyperthreading là gì****

Hyperthreading là khi CPU có khả năng cho một core đơn thực thi nhiều hơn một luồng cùng một lúc. Hyperthreading có khả năng tăng mức độ xử lí/ tạo nhiều luồng hơn cho nhiều core, nhưng không phải tất cả core.. Tùy thuộc vào một nhiệm vụ đang làm, hyperthreading có thể giúp mang lại hiệu năng khác nhau trong từng core khác nhau, nhưng đôi khi tổng thể có thể hụt hiệu năng.

Với công nghệ hiện nay, một core có thể sản sinh ra 2 threads, đó là lý do có nhiều CPU đời mới, ví dụ: [ryzen 5 1600](https://www.amd.com/en/products/cpu/amd-ryzen-5-1600)với số core, thread lần lượt là 6, 12. Tuy vậy, cũng có những con CPU đời mới, ví dụ: [Intel Core i5 7400](https://ark.intel.com/products/97147/Intel-Core-i5-7400-Processor-6M-Cache-up-to-3_50-GHz)(Kaby Lake), lại chỉ có số core, thread lần lượt là 4, 4. Do đó, tùy thuộc vào công nghệ sử dụng của từng hãng, cho từng con CPU mà số core, số thread nó lại khác nhau.

**Kết luận, Hyperthreading là khi CPU có khả năng cho một core đơn thực thi nhiều hơn một luồng cùng một lúc.**