



# *Chương 2: Process & Threads* *(Tiến trình & Luồng)*

---

*Khái niệm tiến trình, luồng và các  
vấn đề quản lý luồng, tiến trình*



# Nội dung

---

- Tiến trình:
  - Khái niệm tiến trình
  - Lập lịch tiến trình
  - Các hoạt động trên tiến trình
  - Các tiến trình hợp tác (Cooperating Processes)
  - Liên lạc liên tiến trình (Process Communication)
- Luồng( tiến trình mức thấp-tiểu trình):
  - Mô tả luồng
  - Các mô hình đa luồng



# 1.1. Khái niệm tiến trình(1)

---

- Việc thực hiện công việc được mô tả thông qua các chương trình.
- Khi chương trình hoạt động, nó chuyển thành tiến trình; để thực hiện, tiến trình cần
  - Được cung cấp đầy đủ tài nguyên cần thiết
  - Được CPU tiếp nhận & thực hiện
- Hệ điều hành: điều phối việc thực hiện các tiến trình cũng như phân phối tài nguyên cần thiết cho tiến trình
- Một tiến trình gồm:
  - Mã nguồn chương trình (code) (không thay đổi)
  - Dữ liệu (data)
  - Bộ đếm CT (Program Counter)
  - Ngăn xếp (Stack)
  - Giá trị ở các thanh ghi (Register values)



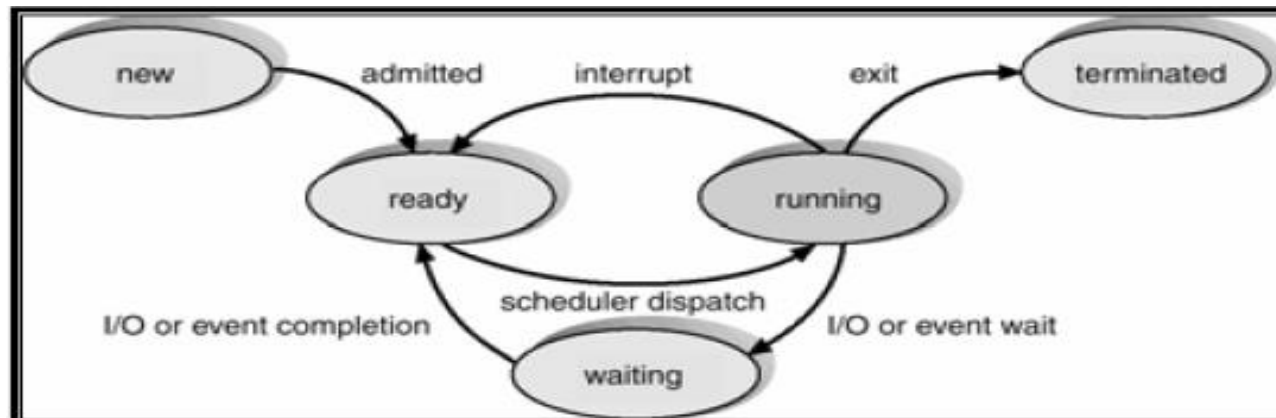
## 1.1.1. Các trạng thái tiến trình(1)

---

- Trạng thái của tiến trình tại một thời điểm xác định bởi hoạt động của tiến trình tại thời điểm đó.
- Trong quá trình sống, tiến trình có thể thay đổi trạng thái do các nguyên nhân:
  - Phải dừng hoạt động do hết thời gian
  - Đợi một thao tác I/O hoàn tất
  - Phải chờ một sự kiện xảy ra

## 1.1.1. Các trạng thái tiến trình(2)

- Tại một thời điểm, tiến trình có thể có một trong các trạng thái:
  - **new**: Tiến trình đang được tạo
  - **running**: Tiến trình đang chiếm hữu CPU & thực hiện các lệnh.
  - **waiting**: Tiến trình *đang chờ cung được cấp tài nguyên* hoặc *chờ một sự kiện nào đó xuất hiện* để chuyển sang trạng thái sẵn sàng.
  - **ready**: Tiến trình ở trạng thái sẵn sàng, được phân phối đủ tài nguyên cần thiết, đang chờ đến lượt được thực hiện theo cơ chế lập lịch của hệ điều hành.
  - **terminated**: Tiến trình kết thúc. Nó không biến mất cho đến khi một tiến trình khác đọc được trạng thái thoát của nó.





## 1.1.1. Các trạng thái tiến trình(2)

### ■ Hoạt động(quá trình chuyển trạng thái)

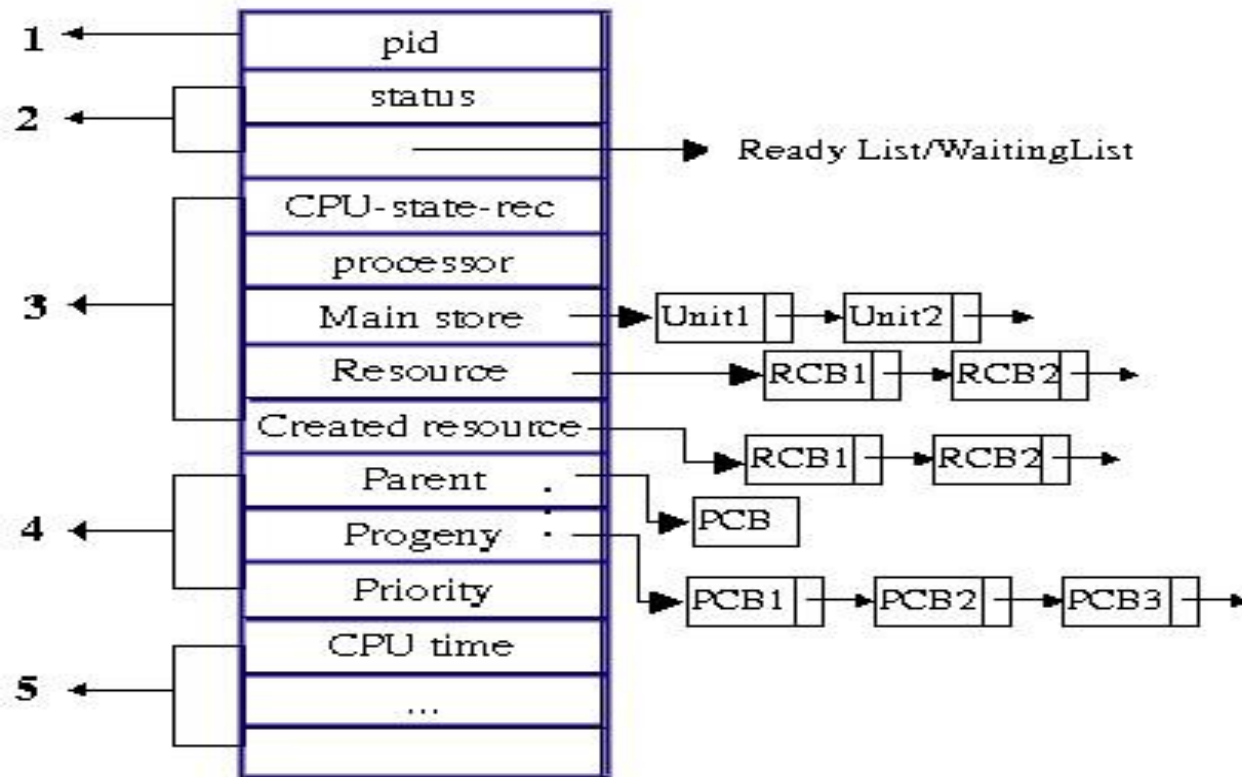
- Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái **running**. Trong khi đó, nhiều tiến trình có thể ở trạng thái **waiting** hay **ready**.
- Tiến trình mới tạo được đưa vào hệ thống, được cung cấp đủ tài nguyên ở trạng thái **ready**(chờ được phân phối CPU để thực hiện)
- Khi tiến trình đang thực hiện(running), nó có thể chuyển sang trạng thái:
  - **Kết thúc(terminal)** nếu thực hiện xong
  - **Chờ(waiting)** tiến trình yêu cầu một tài nguyên nhưng chưa được đáp ứng vì tài nguyên chưa sẵn sàng để cấp phát tại thời điểm đó ; hoặc tiến trình phải chờ một sự kiện hay thao tác nhập/xuất
  - **Sẵn sàng(ready)** khi xảy ra ngắt để chuyển CPU cho tiến trình có mức ưu tiên cao hơn Bộ điều phối cấp phát cho tiến trình một khoảng thời gian sử dụng CPU hoặc hết thời gian chiếm hữu CPU
- Bộ điều phối chọn một tiến trình khác có trạng thái ready cho xử lý.
- Tài nguyên mà tiến trình yêu cầu trở nên sẵn sàng để cấp phát ; hay sự kiện hoặc thao tác I/O tiến trình đang đợi(có trạng thái waiting) hoàn tất, tiến trình chuyển sang ready

# 1.1.2. Khởi điều khiển tiến trình

## Process Control Block (PCB)(1)

- **PCB:** là vùng nhớ lưu trữ các thông tin mô tả cho tiến trình; **mỗi tiến trình có một PCB**
- **Cấu trúc PCB:**
  - 1. Định danh tiến trình(Pid-Process Id): để phân biệt các proces
  - 2. Trạng thái tiến trình - Process state: xác định trạng thái hiện thời
  - 3. Ngưỡng cảnh tiến trình: mô tả các tài nguyên liên quan đến tiến trình(hiện có hoặc đang đợi phân bổ)
    - Trạng thái CPU: Con trỏ lệnh, CPU registers; được lưu trữ khi xảy ra ngắt để có thể phục hồi trạng thái khi phục vụ ngắt xong
    - Thông tin lịch trình CPU - CPU scheduling information
    - Thông tin quản lý bộ nhớ: danh sách khối nhớ đang cấp cho tiến trình
    - Tài nguyên sử dụng: danh sách tài nguyên tiến trình đang sử dụng
    - Tài nguyên tạo lập: danh sách các tài nguyên mà tiến trình yêu cầu
  - 4. Thông tin giao tiếp; phản ánh quan hệ giữa tiến trình này với các tiến trình khác trong hệ thống
  - 5. Thông tin thống kê: những thông tin về hoạt động tiến trình(t thực hiện, t chờ...)

## 1.1.2. Khôi điều khiển tiến trình Process Control Block (PCB)(2)



**Structure Of PCB**

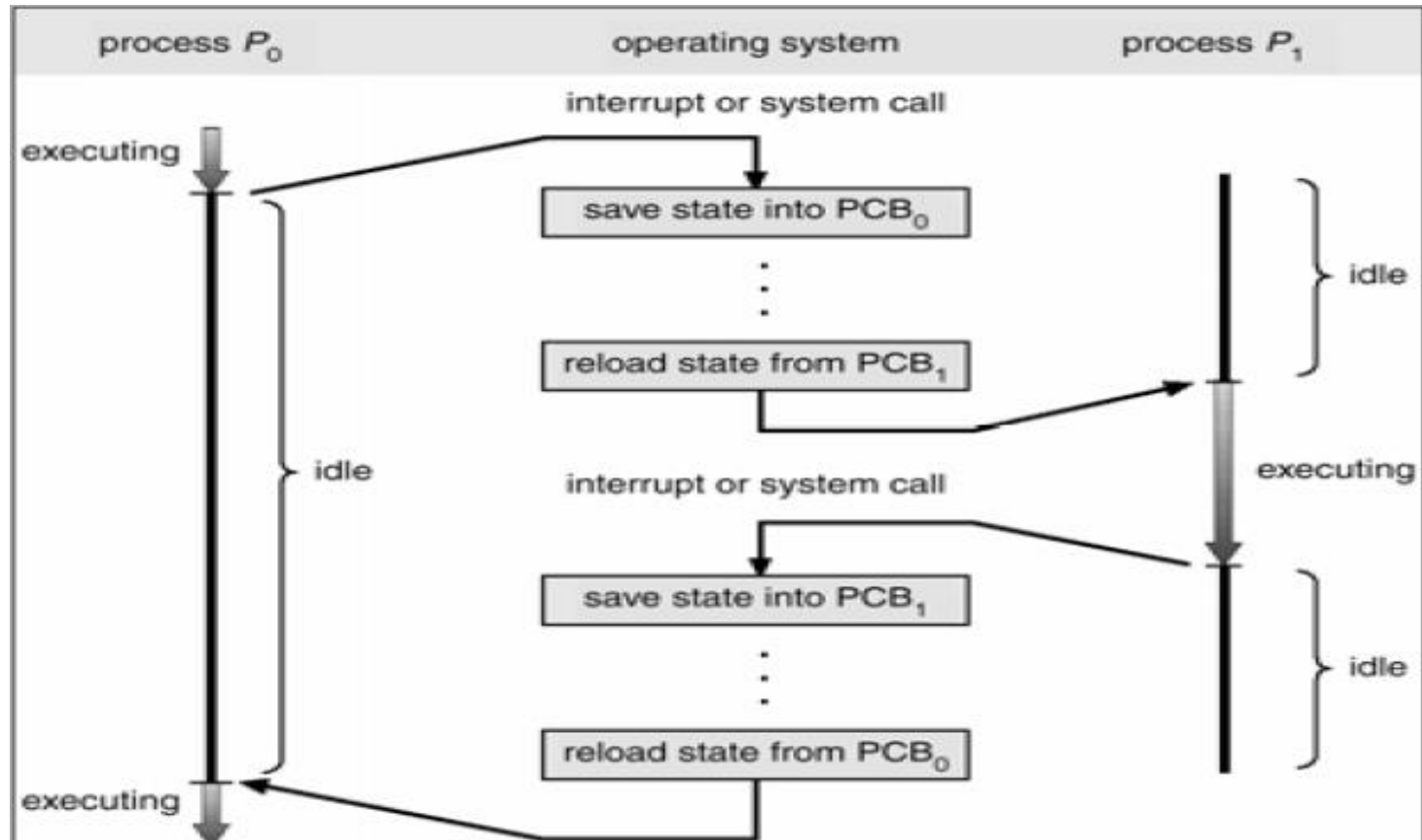


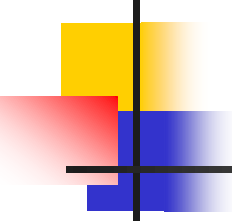
# 1.1.2. Khối điều khiển tiến trình

## Process Control Block (PCB)(3)

- Các PCB thường liên kết với một số hàng đợi để điều phối CPU
  - PCB sẽ quyết định tiến trình nào sẽ được sử dụng CPU
- Hệ điều hành căn cứ vào nội dung của PCB để:
  - Phân phối và phân phối lại CPU
  - Giải phóng CPU ảo mà không phân phối lại
    - Trong chế độ đa chương trình, user quan niệm nhiều ctr thực hiện đồng thời nhưng khi thực hiện CPU chỉ phục vụ một ctr tại một thời điểm(CPU thực); các ctr đang *thực hiện đồng thời* còn lại sử dụng *CPU ảo*
    - CPU ảo là CPU logic được phân phối cho toàn bộ tiến trình
    - CPU ảo tốc độ << CPU thực

# 1.1.3. CPU chuyển giữa các tiến trình





## 1.1.4. Đặc điểm tiến trình(1)

---

- **I/O-bound process – tiến trình hướng I/O:**
  - Sử dụng nhiều thời gian thực hiện vào/ra hơn việc tính toán
  - Chiếm dụng CPU ngắn
  - Cần chuyển ngữ cảnh thường xuyên khi bắt đầu và kết thúc I/O.
- **CPU-bound process – tiến trình hướng xử lý:**
  - Sử dụng nhiều thời gian cho việc tính toán hơn việc I/O
  - Chiếm dụng CPU dài.
  - Cũng cần chuyển ngữ cảnh thường xuyên để tránh tr.hợp một tiến trình ngăn chặn các tiến trình khác sử dụng CPU.
- **Tiến trình tương tác hay xử lý theo lô**
  - Các tiến trình chiếm dụng CPU trong khoảng thời gian như nhau gọi là **lượng tử thời gian(quantum)**
- **Độ ưu tiên của tiến trình**
  - Các tiến trình có thể được phân cấp theo một số tiêu chuẩn đánh



## 1.1.4. Đặc điểm tiến trình(2)

---

- **Thời gian đã sử dụng CPU của tiến trình**
  - Cần biết thời gian đã sử dụng CPU của tiến trình để tiến hành điều phối(lập lịch)
- **Thời gian còn lại tiến trình cần để hoàn tất**
  - Giảm thiểu thời gian chờ đợi trung bình của các tiến trình bằng cách cho các tiến trình cần ít thời gian nhất để hoàn tất được thực hiện trước

## 1.2. Lập lịch(Điều phối) tiến trình (process scheduling)(1)

---

- Mục tiêu của multiprogramming là có nhiều tiến trình cùng chạy tại mọi thời điểm để tối đa hóa sử dụng CPU.
- Mục tiêu của time-sharing là chuyển CPU giữa các tiến trình càng thường xuyên càng tốt để người sử dụng có thể tương tác với mỗi chương trình khi nó đang chạy.
- Máy tính đơn CPU, tại một thời điểm chỉ có thể chạy 1 tiến trình.
- Nếu có nhiều tiến trình tồn tại, chúng phải đợi đến khi CPU rỗi và được phân phối lại.

## 1.2. Lập lịch tiến trình (process scheduling)(2)

---

- Nguyên tắc chung:
  - Chọn 1 tiến trình trong hàng đợi, ở trạng thái *ready* có độ ưu tiên cao nhất
  - Các yếu tố liên quan đến độ ưu tiên:
    - Thời điểm tạo tiến trình
    - Thời gian phục vụ
    - Thời gian đã dành để phục vụ
    - Thời gian trung bình tiến trình chưa được phục vụ
  - Tiêu chuẩn để chọn một phương pháp điều phối CPU là cần xem xét thời gian chờ đợi xử lý

# 1.2.1. Các danh sách lập lịch tiến trình

- HĐH sử dụng 2 loại danh sách để điều phối các tiến trình:
  - Danh sách sẵn sàng( **ready list**)
  - Danh sách đợi( **waiting list**)
- Khi một tiến trình bắt đầu đi vào hệ thống(tạo lập tiến trình) thuộc vào danh sách tác vụ(**Job list**: tập hợp mọi tiến trình trong hệ thống)
- **Ready list** – tập hợp tất cả các tiến trình *cur trú trong bộ nhớ chính*, ở *trạng thái sẵn sàng* và chờ được thực hiện.
  - HĐH chỉ sử dụng **một ready list** cho toàn bộ hệ thống
- Tiến trình đang thực hiện có thể chuyển sang **trạng thái chờ** khi có yêu cầu tạm dừng, hoặc chờ sự kiện vào ra, hoặc yêu cầu tài nguyên chưa thỏa mãn. Khi đó tiến trình chuyển sang **danh sách chờ**
  - **Device list**: mỗi tài nguyên(thiết bị) có một danh sách chờ riêng bao gồm các tiến trình đang chờ cấp phát tài nguyên đó

## 1.2.2. Các trình lập lịch - Schedulers

- Long-term scheduler (trình lập lịch dài kỳ) còn được gọi là job scheduler (lập lịch công việc, lập lịch tác vụ).
  - Lựa chọn *những* tiến trình nào nên được đưa từ ổ đĩa vào trong **ready list**.
  - Được sử dụng đến rất không thường xuyên (seconds, minutes)
  - May be slow.
  - Kiểm soát *mức đa chương trình* (*degree of multiprogramming*)
- Short-term scheduler (trình lập lịch ngắn kỳ) còn được gọi là CPU scheduler (lập lịch CPU-điều phối CPU).
  - Lựa chọn tiến trình nào nên được thực hiện kế tiếp và phân phối CPU cho nó.
  - Được sử dụng đến rất thường xuyên (milliseconds)
  - Must be fast.





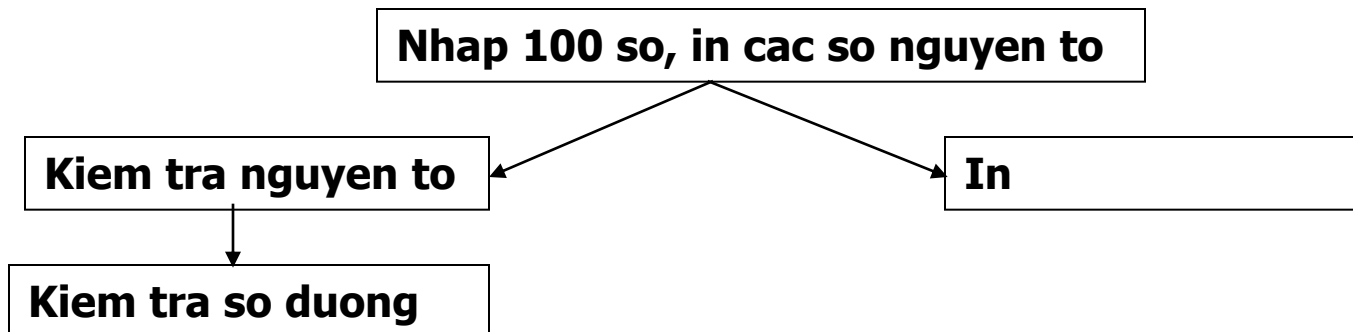
## 1.3. Các hoạt động trên tiến trình

---

- HĐH cấp các thao tác chủ yếu sau đây trên một tiến trình
  - Tạo lập tiến trình (create or new)
  - Kết thúc tiến trình (destroy or terminal)
  - Tạm dừng tiến trình (suspend)
  - Tái kích hoạt tiến trình (resume)
  - Thay đổi độ ưu tiên tiến trình

# 1.3.1. Sự tạo tiến trình - Process Creation(1)

- Một tiến trình có thể tạo lập ra nhiều tiến trình mới bằng cách sử dụng lời gọi hệ thống tương ứng.
  - Tiến trình sử dụng lời gọi hệ thống để tạo lập là **Tiến trình cha**
  - Tiến trình được tạo gọi là tiến trình con
- Một tiến trình con có thể tạo ra các tiến trình mới...quá trình này tạo ra cây tiến trình
- Tạo tiến trình là một công việc "nặng nhọc" vì phải phân phối bộ nhớ và tài nguyên.



# 1.3.1. Sự tạo tiến trình - Process Creation(2)



---

- Các công việc HĐH cần thực hiện khi tạo lập tiến trình:
  - Định danh cho tiến trình mới phát sinh
  - Đưa tiến trình vào danh sách quản lý của hệ thống
  - Xác định độ ưu tiên cho tiến trình
  - Tạo PCB cho tiến trình
  - Cấp phát các tài nguyên ban đầu cho tiến trình

# 1.3.1. Sự tạo tiến trình - Process Creation(3)

- Các lựa chọn chia sẻ tài nguyên (resource sharing)
  - Tiến trình cha và con **chia sẻ tất cả tất cả** các tài nguyên.
  - Tiến trình con **chia sẻ tập con** các tài nguyên của tiến trình cha.
  - Tiến trình cha và con **không có sự chia sẻ** tài nguyên.
- Không gian địa chỉ (Address space)
  - Tiến trình con là một bản sao của tiến trình cha(ex: gọi đệ qui)
  - Tiến trình con là một cái mới, chương trình hoàn toàn khác được tải vào trong bộ nhớ(ex: gọi hàm con)
- Sự thực hiện (execution)
  - Tiến trình cha và con **thực hiện đồng thời**.
  - Tiến trình **cha đợi cho đến khi tiến trình con kết thúc**.

## 1.3.2. Sự kết thúc tiến trình- Process Termination

- Tiến trình kết thúc xử lý khi nó hoàn tất chỉ thị cuối cùng và sử dụng một lời gọi hệ thống để yêu cầu HĐH hủy bỏ
- Trong quá trình kết thúc:
  - Dữ liệu ra từ tiến trình con đến tiến trình cha (qua lệnh **wait**).
- Tiến trình cha có thể chấm dứt việc thực hiện tiến trình con (**abort**).
  - Tiến trình con dùng quá tài nguyên được phân phối.
  - Nhiệm vụ mà tiến trình con thực hiện không còn cần thiết.
- Khi tiến trình kết thúc, HĐH thực hiện các công việc:
  - Thu hồi các tài nguyên hệ thống đã cấp phát cho tiến trình
  - Hủy tiến trình khỏi tất cả các danh sách quản lý của hệ thống
  - Hủy bỏ PCB của tiến trình
  - Các tài nguyên được phân phối lại
- Hầu hết các hệ điều hành không cho phép các tiến trình con tiếp tục tồn tại nếu tiến trình cha đã kết thúc



## 1.4. Các tiến trình hợp tác(1)

---

- Tiến trình *độc lập* (*Independent process*):
  - Không thể tác động hay chịu tác động bởi sự thực hiện của tiến trình khác.
- Tiến trình *hợp tác* (*Cooperating process*):
  - Có thể tác động hoặc chịu tác động bởi sự thực hiện của tiến trình khác.
  - Ví dụ: tiến trình này chia sẻ dữ liệu với tiến trình khác.



## 1.4. Các tiến trình hợp tác(2)

---

- Các lợi điểm của tiến trình hợp tác
  - Chia sẻ thông tin - Information sharing
  - Tăng tốc độ tính toán - Computation speed-up
  - Mô-đun hóa - Modularity
  - Sự tiện lợi - Convenience (vd người sử dụng cùng thực hiện soạn thảo, in ấn, biên dịch song song)
- Mô hình các tiến trình hợp tác: tiến trình sản xuất (*producer* process) tạo ra các thông tin để tiến trình tiêu thụ (*consumer* process) sử dụng.

# 1.5. Liên lạc( giao tiếp) tiến trình(1)

- Mỗi tiến trình sở hữu một không gian địa chỉ riêng biệt, nên các tiến trình không thể liên lạc trực tiếp dễ dàng mà phải nhờ vào các cơ chế do HĐH cung cấp
- Hệ điều hành thường phải tìm giải pháp cho các vấn đề chính yếu:
  - **Liên lạc tường minh hay tiềm ẩn**(*explicit naming/implicit naming*): tiến trình có cần phải biết tiến trình nào đang trao đổi hay chia sẻ thông tin với nó
  - **Liên lạc theo chế độ đồng bộ hay không đồng bộ**(*blocking / non-blocking*): khi một tiến trình trao đổi thông tin với một tiến trình khác, các tiến trình có cần phải đợi cho thao tác liên lạc hoàn tất rồi mới tiếp tục các xử lý khác=>đồng bộ
  - **Liên lạc giữa các tiến trình trong hệ thống tập trung và hệ thống phân tán**
- Mỗi HĐH thường đưa ra nhiều cơ chế liên lạc khác nhau, mỗi cơ chế có những đặc tính riêng, và thích hợp trong một hoàn cảnh chuyên biệt



## 1.5. Liên lạc( giao tiếp) tiến trình(2)

---

- Các cơ chế liên lạc:
  - Tín hiệu (Signal)
  - Pipe( đường ống)
  - Vùng nhớ chia sẻ
  - Trao đổi thông điệp (Message)
  - Sockets



## 1.5.1. Signal(1)

---

- Tín hiệu là một cơ chế phần mềm tương tự như các ngắt cứng tác động đến các tiến trình
- Một tín hiệu được sử dụng để thông báo cho tiến trình về một sự kiện nào đó xảy ra.
- Có nhiều tín hiệu được định nghĩa, mỗi một tín hiệu có một ý nghĩa tương ứng với một sự kiện đặc trưng
- Ex:
  - Tín hiệu **End Task** trong windows
  - **Close all** on the taskbar of windows



## 1.5.1. Signal(2)

---

- Mỗi tiến trình sở hữu một bảng biểu diễn các tín hiệu khác nhau. Với mỗi tín hiệu sẽ có tương ứng một trình xử lý tín hiệu (*signal handler*) qui định các xử lý của tiến trình khi nhận được tín hiệu tương ứng.
- Các tín hiệu được gửi đi bởi:
  - Phần cứng (ví dụ lỗi do các phép tính số học)
  - HĐH gửi đến một tiến trình (ví dụ lưu ý tiến trình khi có một thiết bị I/O tự do).
  - Một tiến trình gửi đến một tiến trình khác (ví dụ tiến trình cha yêu cầu một tiến trình con kết thúc)
  - Người dùng ( ví dụ nhấn phím Ctl-F4 để ngắt xử lý của tiến trình)
- Khi một tiến trình nhận một tín hiệu, nó có thể xử sự theo một trong các cách sau:
  - Bỏ qua tín hiệu; hoặc Xử lý tín hiệu theo kiểu mặc định; hoặc Tiếp nhận tín hiệu và xử lý theo cách đặc biệt của tiến trình.



## 1.5.1. Signal(3)

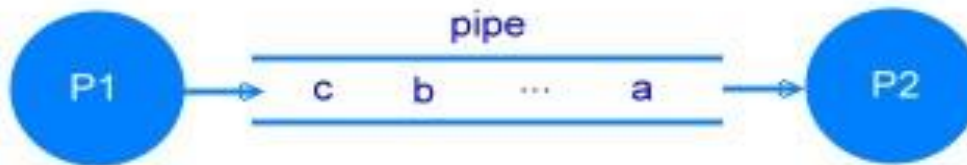
---

- Nhận xét:

- Liên lạc bằng tín hiệu mang tính chất *không đồng bộ*, nghĩa là một tiến trình nhận tín hiệu không thể xác định trước thời điểm nhận tín hiệu.
- Các tiến trình không thể kiểm tra được sự kiện tương ứng với tín hiệu có thật sự xảy ra?
- Các tiến trình chỉ có thể thông báo cho nhau về một biến cố nào đó
- Không trao đổi dữ liệu theo cơ chế này được

## 1.5.2. Pipe(1)

- pipe là một kênh liên lạc **trực tiếp một chiều** giữa hai tiến trình : dữ liệu xuất của tiến trình này được chuyển đến làm dữ liệu nhập cho tiến trình kia dưới dạng một dòng các byte
- Khi một pipe được thiết lập giữa hai tiến trình
  - Một tiến trình sẽ ghi dữ liệu vào pipe
  - Tiến trình kia sẽ đọc dữ liệu từ pipe.
- Thứ tự dữ liệu truyền qua pipe được bảo toàn theo nguyên tắc FIFO.
- Một pipe có kích thước giới hạn (thường là 4096 ký tự)





## 1.5.2. Pipe(2)

---

- Một tiến trình **chỉ có thể sử dụng một pipe** do nó tạo ra hay **kế thừa từ tiến trình cha**.
- Hệ điều hành cung cấp các lời gọi hệ thống **read/write** cho các tiến trình thực hiện thao tác đọc/ghi dữ liệu trong pipe.
- Hệ điều hành cũng chịu trách nhiệm đồng bộ hóa việc truy xuất pipe trong các tình huống:
  - Tiến trình đọc pipe sẽ bị khóa nếu pipe trống, nó sẽ phải đợi đến khi pipe có dữ liệu để truy xuất.
  - Tiến trình ghi pipe sẽ bị khóa nếu pipe đầy, nó sẽ phải đợi đến khi pipe có chỗ trống để chứa dữ liệu.



## 1.5.2. Pipe(3)

---

- Liên lạc bằng pipe là một cơ chế liên lạc *một chiều* (*unidirectional*):
  - Một tiến trình kết nối với một pipe chỉ có thể thực hiện một trong hai thao tác đọc hoặc ghi
  - Một số hệ điều hành cho phép thiết lập **hai pipe** giữa một cặp tiến trình để tạo liên lạc hai chiều.
    - Vẫn có nguy cơ xảy ra tình trạng **khóa chết** (deadlock) khi cả hai pipe nối kết hai tiến trình đều đầy(hoặc đều trống) và cả hai tiến trình đều muốn ghi (hay đọc) dữ liệu vào pipe(mỗi tiến trình ghi dữ liệu vào một pipe), chúng sẽ cùng bị khóa và chờ lẫn nhau mãi!
- Cơ chế này cho phép truyền dữ liệu với cách thức không cấu trúc.
- Ngoài ra, một giới hạn của hình thức liên lạc này là chỉ cho phép kết nối hai tiến trình có quan hệ cha-con, và trên cùng một máy tính



## 1.5.3. Vùng nhớ chia sẻ(1)

---

- *Ý tưởng*: cho nhiều tiến trình cùng truy xuất đến một vùng nhớ chung gọi là *vùng nhớ chia sẻ (shared memory)*.
- *Không có bất kỳ hành vi truyền dữ liệu nào cần phải thực hiện ở đây, dữ liệu chỉ đơn giản được đặt vào một vùng nhớ mà nhiều tiến trình có thể cùng truy cập được.*
- Một vùng nhớ chia sẻ tồn tại độc lập với các tiến trình
  - Khi một tiến trình muốn truy xuất đến vùng nhớ này, tiến trình phải kết gắn vùng nhớ chung đó vào không gian địa chỉ riêng của từng tiến trình
  - Thao tác trên đó như một vùng nhớ riêng của mình.





## 1.5.3. Vùng nhớ chia sẻ(2)

---

- Nhận xét:

- Là phương pháp nhanh nhất để trao đổi dữ liệu giữa các tiến trình
- khó khăn trong việc bảo đảm sự toàn vẹn dữ liệu:
  - Không biết dữ liệu đang truy cập là mới nhất?
  - Làm sao ngăn cản nhiều tiến trình cùng truy nhập bộ nhớ chung
- Vùng nhớ chia sẻ cần được bảo vệ bằng những cơ chế đồng bộ hóa thích hợp
- Cơ chế này không thể áp dụng trong các hệ phân tán



## 1.5.4. Trao đổi thông điệp

- Là cơ chế để các tiến trình giao tiếp và để đồng bộ các hành động của chúng không phải chia sẻ không gian địa chỉ chung mà thông qua trao đổi các thông điệp
- HĐH cung cấp các hàm IPC chuẩn (InterProcess communication), cơ bản là hai hàm:
  - **send** (*message*— kích thước của message cố định hoặc biến đổi)
  - **receive** (*message*)
- Nếu các tiến trình  $P$  và  $Q$  muốn giao tiếp, chúng cần phải:
  - Thiết lập một *liên kết giao tiếp* (*communication link*) giữa chúng.
  - Trao đổi các message qua các hoạt động **send/receive**.
  - Kết thúc trao đổi liên kết giao tiếp bị hủy bỏ
- Có nhiều cách thức để thực hiện sự liên kết giữa hai tiến trình và cài đặt các theo tác send /receive tương ứng : liên lạc trực tiếp hay gián tiếp, liên lạc đồng bộ hoặc không đồng bộ
- Nhận xét:
  - Đơn vị thông tin trao đổi là thông điệp, các thông điệp có thể có cấu trúc

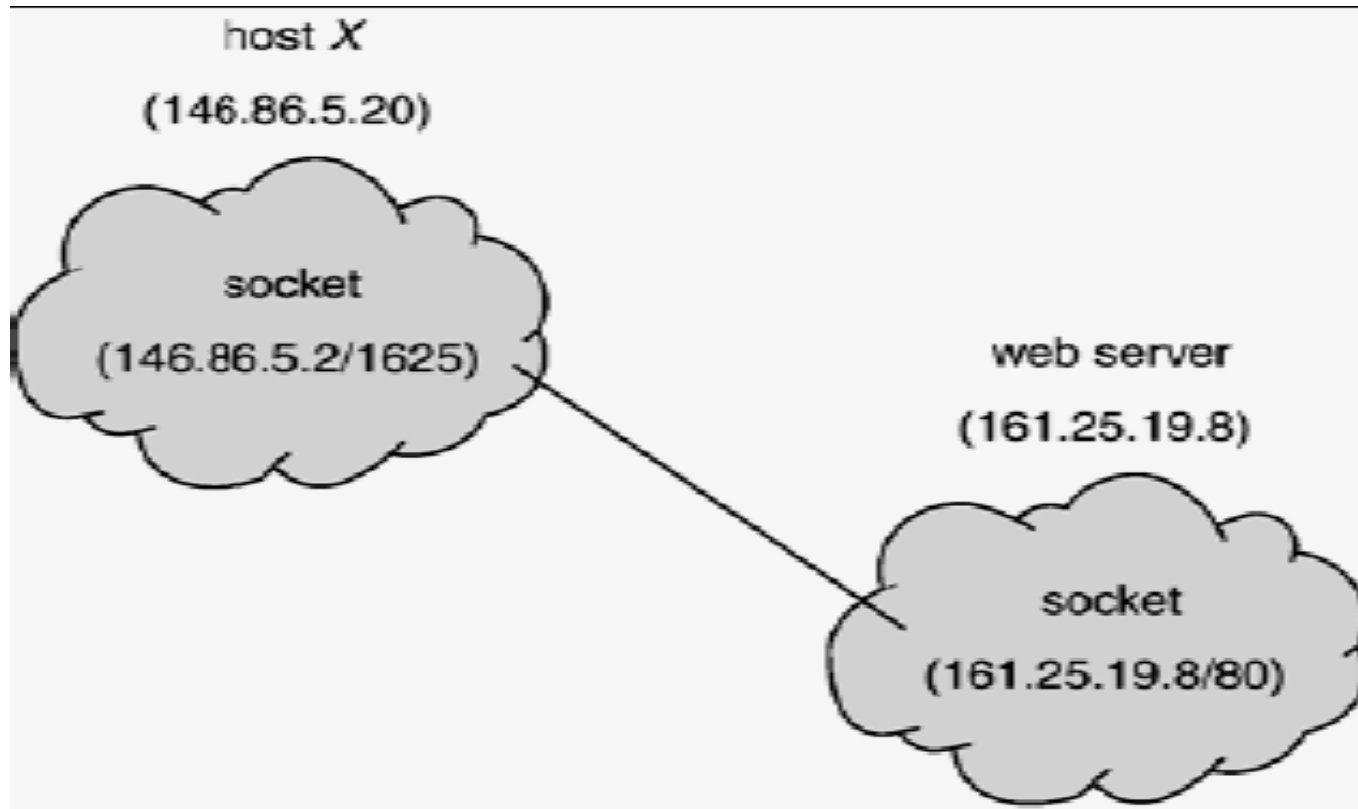


## 1.5.5. Sockets(1)

---

- Một socket được định nghĩa là một *điểm cuối giao tiếp* (*endpoint for communication*).
- Được xác định bởi địa chỉ IP và số hiệu cổng.
- Socket **161.25.19.8:1625** có nghĩa là port **1625** trên host **161.25.19.8**
- Sự giao tiếp diễn ra giữa một cặp socket:
  - Server socket được chọn
  - Client socket được gán tự động

## 1.5.5. Sockets(2)



# 1.6. Phân phối tài nguyên cho tiến trình(1)

- Hệ điều hành quản lý nhiều loại tài nguyên khác nhau (CPU, bộ nhớ chính, các thiết bị ngoại vi ...)
  - Mỗi loại cần có một cơ chế cấp phát và các chiến lược cấp phát hiệu quả.
- Mỗi tài nguyên được biểu diễn thông qua một cấu trúc dữ liệu, khác nhau về chi tiết cho từng loại tài nguyên, nhưng cơ bản chứa đựng các thông tin sau:
  - **Định danh tài nguyên:** để phân biệt các tài nguyên
  - **Trạng thái tài nguyên :** đây là các thông tin mô tả chi tiết trạng thái tài nguyên:
    - Phần nào của tài nguyên đã cấp phát cho tiến trình
    - Phần nào còn có thể sử dụng
  - **Hàng đợi trên một tài nguyên :** danh sách chứa các tiến trình đang chờ được cấp phát tài nguyên tương ứng.
  - **Bộ cấp phát:** là đoạn code đảm nhiệm việc cấp phát một tài nguyên đặc thù.

# 1.6. Phân phối tài nguyên cho tiến trình(2)

Khối quản lý tài nguyên



- Mục tiêu của kỹ thuật cấp phát :
  - Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được.
  - Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được.
  - Tối ưu hóa sự sử dụng tài nguyên



## 2. Luồng - Threads

---

- Nội dung
  - Mô tả luồng
  - Các mô hình đa luồng



## 2.1. Mô tả luồng(1)

---

- Vấn đề:
  - Mỗi tiến trình có một không gian địa chỉ và chỉ có **một dòng xử lý**
  - Để tăng tốc độ và sử dụng CPU hiệu quả hơn → cần **nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ**, và các dòng xử lý này hoạt động song song tương tự như các tiến trình phân biệt (ngoại trừ việc chia sẻ không gian địa chỉ)
  - Mỗi dòng xử lý được gọi là một luồng(thread) hay một tiểu trình





## 2.1. Mô tả luồng(2)

---

- **Khái niệm:** Một luồng là một dòng xử lý cơ bản trong hệ thống . Mỗi luồng xử lý tuần tự đoạn code của nó, sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng
- Phân biệt luồng và tiến trình:
  - Luồng được coi là mức thấp hơn của tiến trình, mỗi tiến trình có thể gồm nhiều luồng
  - Hoạt động của các luồng giống như tiến trình nhưng các luồng cùng **chia sẻ không gian địa chỉ chung**, các tiến trình thì hoàn toàn độc lập
- Khi máy tính có nhiều CPU, mỗi CPU có thể thực hiện các công việc khác nhau hoặc các luồng khác nhau cho cùng một công việc.
- Khi hệ thống chỉ có một CPU, mỗi luồng được thực hiện luân phiên nhau. Không có luồng nào chiếm ưu thế trong CPU.
- Luồng là hữu ích vì chúng loại trừ được sự cần thiết phải để cho hệ điều hành liên tục tải thông tin vào/ra bộ nhớ.

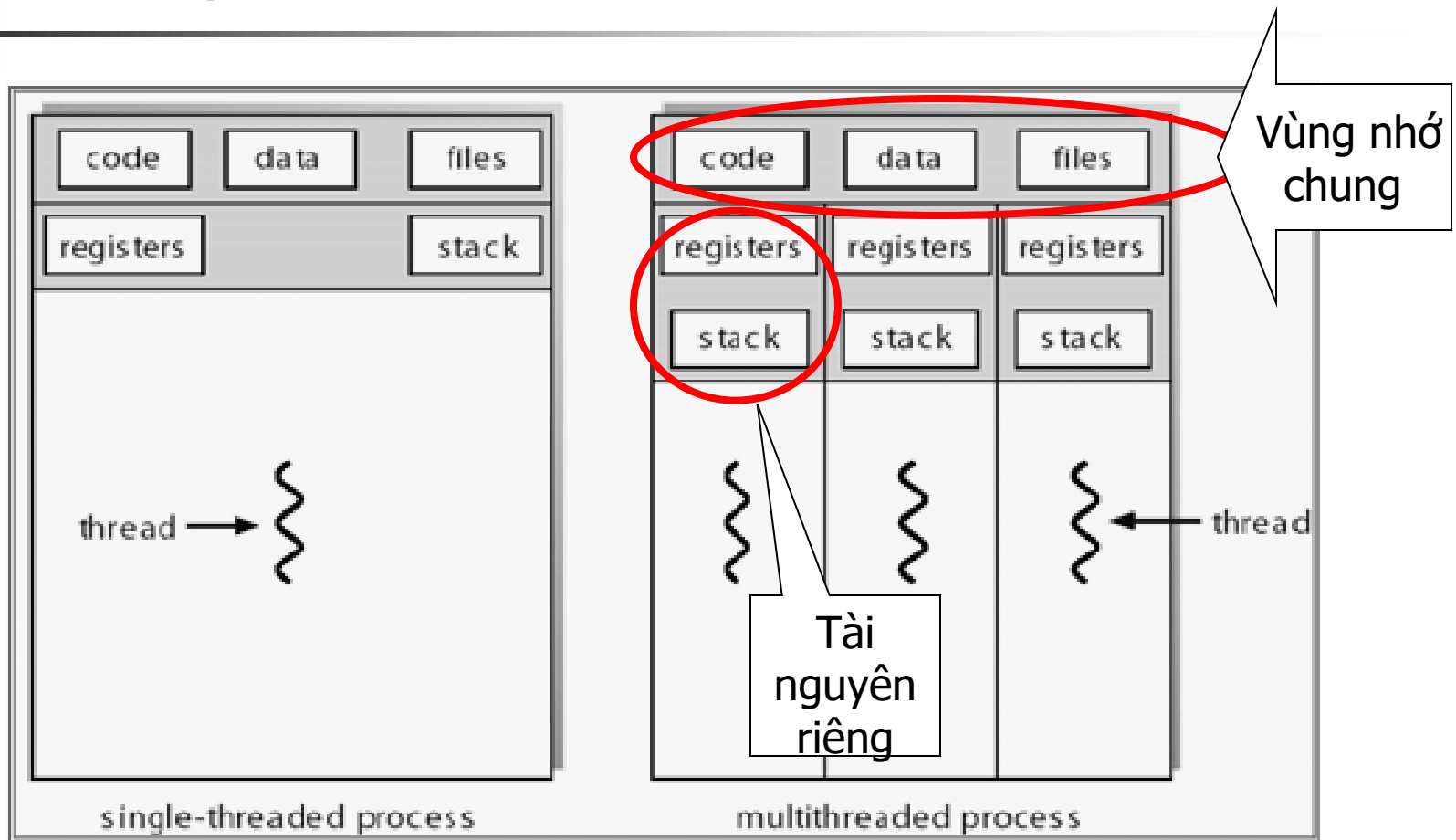


## 2.1. Mô tả luồng(3)

---

- Mỗi luồng có thể tương tác với một phần riêng của hệ thống, như đĩa, I/O trên mạng, hoặc người dùng.
- Các luồng được lập lịch để thực hiện vì một số luồng có thể chờ một biến cố nào đó xảy ra hoặc chờ kết thúc một công việc nào đó từ luồng khác.
- Luồng bao gồm:
  - Mã luồng (thread ID)
  - Bộ đếm chương trình (PC)
  - Tập thanh ghi (register set)
  - Stack
- Các luồng trong một tiến trình chia sẻ với nhau đoạn mã (code), đoạn dữ liệu (data) và các tài nguyên hệ thống khác như các tệp mở, các tín hiệu.

## 2.1.1. Các tiến trình đơn và đa luồng





## 2.1.2. Sự thúc đẩy

---

- Tạo tiến trình là một công việc "nặng nhọc"
- Nhiều phần mềm chạy trên các PC hiện nay là đa luồng (multithreaded). Một ứng dụng thường được thực hiện như một tiến trình riêng với một **vài luồng điều khiển**.
- Ex1: Trình soạn thảo văn bản
  - 1 luồng hiển thị ảnh, chữ
  - 1 luồng đọc phím nhấn bởi người sử dụng
  - 1 luồng thực hiện việc kiểm tra chính tả và ngữ pháp
- Ex2: web-server tạo 1 luồng nghe các yêu cầu từ client.
  - Khi có yêu cầu, thay vì tạo 1 tiến trình khác, nó sẽ tạo một luồng khác để phục vụ yêu cầu.



## 2.1.3. Lợi ích của tiến trình đa luồng

---

- **Đáp ứng nhanh (đáp ứng với người dùng):**
  - Cho phép chương trình vẫn thực hiện ngay khi một phần đang chờ đợi (block) hoặc đang thực hiện tính toán tăng cường.
  - Ví dụ trình duyệt Web đa luồng
    - Một luồng tương tác người dùng
    - Một luồng thực hiện nhiệm vụ tải dữ liệu
- **Chia sẻ tài nguyên:**
  - Các luồng chia sẻ bộ nhớ và tài nguyên của tiến trình chưa có
    - Tốt cho các thuật toán song song (sử dụng chung các CTDL)
    - Trao đổi giữa các luồng thông qua bộ nhớ phân chia.
  - Cho phép một ứng dụng chứa nhiều luồng hoạt động trong cùng không gian địa chỉ



## 2.1.3. Lợi ích của tiến trình đa luồng

---

- **Kinh tế:**
  - Các thao tác khởi tạo, hủy bỏ và luân chuyển luồng ít tốn kém
- **Thực hiện trong kiến trúc multiprocessor:**
  - Các luồng chạy song song thực sự trên các bộ VXL khác nhau.



## 2.1.4. Kernel Threads

---

- Khái niệm luồng có thể được cài đặt trong kernel của Hệ điều hành, khi đó đơn vị cơ sở sử dụng CPU để xử lý là luồng
- Hệ điều hành sẽ phân phối CPU cho các luồng trong hệ thống
- ⇒ Các luồng này gọi là các **luồng mức nhân(kernel)**
- Kernel thực hiện tạo luồng, lập lịch và quản lý trong không gian kernel. Do đó, tạo và quản lý các kernel thread nói chung chậm hơn các **user thread**.
- Nếu một luồng thực hiện một system call bị khóa, kernel có thể lập lịch một luồng khác để thực hiện.
- Trong môi trường multiprocessor, kernel có thể lập lịch các luồng trên các processor khác nhau.



## 2.1.5. User-level Threads

---

- Được hỗ trợ trên kernel và được thực hiện bởi một **thư viện luồng tại mức người sử dụng** (user level).
- Tất cả sự tạo luồng và lập lịch được thực hiện trong không gian người sử dụng. Do đó, các user-level thread nói chung nhanh để tạo và quản lý.
- Luồng mức người dùng là các xử lý mức trên thuộc **user mode**, các luồng mức này sẽ được ánh xạ vào luồng hoặc tiến trình mức kernel để thực hiện
- **Hạn chế:**
  - Khi kernel là đơn luồng, nếu có 1 user-level thread thực hiện một system call bị khóa, nó sẽ gây cho toàn bộ tiến trình bị khóa, mặc dù các tiến trình khác vẫn có thể chạy trong ứng dụng.





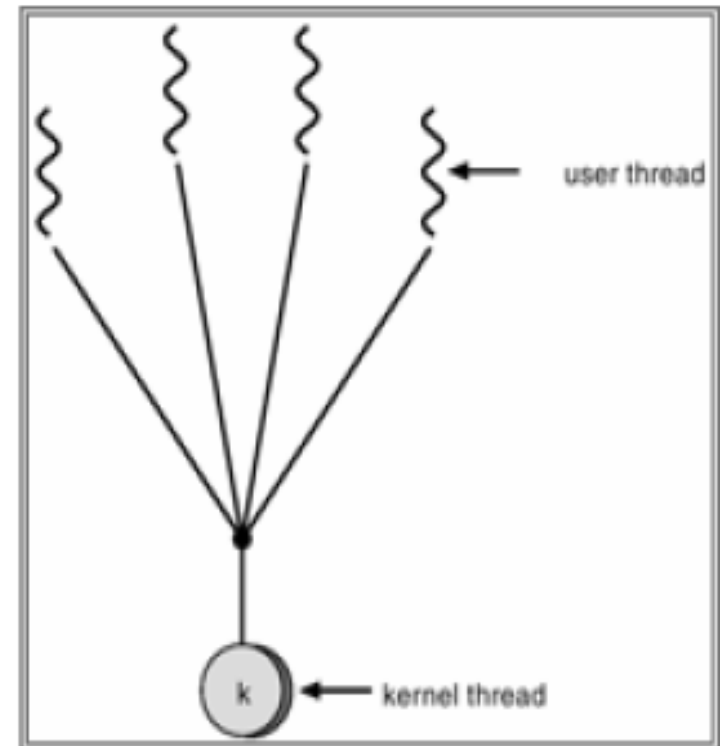
## 2.2. Các mô hình đa luồng

---

- Nhiều HĐH hỗ trợ cả user thread và kernel thread, thể hiện trong 3 mô hình đa luồng phổ biến:
  - Many-to-One
  - One-to-One
  - Many-to-Many

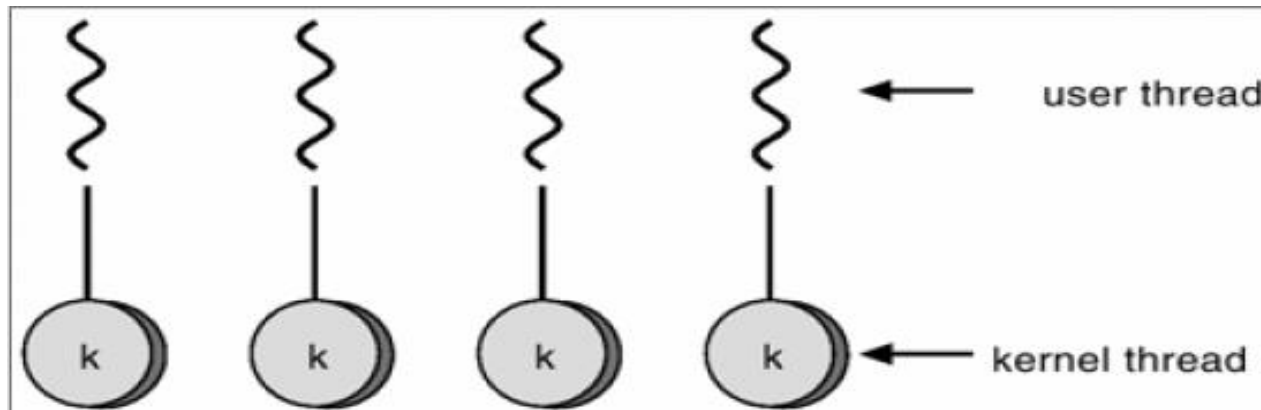
## 2.2.1. Mô hình Many-to-One

- Nhiều user-level thread được ánh xạ vào 1 kernel thread
- Quản lý luồng được thực hiện trong không gian người sử dụng → nhanh nhưng tiến trình dễ bị khóa.
- Các luồng không thể chạy song song trong các hệ thống multiprocessor.



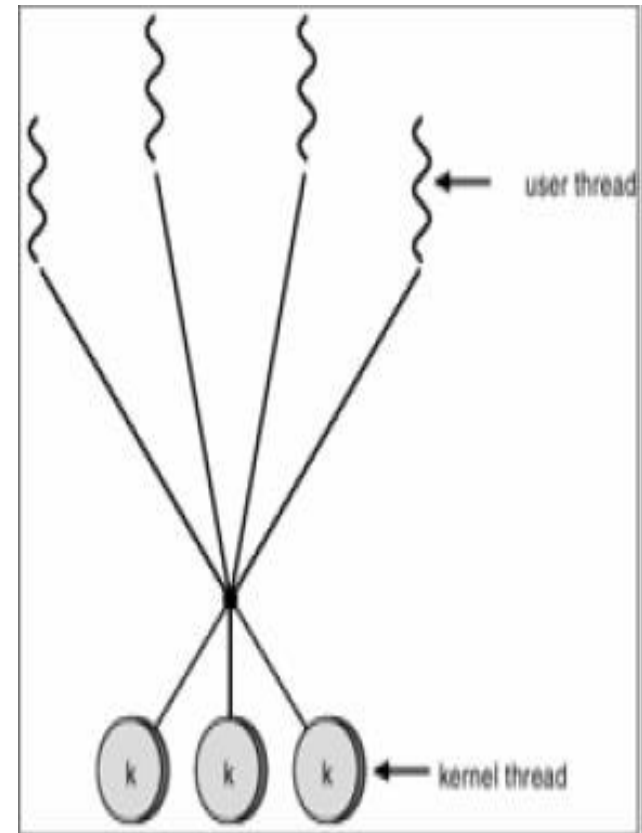
## 2.2.2. Mô hình One-to-One

- Mỗi user-level thread được ánh xạ vào 1 kernel thread
- Cho phép tiến trình khác chạy khi có 1 tiến trình tạo system call bị khóa.
- Cho phép nhiều luồng chạy song song trên multiprocessor.
- Cần giới hạn số luồng được hỗ trợ bởi HĐH
- Vd: Windows NT/2000/XP



## 2.2.3. Mô hình Many-to-Many

- Nhiều user-level thread (n) được ánh xạ vào nhiều kernel thread (m)
- $m \leq n$
- Người phát triển có thể tạo bao nhiêu user-level thread tùy ý, các kernel thread tương ứng có thể chạy song song trên multiprocessor.
  - Khi 1 thread thực hiện 1 system call bị khóa, kernel có thể lập lịch 1 thread khác để thực hiện.
- Vd: UNIX





## Ví dụ: Vector

---

- Tính toán trên vector kích thước lớn

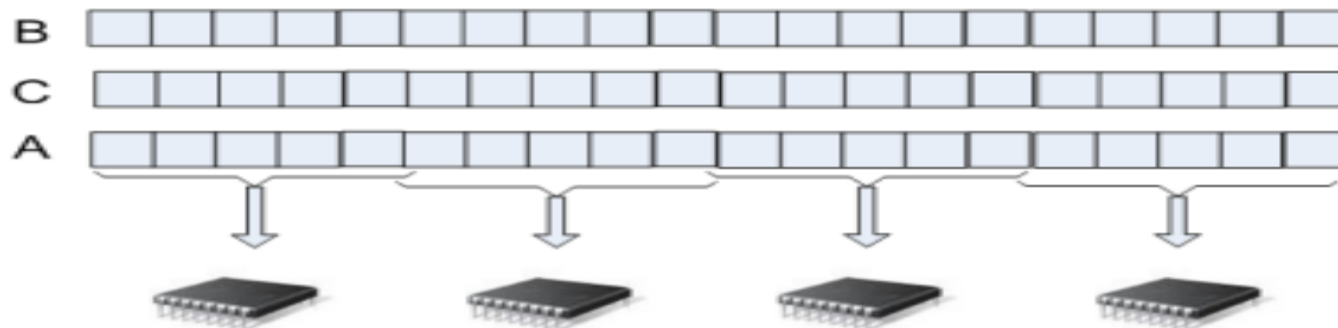
```
For (k=0; k<n; k++){  
    a[k] = b[k] * c[k];  
}
```

# Ví dụ: Vector

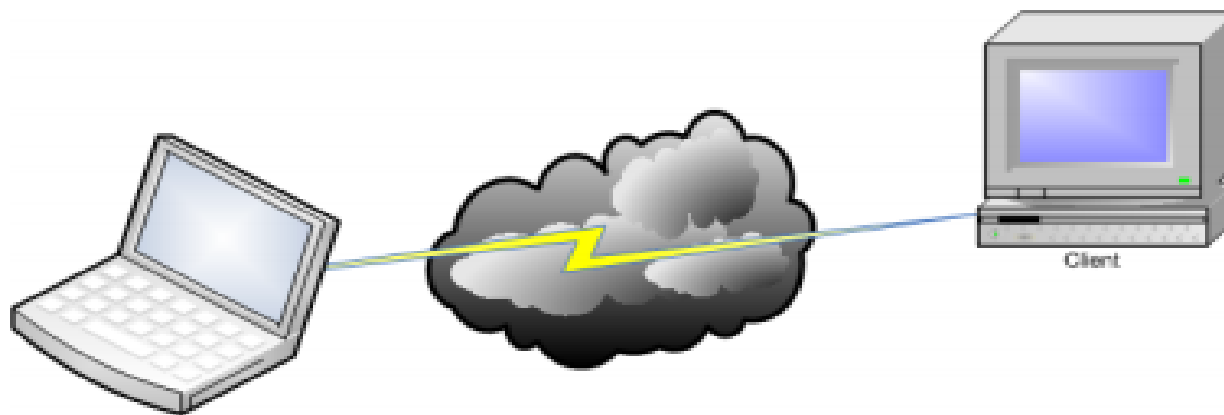
- Tính toán trên vector kích thước lớn

```
For (k=0; k<n; k++){  
    a[k] = b[k] * c[k];  
}
```

- Với hệ thống nhiều vi xử lý



# Ví dụ: Chat



## Process Q

```
while(1){  
    Receive(P,Msg);  
    PrintLine(Msg);  
    ReadLine(Msg);  
    Send(P,Msg);  
}
```

## Vấn đề nhận Msg

- Blocking Receive
- Non-blocking Receive

## Giải quyết

Thực hiện song song  
Receive & Send

## Process P

```
while(1){  
    ReadLine(Msg);  
    Send(Q,Msg);  
    Receive(Q,Msg);  
    PrintLine(Msg);  
}
```



# So sánh tiến trình và luồng

## Tiến trình

- Tiến trình có đoạn mã/dữ liệu/heap & các đoạn khác
- Phải có ít nhất một luồng trong mỗi tiến trình
- Các luồng trong phạm vi một tiến trình chia sẻ mã/dữ liệu/heap, vào/ra nhưng có stack và tập thanh ghi riêng
- Thao tác khởi tạo, luân chuyển tiến trình tốn kém
- Bảo vệ tốt do có không gian địa chỉ riêng
- Khi tiến trình kết thúc, các tài nguyên được đòi lại và các luồng phải kết thúc theo

## Luồng

- Luồng không có đoạn dữ liệu hay heap riêng
- Luồng không đứng riêng mà nằm trong một tiến trình
- Có thể tồn tại nhiều luồng trong một tiến trình. Luồng đầu là luồng chính và sở hữu không gian stack của tiến trình
- Thao tác khởi tạo và luân chuyển luồng không tốn kém
- Không gian địa chỉ chung, cần phải bảo vệ
- Luồng kết thúc, stack của nó được thu hồi



# Cài đặt luồng

## Một số hàm với luồng trong win32

### ■ HANDLE CreateThread(...)

- **LPSECURITY\_ATTRIBUTES** IpThreadAttributes

=> Trỏ tới cấu trúc an ninh: thẻ trả về có thể được kế thừa

- **DWORD** dwStackSize

=> Kích thước ban đầu của Stack cho luồng mới

- **LPTHREAD\_START\_ROUTINE** IpStartAddress

=> Trỏ tới hàm được thực hiện bởi luồng mới

- **LPVOID** IpParameter

=> Trỏ tới các biến được gửi tới luồng mới (tham số của hàm)

- **DWORD** dwCreationFlags

=> Phương pháp tạo luồng

- **CREATE\_SUSPENDED**: luồng ở trạng thái tạm ngừng
- 0: Luồng được thực hiện ngay lập tức

- **LPDWORD** IpThreadId

=> Biến ghi nhận định danh luồng mới

3-May-17

Khoa CNTT - HvKTMM

1. Kết quả: thẻ của luồng mới hoặc giá trị NULL nếu không tạo được



# Ví dụ: Luồng trong C#

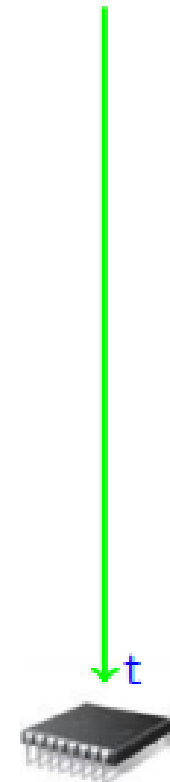
---

# Vấn đề đa luồng

Shared int $y = 1$	
Thread $T_1$	Thread $T_2$
$x \leftarrow y + 1$	$y \leftarrow 2$
	$y \leftarrow y * 2$
$x = ?$	

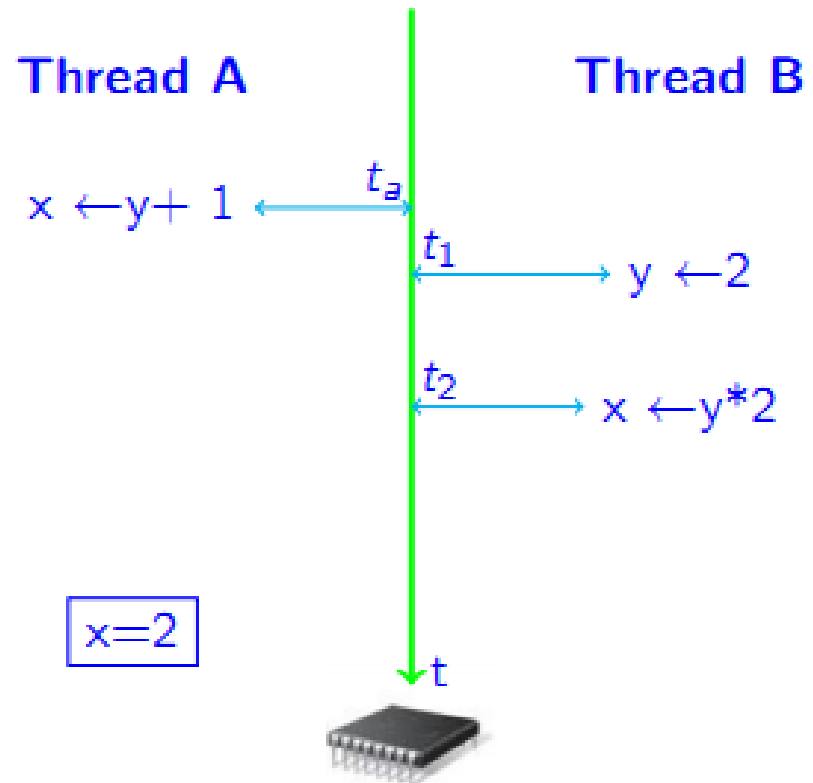
Thread A

Thread B



# Vấn đề đa luồng

Shared int $y = 1$	
Thread $T_1$	Thread $T_2$
$x \leftarrow y + 1$	$y \leftarrow 2$
	$y \leftarrow y * 2$
$x = ?$	

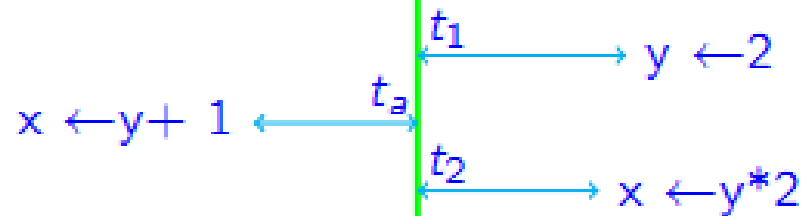


# Vấn đề đa luồng

Shared int $y = 1$	
Thread $T_1$	Thread $T_2$
$x \leftarrow y + 1$	$y \leftarrow 2$
	$y \leftarrow y * 2$
$x = ?$	

Thread A

Thread B



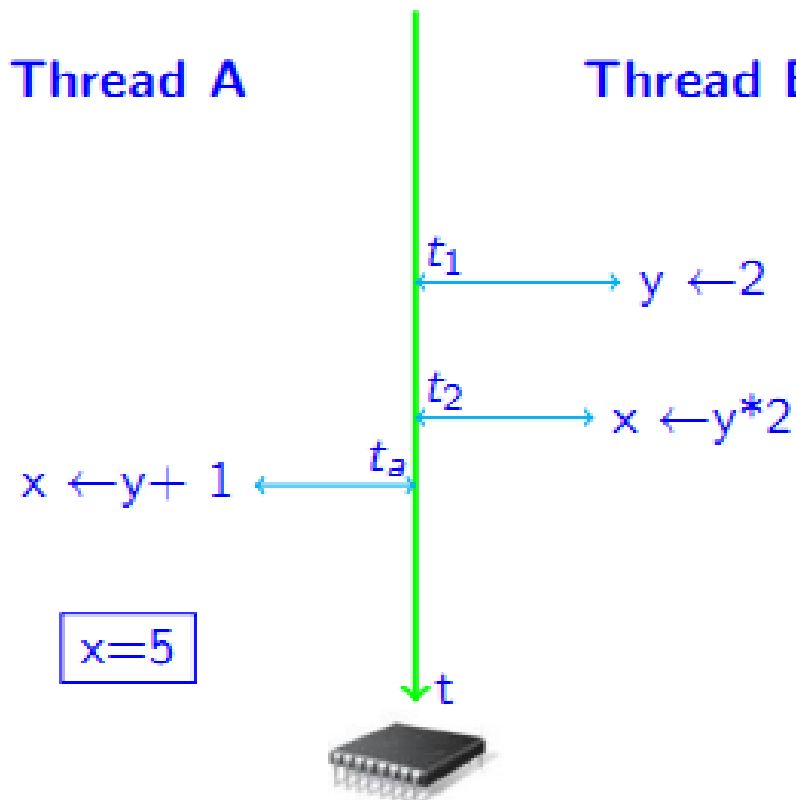
$x=3$

# Vấn đề đa luồng

Shared int $y = 1$	
Thread $T_1$	Thread $T_2$
$x \leftarrow y + 1$	$y \leftarrow 2$
	$y \leftarrow y * 2$
$x = ?$	

Thread A

Thread B



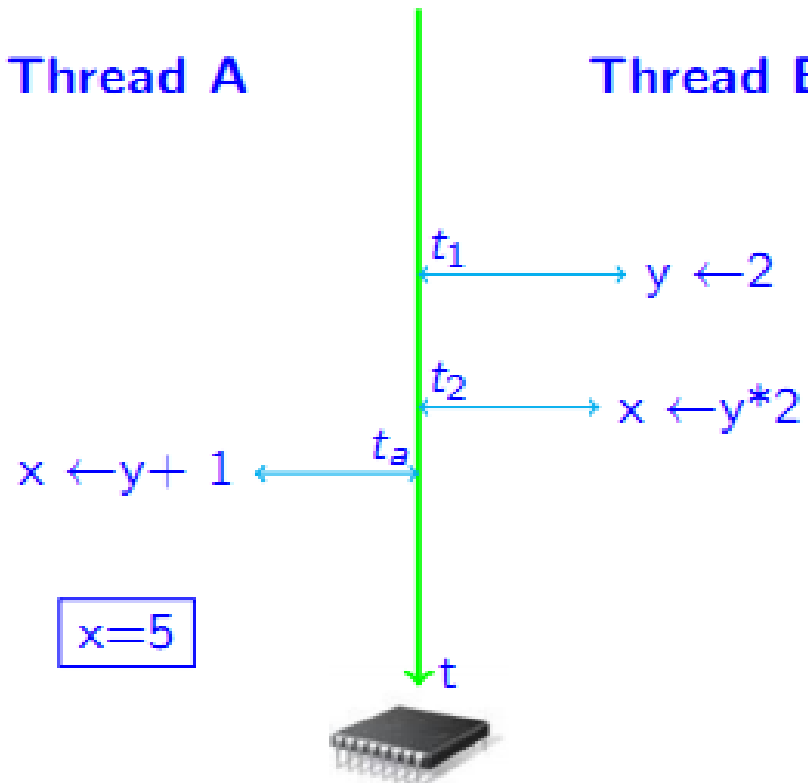
# Vấn đề đa luồng

Shared int $y = 1$	
Thread $T_1$	Thread $T_2$
$x \leftarrow y + 1$	$y \leftarrow 2$
	$y \leftarrow y * 2$
$x = ?$	

Kết quả thực hiện các luồng song song phụ thuộc trật tự truy nhập biến dùng chung giữa chúng

Thread A

Thread B





# Q & A

---

```
int x=0, y =1;
void T1(){
    while(1){
        x=y+1;
        printf("%4d",x);
    }
}
```

```
void T2(){
    while(1){
        y=2;
        y=y*2;
    }
}
```





# Q & A

---

```
int main()
{
    HANDLE h1,h2;
    DWORD ThreadId;
    h1 = CreateThread(NULL,0,T1,NULL,0,&ThreadId);
    h2 = CreateThread(NULL,0,T2,NULL,0,&ThreadId);
    WaitForSingleObject(h1,INFINITE);
    WaitForSingleObject(h1,INFINITE);
    return 0;
}
```

— ☐ ☒