Chương 6

TỔ CHỨC VÀ HOẠT ĐỘNG CỦA HỆ ĐIỀU HÀNH

Nội dung

1. Tiến trình và tiểu trình.
2. Lập lịch tiến trình.
3. Quản lý bộ nhớ.
4. Bộ nhớ ảo.

Tiến trình

- Tiến trình (process):

  Là một chương trình đang thực thi

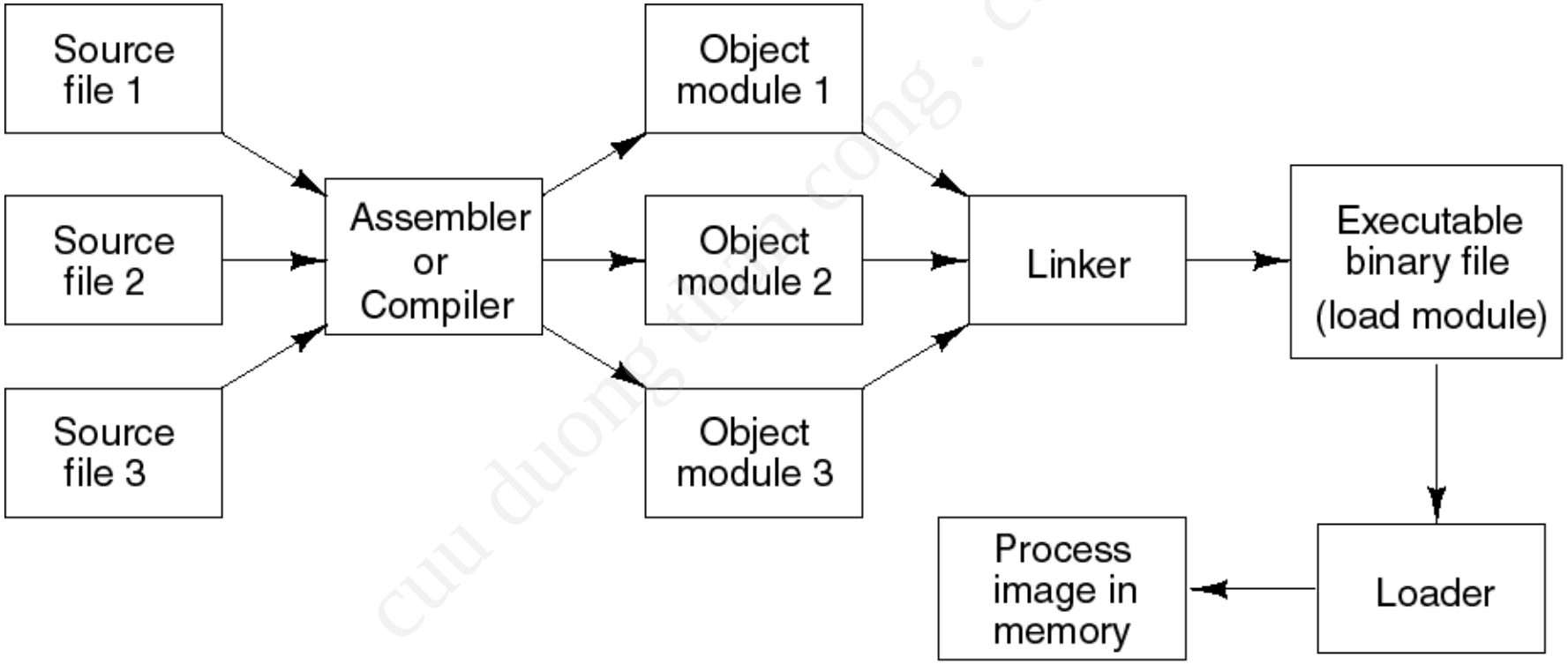
- Một tiến trình bao gồm

  Text section (program code), data section (chứa global variables)

- Hoạt động hiện thời:

  program counter (PC), process status word (PSW), stack pointer (SP), memory management registers

Tiến trình



Các bước nạp chương trình vào bộ nhớ

Tiến trình

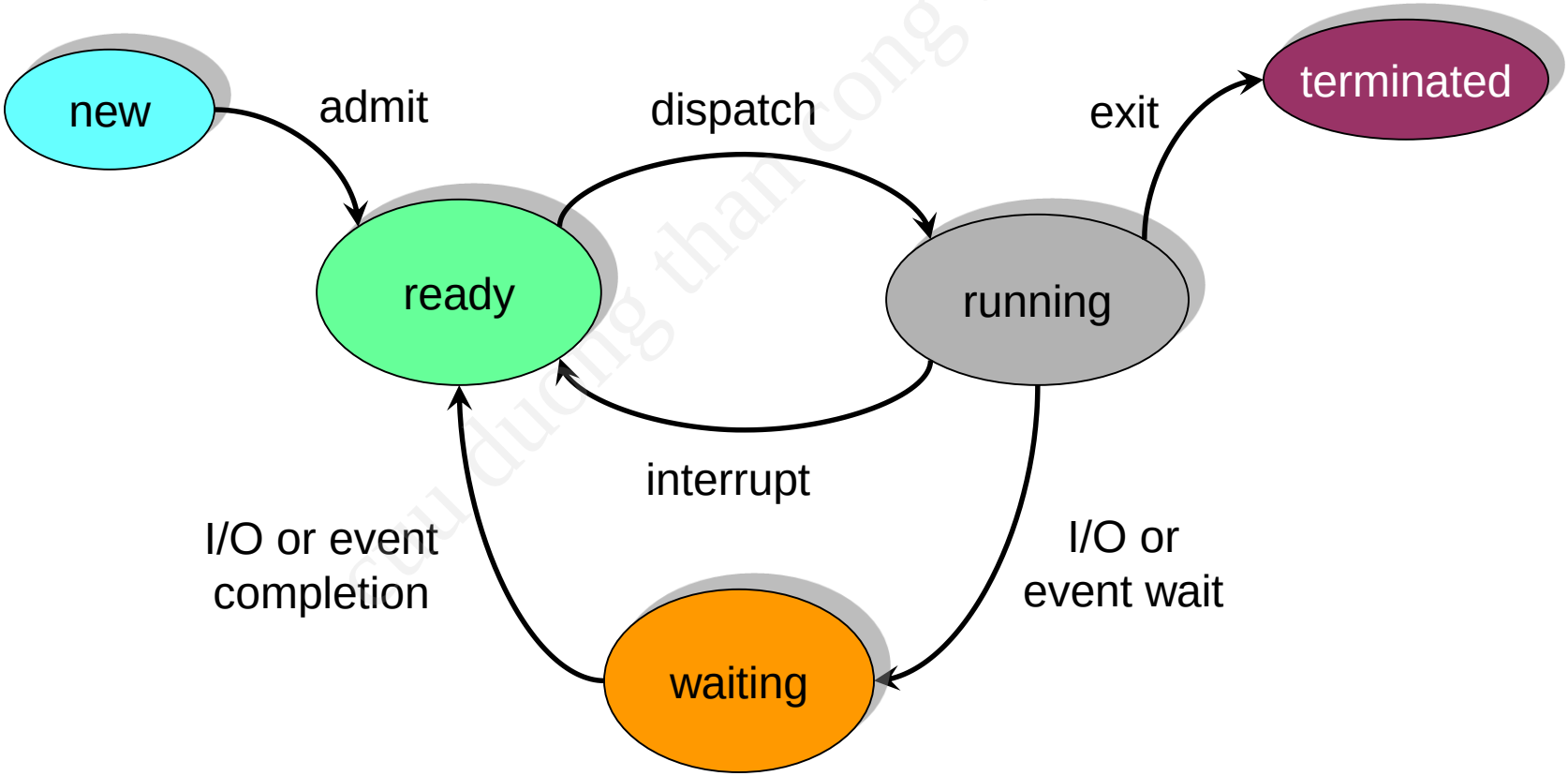
- Các bước hệ điều hành khởi tạo tiến trình:
  - ✓ Cấp phát một định danh duy nhất (process number hay process identifier, pid) cho quá trình
  - ✓ Cấp phát không gian nhớ để nạp quá trình
  - ✓ Khơi tạo khởi dữ liệu Process Control Block (PCB) cho quá trình

    PCB là nơi hệ điều hành lưu các thông tin về quá trình
  - ✓ Thiết lập các mối liên hệ cần thiết (vd: sắp PCB vào hàng đợi định thời,…)

Tiến trình

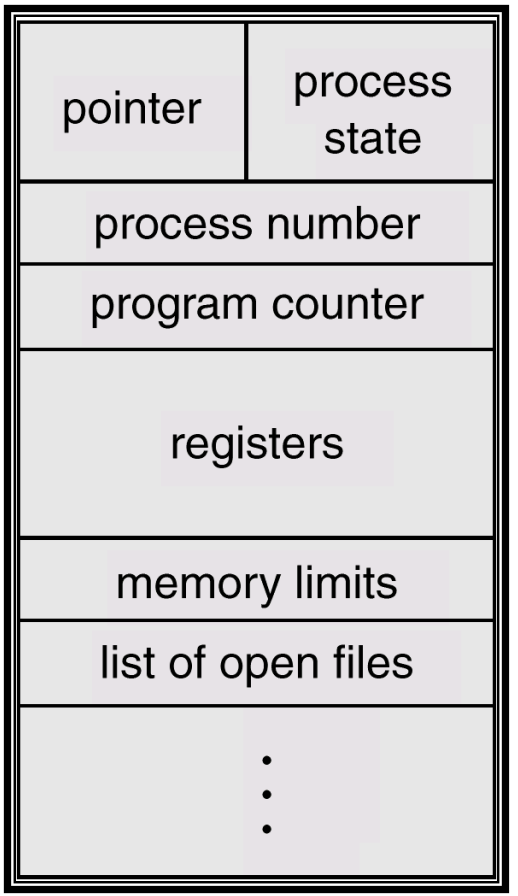
- ✓ new*:* tiến trình vừa được tạo
- ✓ ready*:* tiến trình đã có đủ tài nguyên, chỉ còn cần CPU
- ✓ running*:* các lệnh của tiến trình đang được thực thi
- ✓ waiting*:* hay là blocked, tiến trình đợi I/O hoàn tất, tín hiệu.
- ✓ terminated*:* tiến trình đã kết thúc.

Chuyển đổi giữa các trạng thái của tiến trình



Tiến trình

- Mỗi tiến trình trong hệ thống đều được cấp phát một *Process Control Block* (PCB)
- PCB là một trong các cấu trúc dữ liệu quan trọng nhất của hệ điều hành



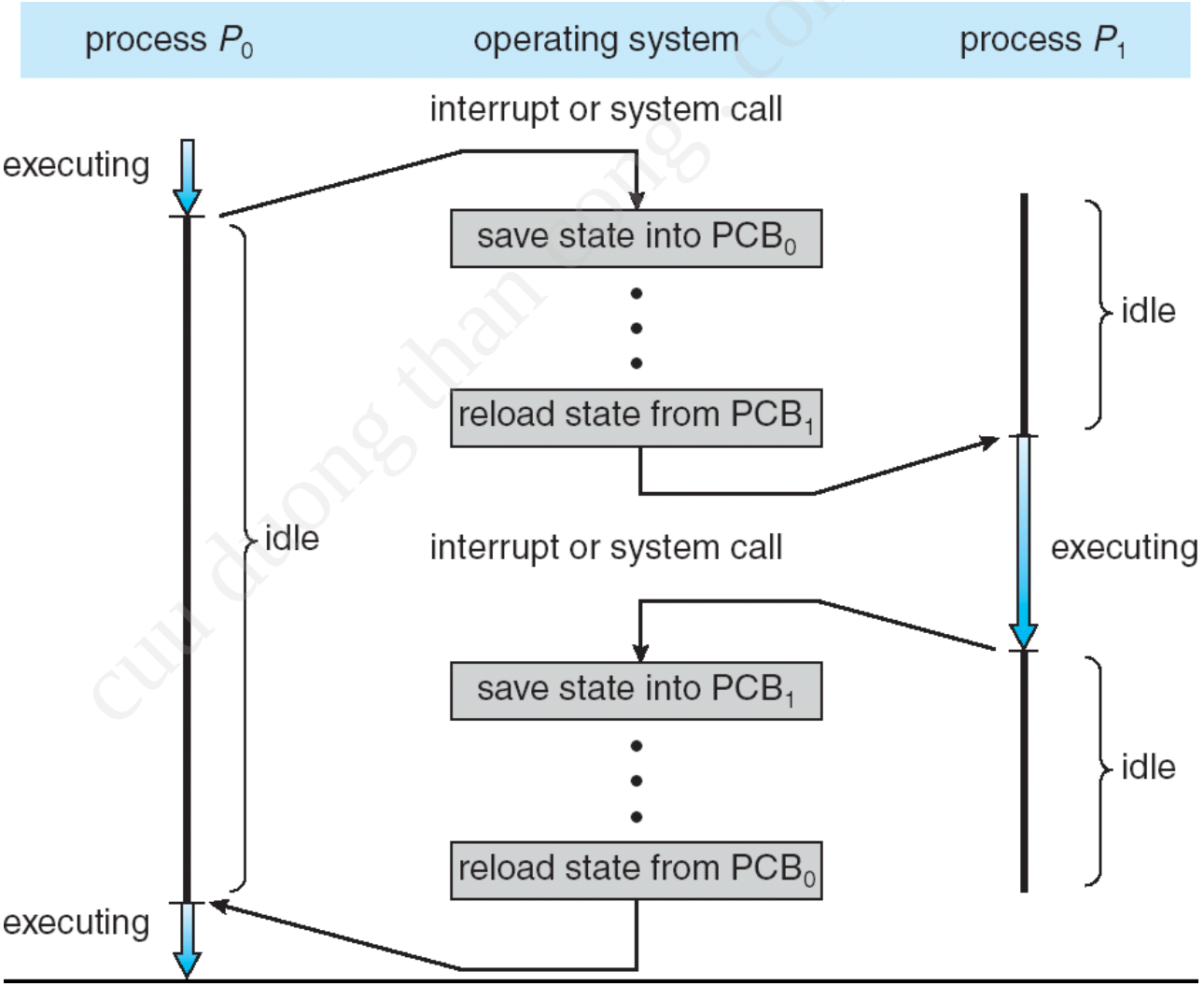
Ví dụ layout của một PCB: (trường pointer dùng để liên kết các PCBs thành một linked list)

Tiến trình

Chuyển ngữ cảnh (Context switch)

- *Ngữ cảnh* (context) của một tiến trình là trạng thái của tiến trình
- Ngữ cảnh của tiến trình được biểu diễn trong PCB của nó
- *Chuyển ngữ cảnh* (context switch) là công việc giao CPU cho tiến trình khác. Khi đó cần:
  - ✓ Lưu ngữ cảnh của tiến trình cũ vào PCB của nó
  - ✓ Nạp ngữ cảnh từ PCB của tiến trình mới để tiến trình mới thực thi

Tiến trình



Tiến trình

Yêu cầu đối với hệ điều hành về quản lý tiến trình

- Hỗ trợ sự thực thi luân phiên giữa nhiều tiến trình
  - ✓ Hiệu suất sử dụng CPU
  - ✓ Thời gian đáp ứng
- Phân phối tài nguyên hệ thống hợp lý
  - ✓ Tránh deadlock, trì hoãn vô hạn định,…
- Cung cấp cơ chế giao tiếp và đồng bộ hoạt động các tiến trình
- Cung cấp cơ chế hỗ trợ user tạo/kết thúc tiến trình

Tiến trình

Yêu cầu đối với hệ điều hành về quản lý tiến trình

- Hỗ trợ sự thực thi luân phiên giữa nhiều tiến trình
  - ✓ Hiệu suất sử dụng CPU
  - ✓ Thời gian đáp ứng
- Phân phối tài nguyên hệ thống hợp lý
  - ✓ Tránh deadlock, trì hoãn vô hạn định,…
- Cung cấp cơ chế giao tiếp và đồng bộ hoạt động các tiến trình
- Cung cấp cơ chế hỗ trợ user tạo/kết thúc tiến trình

Tiểu trình

- Khái niệm tiến trình truyền thống. Tiến trình gồm:
  - Không gian địa chỉ (text section, data section)
  - Một luồng thực thi duy nhất (single thread of execution)
    - ✓ Program counter
    - ✓ Các register
    - ✓ Stack
  - Các tài nguyên khác (các open file, các tiến trình con,…)

Tiểu trình

- Mở rộng khái niệm tiến trình truyền thống bằng cách hiện thực nhiều luồng thực thi trong cùng một môi trường của tiến trình. Tiến trình gồm:
  - Không gian địa chỉ (text section, data section)
  - Một hay nhiều luồng thực thi (thread of execution), mỗi luồng thực thi (thread) có riêng:
    - ✓ Program counter
    - ✓ Các register
    - ✓ Stack
  - Các tài nguyên khác (các open file, các tiến trình con,…)

Tiểu trình

- Các tiểu trình trong cùng một tiến trình chia sẻ code section, data section và tài nguyên khác (các file đang mở,...) của tiến trình.
- Tiến trình *đa luồng* (*multithreaded* process) là tiến trình có nhiều tiểu trình.

Tiểu trình

Ưu điểm của đa tiểu trình

- Tính đáp ứng (responsiveness) cao cho các ứng dụng tương tác multithreaded.
- Chia sẻ tài nguyên (resource sharing).
- Tiết kiệm chi phí hệ thống (lợi ích kinh tế)
  - Chi phí tạo/quản lý tiến trình nhỏ.
  - Chi phí chuyển ngữ cảnh giữa các thread nhỏ.
- Tận dụng kiên trúc đa xử lý (multiprocessor)
  - Mỗi tiến trình chạy trên một processor riêng, do đó tăng mức độ song song của chương trình.

User Thread

- Một thư viện tiểu trình (thread library, run-time system) được hiện thực trong user space để hỗ trợ các tác vụ lên thread
  - Thư viện thread cung cấp các hàm khởi tạo, định thời và quản lý thread như
    - ✓ thread_create,
    - ✓ thread_exit,
    - ✓ thread_wait,
    - ✓ thread_yield.
  - Thư viện tiểu trình dùng *Thread Control Block* (TCB) để lưu trạng thái của user thread (program counter, các register, stack)
- Kernel không biết sự có mặt của user thread

User Thread

- Một thư viện tiểu trình (thread library, run-time system) được hiện thực trong user space để hỗ trợ các tác vụ lên thread
  - Thư viện thread cung cấp các hàm khởi tạo, định thời và quản lý thread như
    - ✓ thread_create,
    - ✓ thread_exit,
    - ✓ thread_wait,
    - ✓ thread_yield.
  - Thư viện tiểu trình dùng *Thread Control Block* (TCB) để lưu trạng thái của user thread (program counter, các register, stack)
- Kernel không biết sự có mặt của user thread

Tiểu trình

Kernel Thread

- Cơ chế multithreading được hệ điều hành trực tiếp hỗ trợ:
  - Kernel quản lý cả process và các thread
  - Việc định thời CPU được kernel thực hiện trên thread

Tiểu trình

Kernel Thread

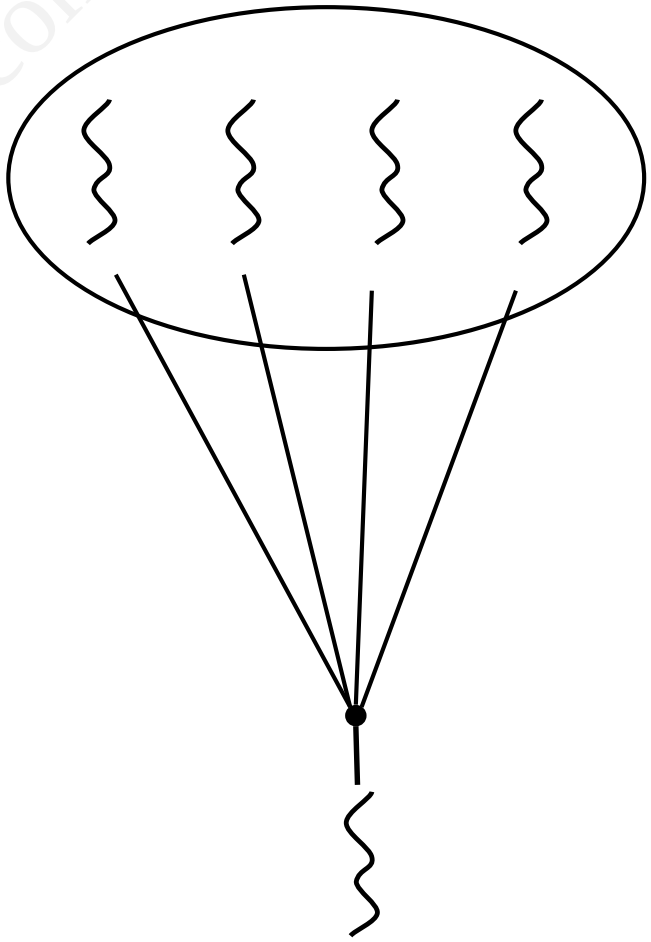
- Cơ chế multithreading được hỗ trợ bởi kernel
  - Khởi tạo và quản lý các thread chậm hơn
  - Tận dụng được lợi thế của kiến trúc multiprocessor
  - Thread bị blocked không kéo theo các thread khác bị blocked.
- Một số hệ thống multithreading (multitasking)
  - Windows 9x/NT/2000/XP/7/10
  - Solaris
  - Linux

Tiểu trình

- Mộ số mô hình hiện thực thread
  - Mô hình *many-to-one*
  - Mô hình *one-to-one*
  - Mô hình *many-to-many*

Mô hình many – to – one

- Nhiều user-level thread "chia sẻ" một kernel thread để thực thi
  - Việc quản lý thread được thực hiện thông qua các hàm của một thread library được gọi ở user level.
  - Blocking problem: Khi một thread trở nên blocked thì kernel thread cũng trở nên blocked, do đó mỗi thread khác của process cũng sẽ trở nên blocked.
- Có thể được hiện thực đối với hầu hết các hệ điều hành.

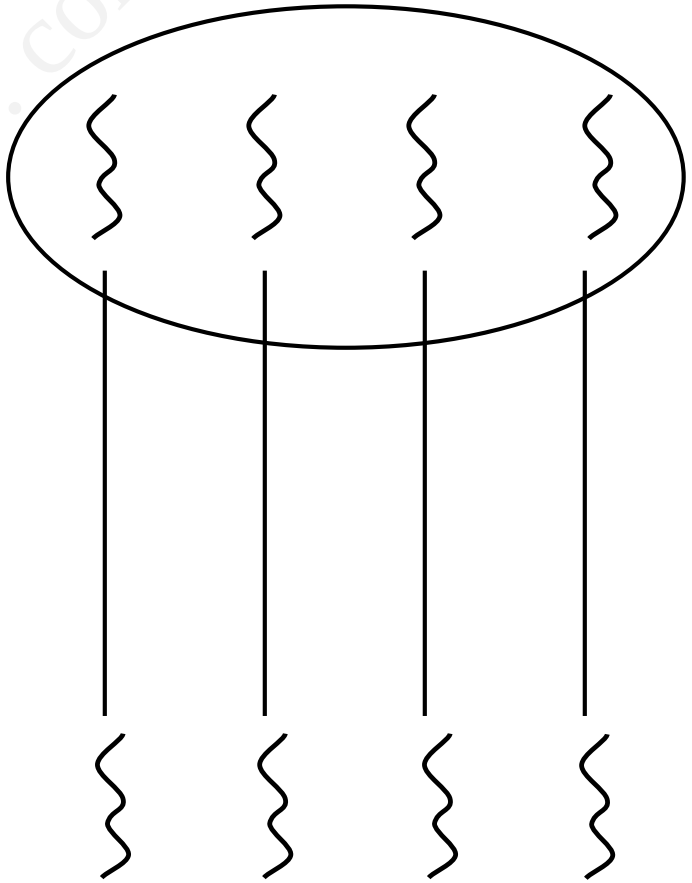


kernel thread

Tiểu trình

Mô hình one – to – one

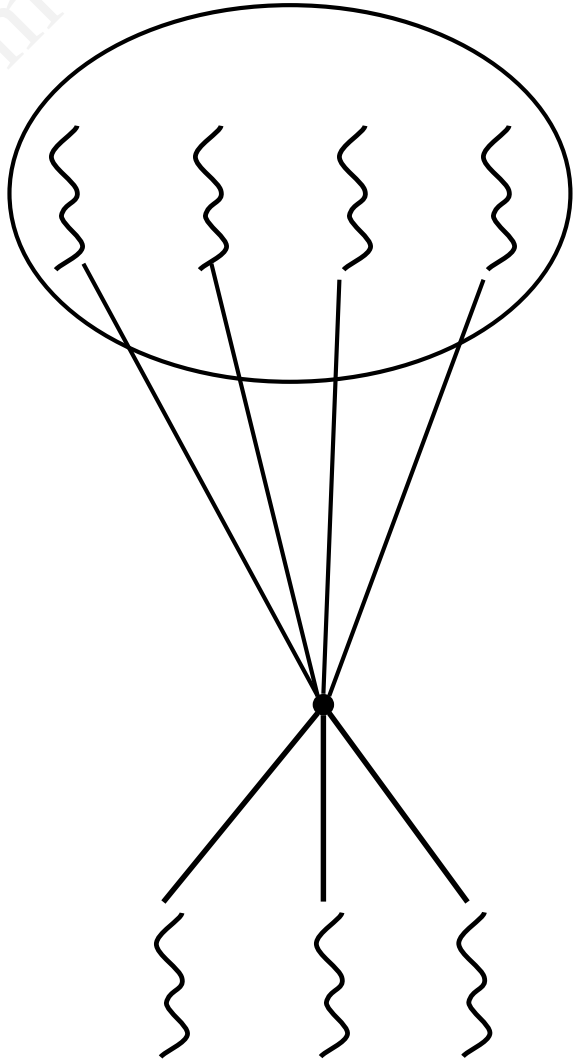
- Mỗi user-level thread thực thi thông qua một kernel thread riêng của nó
  - Mỗi khi một user thread được tạo ra thì cũng cần tạo một kernel thread tương ứng
- Hệ điều hành phải có cơ chế cung cấp được nhiều kernel thread cho một tiến trình



Tiểu trình

Mô hình many – to – many

- Nhiều user-level thread được phân chia thực thi (multiplexed) trên một số kernel thread.
  - Tránh được một số khuyết điểm cuûa hai mô hình many-to-one và one-to-one
- Ví dụ
  - Solaris 2
  - Windows NT/2000 với package ThreadFiber



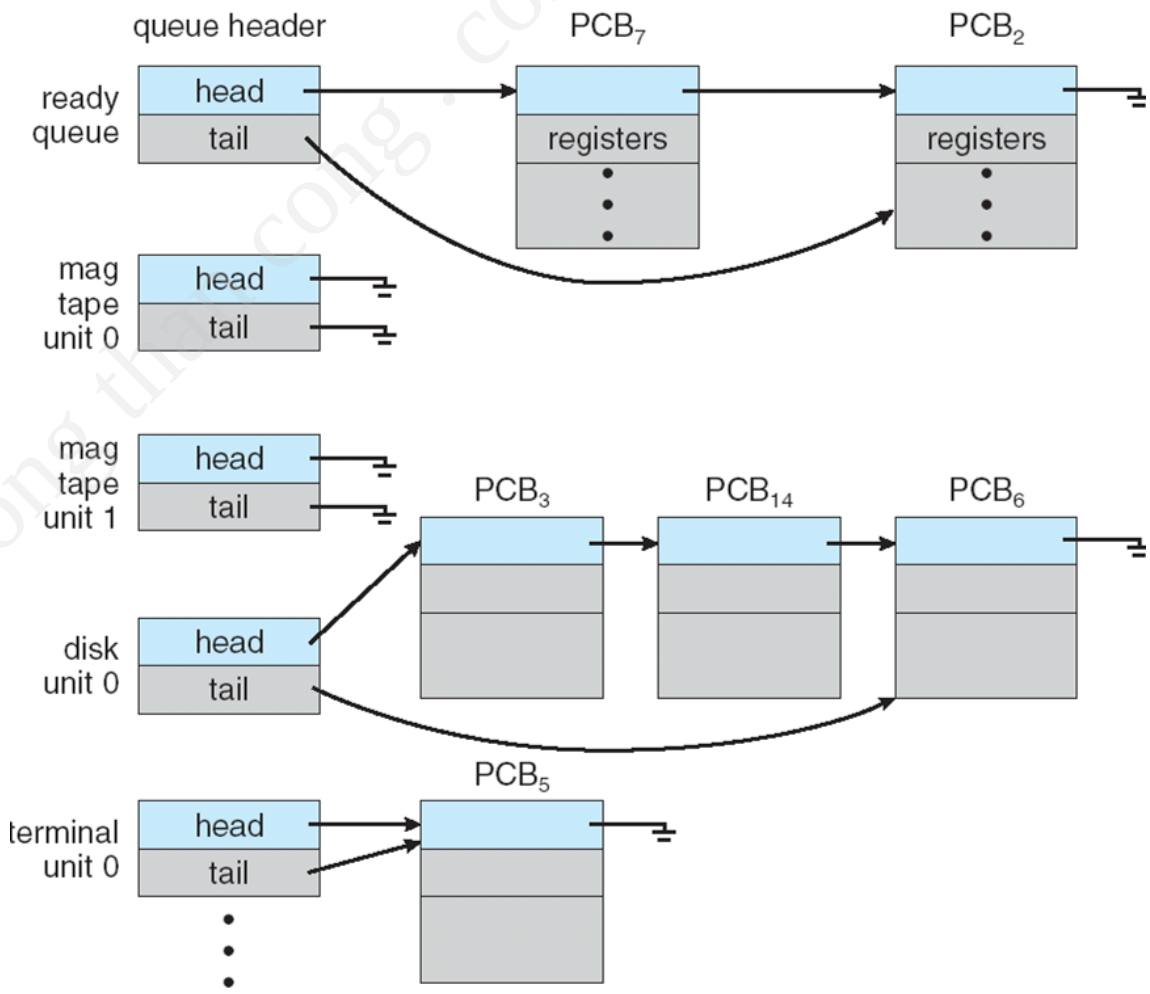
Lập lịch tiến trình

- Tại sao phải lập lịch?
  - Multiprogramming
    - ✓ Có nhiều tiến trình phải thực thi luân phiên nhau,
    - ✓ Mục tiêu: cực đại hiệu suất sử dụng của CPU.
  - Time-sharing
    - ✓ Cho phép users tương tác với tiến trình đang thực thi,
    - ✓ Mục tiêu: tối thiểu thời gian đáp ứng
- Một số khái niệm cơ bản
  - Các bộ định thời (scheduler)
  - Các hàng đợi định thời (scheduling queue)

Lập lịch tiến trình

Các hàng đợi định thời

- Job queue
- Ready queue
- Device queues
- …



Lập lịch tiến trình

Thêm medium-term scheduling

- Đôi khi hệ điều hành (như time-sharing system) có thêm medium-term scheduling để điều chỉnh mức độ multiprogramming của hệ thống.
- Medium-term scheduler
  - ✓ Chuyển tiến trình từ bộ nhớ sang đĩa (swap out)
  - ✓ Chuyển tiến trình từ đĩa vào bộ nhớ (swap in)

Lập lịch tiến trình

Một số khái niệm cơ bản

- Chu kỳ CPU-I/O: gồm chu kỳ thực thi CPU (CPU burst) và chu kỳ chờ đợi vào ra (I/O burst).
- *CPU-bound* process có thời gian sử dụng CPU nhiều hơn thời gian sử dụng I/O.
- *I/O-bound* process dùng phần lớn thời gian để đợi I/O.

Lập lịch tiến trình

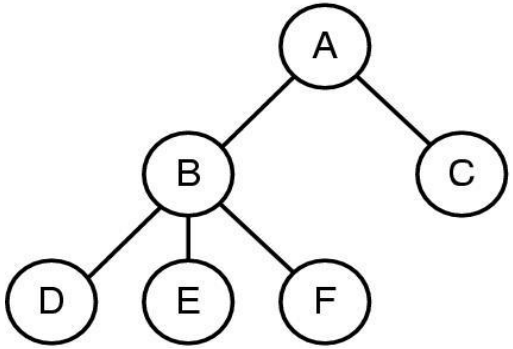
- Trong các hệ thống multitasking
  - Tại một thời điểm trong bộ nhớ có nhiều process
  - Tại mỗi thời điểm chỉ có một process được thực thi
  - Do đó, cần phải giải quyết vấn đề phân chia, lựa chọn process thực thi sao cho được hiệu quả nhất. Cần có chiến lược lập lịch cho các tiến trình.

Thao tác trên tiến trình

- Tạo lập tiến trình (create)
- Kết thúc tiến trình (destroy)
- Tạm dừng tiến trình (suspend)
- Tái kích hoạt tiến trình (resume)
- Thay đổi độ ưu tiên tiến trình (change priority)

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.



Kết thúc tiến trình

- 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

Cấp phát tài nguyên cho tiến trình

Khối quản lý tài nguyên

Các 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.

Điều phối tiến trình

- Hệ điều hành điều phối tiến trình thông qua bộ điều phối (scheduler) và bộ phân phối (dispatcher).
- Bộ điều phối sử dụng một giải thuật thích hợp để lựa chọn tiến trình được xử lý tiếp theo.
- Bộ phân phối chịu trách nhiệm cập nhật ngữ cảnh của tiến trình bị tạm ngưng và trao CPU cho tiến trình được chọn bởi bộ điều phối để tiến trình thực thi.

Mục tiêu điều phối

- ✓ Sự công bằng (Fairness)
- ✓ Tính hiệu qủa (Efficiency)
- ✓ Thời gian đáp ứng hợp lý (Response time)
- ✓ Thời gian lưu lại trong hệ thống (TurnaroundTime)
- ✓ Thông lượng tối đa (Throughput)

Các đặc điểm của tiến trình

- Tính hướng xuất / nhập của tiến trình (I/O-boundedness):
  - ✓ Nhiều lượt sử dụng CPU, mỗi lượt dùng thời gian ngắn.
- Tính hướng xử lý của tiến trình (CPU-boundedness):
  - ✓ Ít lượt sử dụng CPU, mỗi lượt dùng thời gian dài.
- Tiến trình tương tác *hay* xử lý theo lô
- Độ ưu tiên của tiến trình
- Thời gian đã sử dụng CPU của tiến trình
- Thời gian còn lại tiến trình cần để hoàn tất

Các nguyên lý điều phối

- *Điều phối độc quyền (preemptive)*
  - Độc chiếm CPU
  - Không thích hợp với các hệ thống nhiều người dùng
- *Điều phối không độc quyền (nopreemptive)*
  - Tránh được tình trạng một tiến trình độc chiếm CPU
  - có thể dẫn đến các mâu thuẫn trong truy xuất -> cần phương pháp đồng bộ hóa thích hợp để giải quyết.
  - phức tạp trong việc phân định độ ưu tiên
  - phát sinh thêm chi phí khi chuyển đổi CPU qua lại giữa các tiến trình

Thời điểm thực hiện điều phối

running -> blocked

Ví dụ chờ một thao tác nhập xuất hay chờ một tiến trình con kết thúc…

running -> ready

Ví dụ xảy ra một ngắt.

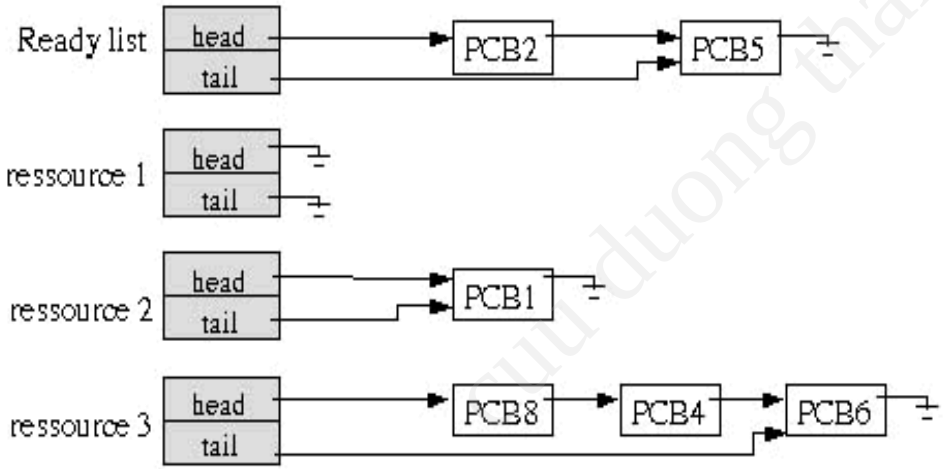
blocked -> ready

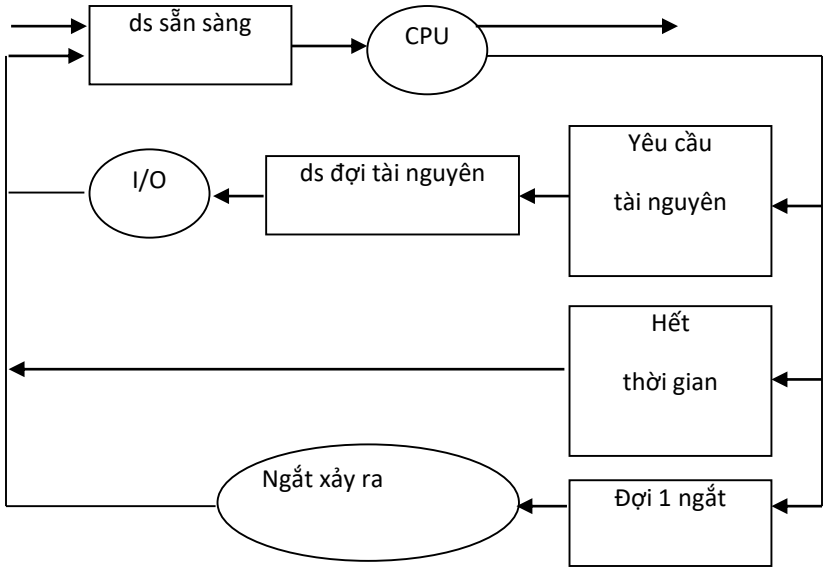
Ví dụ một thao tác nhập/xuất hoàn tất.

- Tiến trình kết thúc.
- Tiến trình có độ ưu tiên cao hơn xuất hiện
  - ✓chỉ áp dụng đối với điều phối không độc quyền

Tổ chức điều phối - Các danh sách điều phối

- ✓ Danh sách tác vụ (job list)
- ✓ Danh sách sẵn sàng (ready list)
- ✓ Danh sách chờ đợi (waiting list)





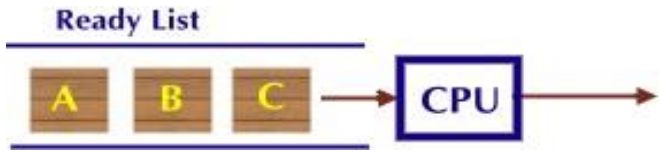
Các loại điều phối

- ***Điều phối tác vụ (job scheduling)***
  - Chọn tác vụ nào được đưa vào bộ nhớ chính để thực hiện
  - Quyết định mức độ đa chương
  - Tần suất hoạt động thấp
- ***Điều phối tiến trình (process scheduling)***
  - Chọn một tiến trình ở trạng thái sẵn sàng (đã được nạp vào bộ nhớ chính, và có đủ tài nguyên để hoạt động) để cấp phát CPU cho tiến trình đó thực hiện
  - Có tần suất hoạt động cao (1 lần/100 ms).
  - Sử dụng các thuật toán tốt nhất

Các thuật toán điều phối

- ***Thuật toán FIFO***
- ***Thuật toán phân phối xoay vòng (Round Robin)***
- ***Thuật toán độ ưu tiên***
- ***Thuật toán công việc ngắn nhất (Shortest-job-first SJF)***
- ***Thuật toán nhiều mức độ ưu tiên***
- ***Chiến lược điều phối xổ số (Lottery)***

Thuật toán FIFO



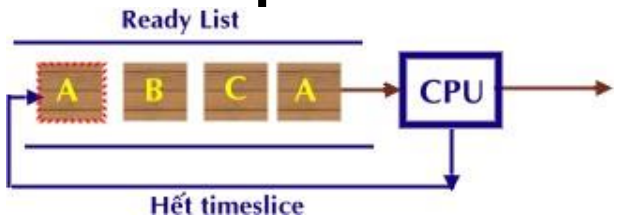
Điều phối theo nguyên tắc độc quyền

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

P1	P2	P3
0	24	27
		30

Thời gian chờ đợi được xử lý là 0 đối với P1, (24 -1) với P2 và (27-2) với P3. Thời gian chờ trung bình là (0+23+25)/3 = 16 miliseconds.

Thuật toán phân phối xoay vòng (Round Robin)



Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	24
P2	1	3
P3	2	3

P1	P2	P3	P1	P1	P1	P1	P1	
0	4	7	10	14	18	22	26	30

quantum là 4 miliseconds,

Thời gian chờ đợi trung bình sẽ là (6+3+5)/3 = 4.66 milisecondes.

Thuật toán độ ưu tiên

Độ ưu tiên P2 > P3> P1

Tiến trình	Thời điểm vào RL	Độ ưu tiên	Thời gian xử lý
P1	0	3	24
P2	1	1	3
P3	2	2	3

Thuật giải độ ưu tiên độc quyền

P1	P2	P3
0	24	27 30

Thời gian chờ đợi trung bình sẽ là (0+23+25)/3 =16 milisecondes

Thuật giải độ ưu tiên không độc quyền

P1	P2	P3	P1
0	1	4	7 30

Thời gian chờ đợi trung bình sẽ là (6+0+2)/3 = 2.7 milisecondes

Thuật toán công việc ngắn nhất (Shortest-job-first SJF)

t: thời gian xử lý mà tiến trình còn yêu cầu

Độ ưu tiên p = 1/t

Tiến trình	Thời điểm vào RL	Thời gian xử lý
P1	0	6
P2	1	8
P3	2	4
P4	3	2

Thuật giải SJF độc quyền

P1	P4	P3	P2
0	6	8	12 20

Thời gian chờ đợi trung bình sẽ là (0+11+6+3)/4 = 5 milisecondes

Thuật giải SJF không độc quyền

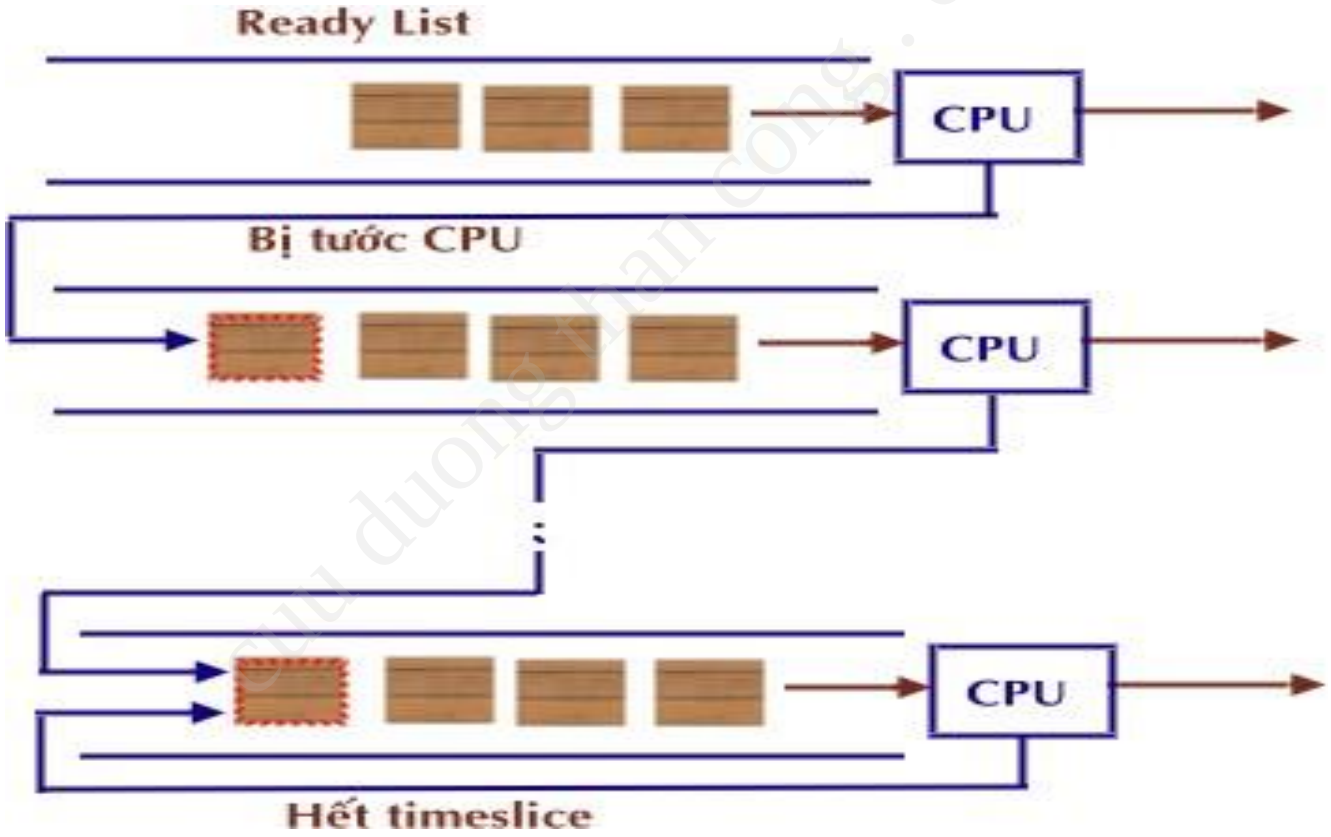
P1	P4	P1	P3	P2
0	3	5	8	12 20

Thời gian chờ đợi trung bình sẽ là (2+11+6+0)/4 = 4.75 milisecondes

Thuật toán nhiều mức độ ưu tiên

- Danh sách sẵn sàng được chia thành nhiều danh sách.
- Mỗi danh sách gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối riêng

Điều phối theo nhiều mức ưu tiên xoay vòng (Multilevel Feedback)



Chiến lược điều phối xổ số (Lottery)

- Mỗi tiến trình được cấp một "vé số".
- HĐH chọn 1 vé "trúng giải", tiến trình nào sỡ hữu vé này sẽ được nhận CPU.
- Là giải thuật độc quyền.
- Đơn giản, chi phí thấp, bảo đảm tính công bằng cho các tiến trình.

Bộ nhớ thực

❖ Khái niệm

- Quản lý bộ nhớ là công việc của hệ điều hành với sự hỗ trợ của phần cứng nhằm phân phối, sắp xếp các process trong bộ nhớ sao cho hiệu quả.
- Mục tiêu cần đạt được là nạp càng nhiều process vào bộ nhớ càng tốt (gia tăng mức độ đa chương trình)

Bộ nhớ thực

- **Khái niệm**
  - Các yêu cầu đối với việc quản lý bộ nhớ
    - ✓ Cấp phát bộ nhớ cho các process
    - ✓ Tái định vị (relocation): khi swapping,…
    - ✓ Bảo vệ: phải kiểm tra truy xuất bộ nhớ có hợp lệ không.
    - ✓ Chia sẻ: cho phép các process chia sẻ vùng nhớ chung.
    - ✓ Kết gắn địa chỉ nhớ luận lý của user vào địa chỉ thực.

Bộ nhớ thực

- **Địa chỉ nhớ**
  - *Địa chỉ vật lý* (physical address - địa chỉ *thực*): một vị trí thực trong bộ nhớ chính.
  - *Địa chỉ luận lý* (logical address): một vị trí nhớ được diễn tả trong một chương trình
    - ✓ Trình biên dịch (compiler) tạo ra mã lệnh chương trình trong đó mỗi tham chiếu bộ nhớ đều là địa chỉ luận lý
    - ✓ *Địa chỉ tương đối* (relative address): kiểu địa chỉ luận lý trong đó các địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó trong chương trình.

    Ví dụ: 12 byte so với vị trí bắt đầu chương trình,…
    - ✓ *Địa chỉ tuyệt đối* (absolute address): địa chỉ tương đương với địa chỉ thực

Bộ nhớ thực

❖ Địa chỉ nhớ

- ➢ Khi một lệnh được thực thi, các tham chiếu đến địa chỉ luận lý phải được chuyển đổi thành địa chỉ thực.
- ➢ Thao tác chuyển đổi thường có sự hỗ trợ của phần cứng để đạt hiệu suất cao

Bộ nhớ thực

❖ Chuyển đổi địa chỉ

- Là quá trình ánh xạ một địa chỉ từ không gian địa chỉ nay sang không gian địa chỉ khác.
- Biểu diễn địa chỉ nhớ

  Trong source code: symbolic (các biến, hằng, pointer,…)

  Thời điểm biên dịch: thường là địa chỉ tương đối

  Ví dụ: a ở vị trí 14 bytes so với vị trí bắt đầu của module.

  Thời điểm linking/loading: có thể là địa chỉ thực.

  Ví dụ: dữ liệu nằm tại địa chỉ bộ nhớ thực 1212

Bộ nhớ thực

❖ Chuyển đổi địa chỉ

- Địa chỉ lệnh (instruction) và dữ liệu (data) được chuyển đổi thành địa chỉ thực có thể xảy ra tại ba thời điểm khác nhau
  - ✓ Compile time: nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể gán địa chỉ tuyệt đối lúc biên dịch.
    - Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình
  - ✓ Load time: tại thời điểm biên dịch, nếu chưa biết tiến trình sẽ nằm ở đâu trong bộ nhớ thì compiler phải sinh mã địa chỉ tương đối. Vào thời điểm loading, loader phải chuyển đổi địa chỉ tương đố thành địa chỉ thực dựa trên một địa chỉ nền (base address).
    - Địa chỉ thực được tính toán vào thời điểm nạp chương trình ⇒ phải tiến hành reload nếu địa chỉ nền thay đổi.

Bộ nhớ thực

- **Chuyển đổi địa chỉ**
  - ✓ Execution time: khi trong quá trình thực thi, process có thể được di chuyển từ segment nay sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi
    – CPU tạo ra địa chỉ luận lý cho process
    – Phần cứng cần hỗ trợ cho việc ánh xạ địa chỉ.
    Ví dụ: trường hợp địa chỉ luận lý là tương đối thì có thể dùng thanh ghi base và limit,…
    – Sử dụng trong đa số các Hệ điều hành đa dụng (general-purpose) trong đó có các cơ chế swapping, paging, segmentation.

Bộ nhớ thực

❖ Cơ chế Overlay

- ➢ Tại mỗi thời điểm, chỉ giữ lại trong bộ nhớ những lệnh hoặc dữ liệu cần thiết, giải phóng các lệnh/dữ liệu chưa hoặc không cần dùng đến.
- ➢ Cơ chế này rất hữu dụng khi kích thước một process lớn hơn không gian bộ nhớ cấp cho process đó.
- ➢ Cơ chế này được điều khiển bởi người sử dụng (thông qua sự hỗ trợ của các thư viện lập trình) chứ không cần sự hỗ trợ của hệ điều hành

Bộ nhớ thực

❖ Cơ chế Swapping

- Một process có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ. Sau đó, process có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi.

  Ví dụ

  Round-robin: swap out P1 (vừa dùng hết quantum của nó), swap in P2 , thực thi P3 ,…

  Roll out, roll in: dùng trong cơ chế điều phối theo độ ưu tiên (priority-based scheduling)
- Process có độ ưu tiên thấp hơn sẽ bị swap out nhường chỗ cho process có độ ưu tiên cao hơn mới đến được nạp vào bộ nhớ để thực thi

Bộ nhớ thực

- **Phân mảnh**
  - Phân mảnh ngoại (external fragmentation)
    - ✓ Kích thước bộ nhớ còn trống đủ để đáp ứng một yêu cầu, nhưng bộ nhớ này không liên tục ⇒ có thể dùng cơ chế kết khối (compaction) để gom lại thành vùng nhớ liên tuc.
  - Phân mảnh nội (internal fragmentation)
    - ✓ Kích thước vùng nhớ được cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu.
    - ✓ Phân mảnh nội xảy ra khi bộ nhớ thực được chia thành các khối kích thước cố định và các process được cấp phát theo đơn vị khối.

    Ví dụ: cơ chế phân trang (paging).

Bộ nhớ thực

❖ Phân vùng

- Khi khởi động hệ thống, bộ nhớ chính được chia thành nhiều phần rời nhau gọi là các partition, có kích thước bằng nhau hoặc khác nhau

Bộ nhớ thực

Phân vùng – vùng cố định

❖ Process nào có kích thước nhỏ hơn hoặc bằng kích thước partition thì có thể được nạp vào partition đó.

❖ Nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.

❖ Không hiệu quả do bị phân mảnh nội: một chương trình dù lớn hay nhỏ đều được cấp phát trọn một partition

Bộ nhớ thực

❖ Phân vùng cố định - Chiến lược sắp xếp

- Partition có kích thước bằng nhau
  - ✓ Còn partition trống ⇒ process mới được nạp vào partition đó
  - ✓ Không còn partition trống, nhưng đó có process đang bị blocked ⇒ swap process đó ra bộ nhớ phụ nhường chỗ cho process mới
- Partition có kích thước không bằng nhau:
  - ✓ Giải pháp 1
    - – Gán mỗi process vào partition nhỏ nhất phù hợp với nó
    - – Có hàng đợi cho mỗi partition
    - – Giảm thiểu phân mảnh nội
  - ✓ Giải pháp 2
    - – Chỉ có một hàng đợi chung cho mọi partition
    - – Khi cần nạp một process vào bộ nhớ chính ⇒ chọn partition nhỏ nhất còn trống

Bộ nhớ thực

- **Phân vùng – vùng động**
  - Số lượng partition không cố định và partition có thể có kích thước khác nhau
  - Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết
  - Gây ra hiện tượng phân mảnh ngoài

Bộ nhớ thực

- **Phân vùng động - Chiến lược sắp xếp**
  - Dùng để quyết định cấp phát khối bộ nhớ trống nào cho một process
  - Mục tiêu: giảm chi phí kết khối (compaction)
  - Các chiến lược placement:
    - ✓ *Best-fit:* chọn khối nhớ trống nhỏ nhất
    - ✓ *First-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ đầu bộ nhớ
    - ✓ *Next-fit:* chọn khối nhớ trống phù hợp đầu tiên kể từ vị trí cấp phát cuối cùng
    - ✓ *Worst-fit:* chọn khối nhớ trống lớn nhất

Bộ nhớ ảo

Khái niệm

- Bộ nhớ ảo (virtual memory)

  Cơ chế được hiện thực trong hệ điều hành để cho phép thực thi một tiến trình mà chỉ cần giữ trong bộ nhớ chính một phần của không gian địa chỉ luận lý của nó, còn phần còn lại được giữ trên bộ nhớ phụ (đĩa cứng).

- Ưu điểm của bộ nhớ ảo

  Số lượng process trong bộ nhớ nhiều hơn

  Một process có thể thực thi ngay cả khi kích thước của nó lớn hơn bộ nhớ thực

Bộ nhớ ảo

Khái niệm

- Thông thường phần không gian địa chỉ luận lý của tiến trình, nếu chưa cần nạp vào bộ nhớ chính sẽ được giữ ở một vùng đặc biệt trên đĩa gọi là không gian tráo đổi (swap space).

Ví dụ:

- ✓ swap partition trong Linux
- ✓ file pagefile.sys trong Windows 2K

Bộ nhớ ảo

Hiện thực bộ nhớ ảo

- Phần cứng quản lý bộ nhớ phải hỗ trợ phân trang và/hoặc phân đoạn
- Hệ điều hành phải quản lý sự di chuyển của trang/đoạn giữa bộ nhớ chính và bộ nhớ thứ cấp

Bộ nhớ ảo

Cơ chế phân trang

- Cho phép không gian địa chỉ thực (physical address space) của một process có thể không liên tục nhau.
- Bộ nhớ thực được chia thành các khối cố định và có kích thước bằng nhau gọi là frame.
- Thông thường kích thước của frame là lũy thừa của 2, từ khoảng 512 byte đến 16MB.
- Bộ nhớ luận lý (logical memory) hay không gian địa chỉ luận lý là tập mọi địa chỉ luận lý mà một chương trình bất kỳ có thể sinh ra.
- Bộ nhớ luận lý cũng được chia thành các khối cố định có cùng kích thước gọi là trang nhớ (page).

Bộ nhớ ảo

Cơ chế phân trang

- Frame và trang nhớ có kích thước bằng nhau.
- Hệ điều hành phải thiết lập một bảng phân trang (page table) để ánh xạ địa chỉ luận lý thành địa chỉ thực
  - Mỗi process có một bảng phân trang, được quản lý bằng một con trỏ lưu giữ trong PCB. Công việc thiết lập bằng phân trang cho process là một phần của chuyển ngữ cảnh
- Cơ chế phân trang khiến bộ nhớ bị phân mảnh nội, tuy nhiên lại khắc phục được phân mảnh ngoại.

Bộ nhớ ảo

Cơ chế phân đoạn

- Một chương trình cấu thành từ nhiều đoạn (segment). Mỗi đoạn là một đơn vị luận lý của chương trình, như:
  - Main program, procedure, function
  - Local variables, global variables, common block, stack, symbol table, arrays,…
- Thông thường, một chương trình được biên dịch. Trình biên dịch sẽ tự động xây dựng các đoạn.
- Trình Loader sẽ gán mỗi đoạn một số định danh riêng

Bộ nhớ ảo

Phần cứng hỗ trợ bộ nhớ ảo

- Mỗi mục của bảng phân trang có thêm các bit trạng thái đặc biệt:
  - ✓ Present bit = 1 ⇒ trang hợp lệ, đang trong bộ nhớ
    = 0 ⇒ trang không hợp lệ/không trong bộ nhớ
  - ✓ Modified bit: cho biết trang có thay đổi kể từ khi được nạp vào bộ nhớ hay không

Bộ nhớ ảo

Hiện thực bộ nhớ ảo

- Demand paging: các trang của tiến trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu.
- Khi có tham chiếu đến một trang không có trong bộ nhớ chính (present bit = 0) thì phần cứng sẽ gây ra một ngắt (*page-fault trap*) kích khởi *page-fault service routine* (PFSR) của hệ điều hành.

Bộ nhớ ảo

Hiện thực bộ nhớ ảo

- Bộ PFSR có nhiệm vụ:
  1. Chuyển tiến trình về trạng thái blocked
  2. Phát ra một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trong; trong khi đợi I/O, một tiến trình khác được cấp CPU để thực thi
  3. Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành; PFSR cấp nhật page table và chuyển tiến trình về trạng thái ready.

Bộ nhớ ảo

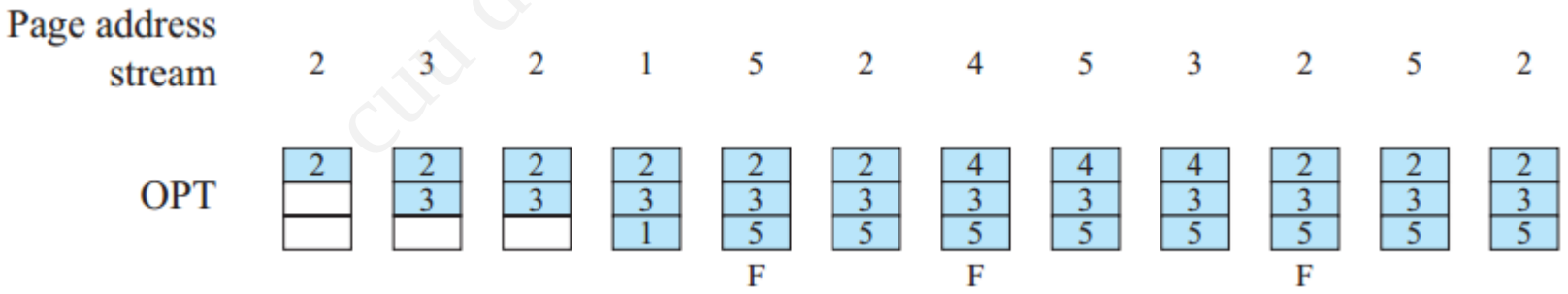
Chiến lược sắp đặt

- Là tập các trang được tham chiếu gần nhau
- Một tiến trình gồm nhiều trang, và trong quá trình thực thi, tiến trình sẽ chuyển từ trang này sang trang khác

Bộ nhớ ảo

Giải thuật thay trang OPT - Optimal

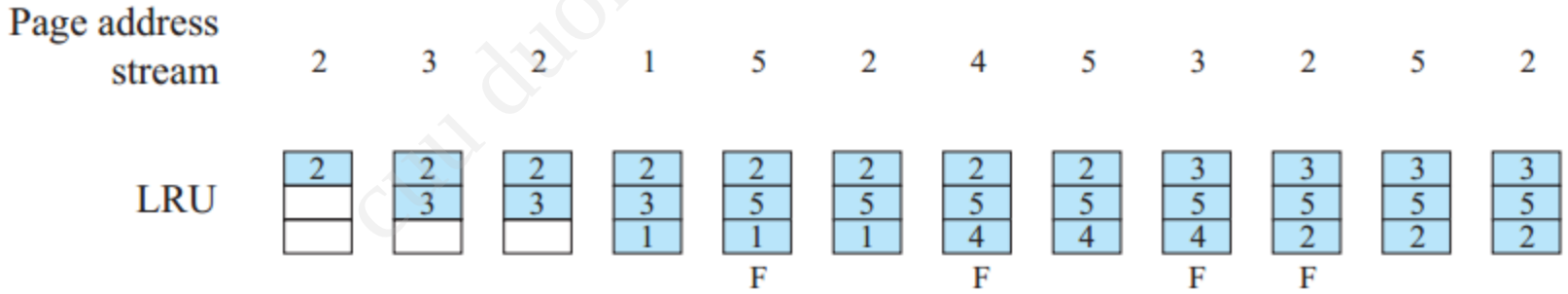
- Thay thế trang nhớ sẽ được tham chiếu trễ nhất trong tương lai
- Ví dụ: một process có 5 trang, và được cấp 3 frame



Bộ nhớ ảo

Giải thuật thay trang *Least Recently Used* (LRU)

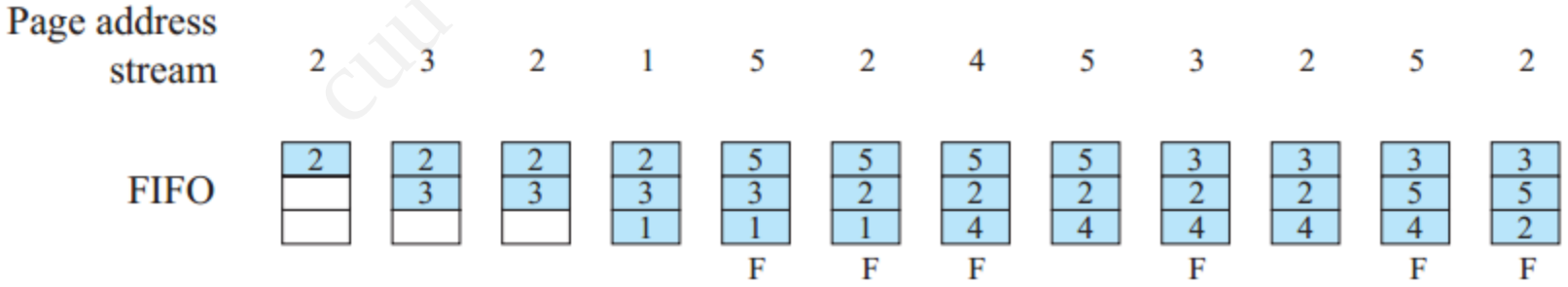
- Thay thế trang nhớ không được tham chiếu lâu nhất
- Ví dụ: một process có 5 trang, và được cấp 3 frame



Bộ nhớ ảo

Giải thuật thay trang *FIFO*

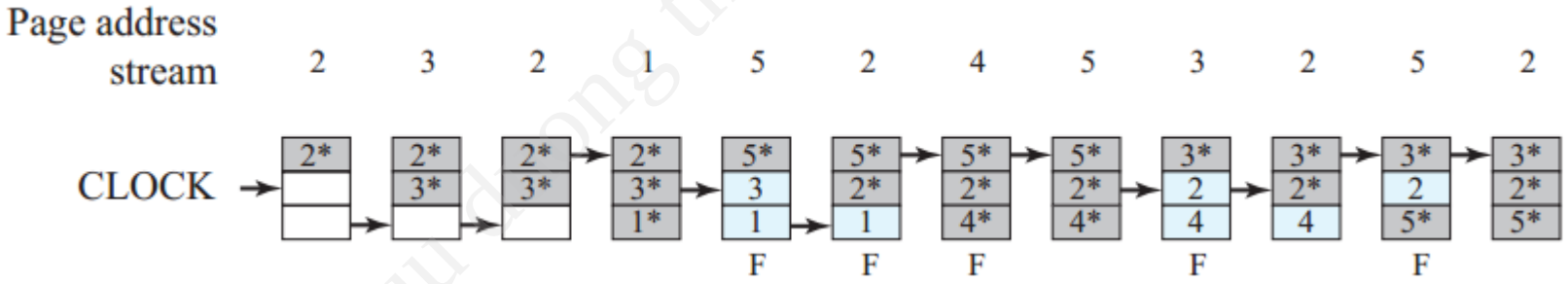
- Xem các frame được cấp phát cho process như circular buffer
  - ✓ Khi bộ đệm đầy, trang nhớ cũ nhất sẽ được thay thế: FIFO
  - ✓ Một trang nhớ hay được dùng sẽ thường là trang cũ nhất
    $\Rightarrow$ hay bị thay thế bởi giải thuật FIFO
  - ✓ Đơn giản: cần một con trỏ xoay vòng các frame của process



Bộ nhớ ảo

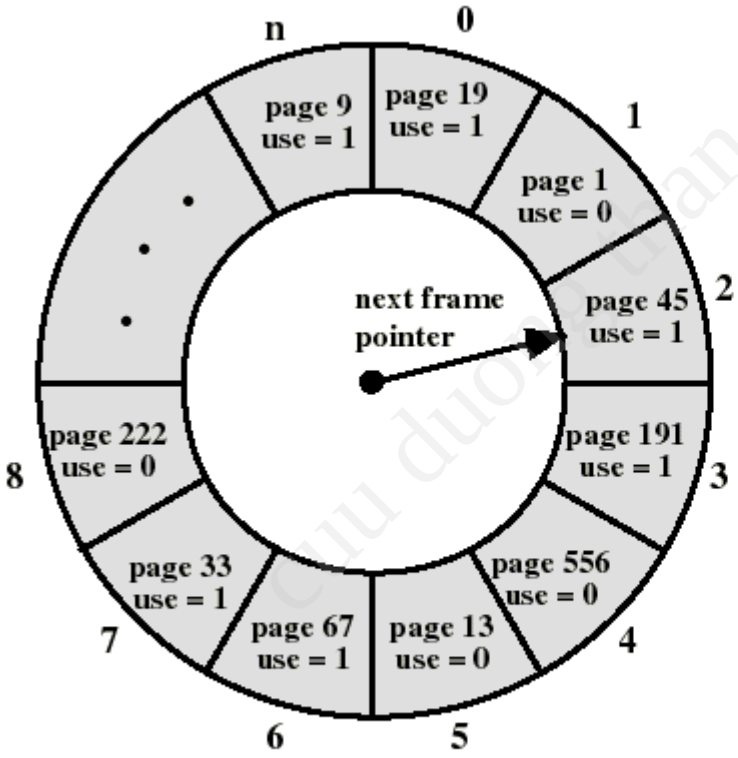
Giải thuật thay trang *Clock*

Ví dụ: một process có 5 trang, và được cấp 3 frame

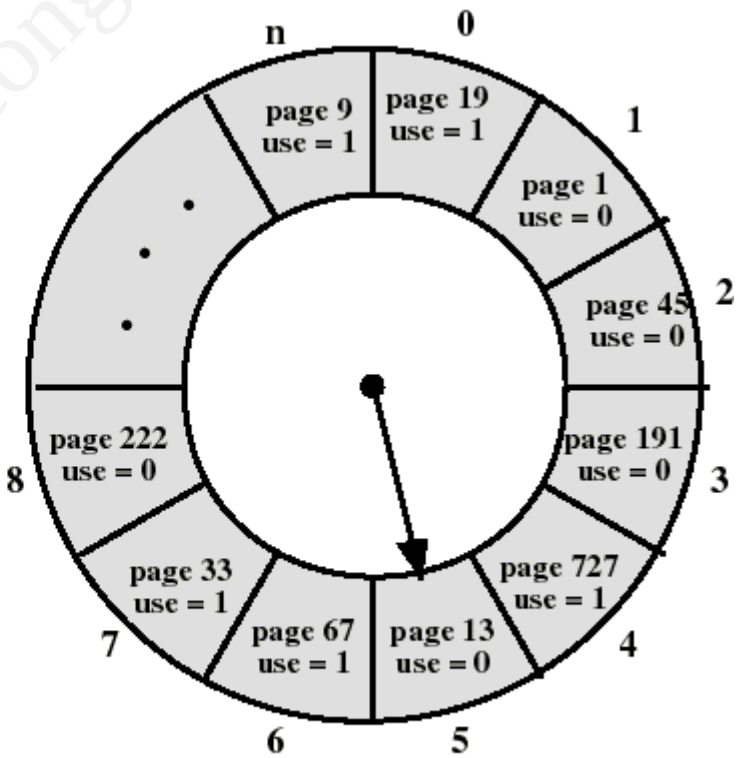


Bộ nhớ ảo

Giải thuật thay trang *Clock*



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Bộ nhớ ảo

Chiến lược cấp phát frame

- Hệ điều hành phải quyết định cấp cho mỗi process bao nhiêu frame.
  - ✓ Cấp ít frame: nhiều page fault
  - ✓ Cấp nhiều frame: giảm mức độ multiprogramming

Bộ nhớ ảo

Chiến lược cấp phát frame

- Chiến lược cấp phát tĩnh (fixed-allocation)
  - ✓ Số frame cấp cho mỗi process không đổi, được xác định vào thời điểm loading và có thể tùy thuộc vào từng ứng dụng (kích thước của nó,…)
- Chiến lược cấp phát động (variable-allocation)
  - ✓ Số frame cấp cho mỗi process có thể thay đổi trong khi nó chạy
    - – Nếu tỷ lệ page-fault cao ⇒ cấp thêm frame
    - – Nếu tỷ lệ page-fault thấp ⇒ giảm bớt frame
  - ✓ Hệ điều hành phải mất chi phí để ước định các process

Bộ nhớ ảo

Chiến lược cấp phát frame

- Cấp phát bằng nhau:
  - ✓ Không quan tâm đến các yếu tố khác nhau của process
  - ✓ Ví dụ, có 100 frame và 5 process → mỗi process được 20 frame
- Cấp phát theo tỉ lệ:
  - ✓ Dựa vào kích thước process

Bộ nhớ ảo

Thrashing

- Là hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục
- Nếu một process không có đủ số frame cần thiết thì tỉ số page faults/sec rất cao. Điều này khiến giảm hiệu suất CPU rất nhiều.
- Ví dụ: một vòng lặp N lần, mỗi lần tham chiếu đến địa chỉ nằm trong 4 trang nhớ trong khi process chỉ được cấp 3 frames