NGUYÊN LÝ HỆ ĐIỀU HÀNH
Ths. GVC. Đoàn Hòa Minh
Khoa Công nghệ thông tin
Trường Đại học Nam Cần Thơ

Email: dhminh@ctu.edu.vn

CHƯƠNG 4: TIẾN TRÌNH VÀ DÒNG

4.1. Tiến trình
4.2. Tiểu trình
4.3. Điều phối tiến trình
4.4. Đồng bộ hóa tiến trình
4.5. Tắc nghẽn
4.6. Quản lý bộ nhớ

5.1. TIẾN TRÌNH (PROCESS)

4.1. Tiến trình

1. Khái niệm
2. Mô hình tiến trình
3. Các trạng thái của tiến trình
4. Chế độ xử lý của tiến trình
5. Cấu trúc dữ liệu khối quản lý tiến trình
6. Thao tác trên tiến trình
7. Chuyển đổi ngữ cảnh (Context switch)
8. Cấp phát tài nguyên cho tiến trình

4.1. TIẾN TRÌNH (PROCESS)

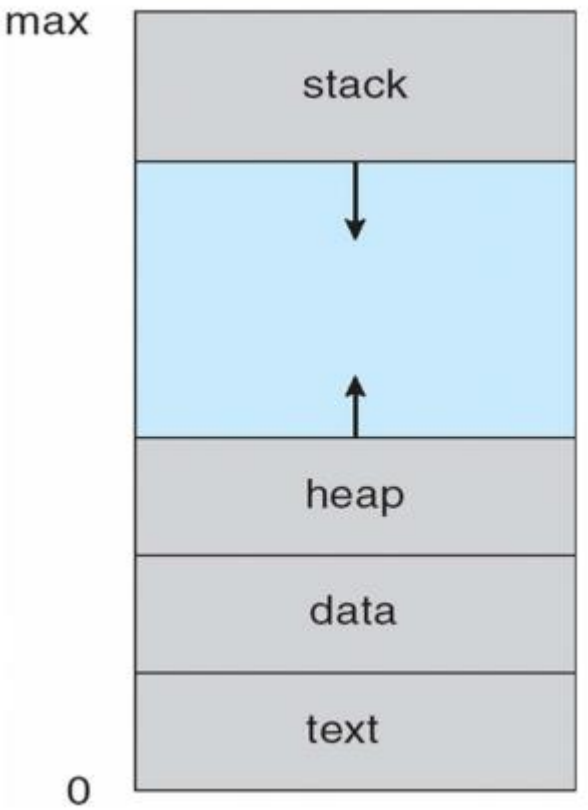
1. KHÁI NIỆM

- Hệ điều hành có thể thực thi nhiều dạng chương trình, chẳng hạn như:
  - Hệ thống bó (batch system) – jobs
  - Hệ thống chia thời gian – chương trình người dùng (user program) hay tác vụ (task)
- Tiến trình - là một chương trình đang thực thi. Sự thực thi của tiến trình diễn ra theo nguyên tắc tuần tự.
- Một tiến trình bao gồm:
  - Mã chương chương trình (program code hay text section)
  - Bộ đếm chương trình (PC: program counter)
  - Bộ nhớ Ngăn xếp (stack) – Vùng nhớ dữ liệu, chứa các dữ liệu tạm thời, như: các tham số của hàm, địa chỉ trở về, biến cục bộ,...
  - Bộ nhớ Heap – Vùng nhớ dữ liệu, bộ nhớ động, chứa các biến toàn cục, được cấp phát trong suốt thời gian thực thi.

4.1. TIẾN TRÌNH (PROCESS)

1. KHÁI NIỆM

- **Chương trình** là một thực thể bị động lưu trữ trên đĩa, **tiến trình** là thực thể chủ động
- **Chương trình trở thành tiến trình** khi nó được nạp vào trong bộ nhớ để thực thi
- Chương trình có thể được kích hoạt qua thao tác nhấp chuột, nhập tên vào CLI (Command Line Interface).
- Một chương trình có thể có vài tiến trình.



4.1. TIẾN TRÌNH (PROCESS)

2. MÔ HÌNH TIẾN TRÌNH

- Để hỗ trợ tính đa chương, máy tính phải có khả năng thực hiện nhiều tác vụ đồng thời.
- Nhưng việc điều khiển nhiều hoạt động song song ở cấp độ phần cứng là rất khó → đề xuất mô hình song song giả lập: chuyển bộ xử lý qua lại giữa các tiến trình (HĐH phân chia thời gian).
- Mỗi tiến trình xem như sở hữu bộ xử lý "ảo" riêng của nó (Nhưng trong thực tế, chỉ có một bộ xử lý thật sự được chuyển đổi qua lại giữa các tiến trình).
- **Bộ điều phối (scheduler)**: quyết định thời điểm cần dừng hoạt động của tiến trình đang xử lý để phục vụ một tiến trình khác và lựa chọn tiến trình tiếp theo sẽ được phục vụ.

4.1. TIẾN TRÌNH (PROCESS)

3. CÁC TRẠNG THÁI CỦA TIẾN TRÌNH

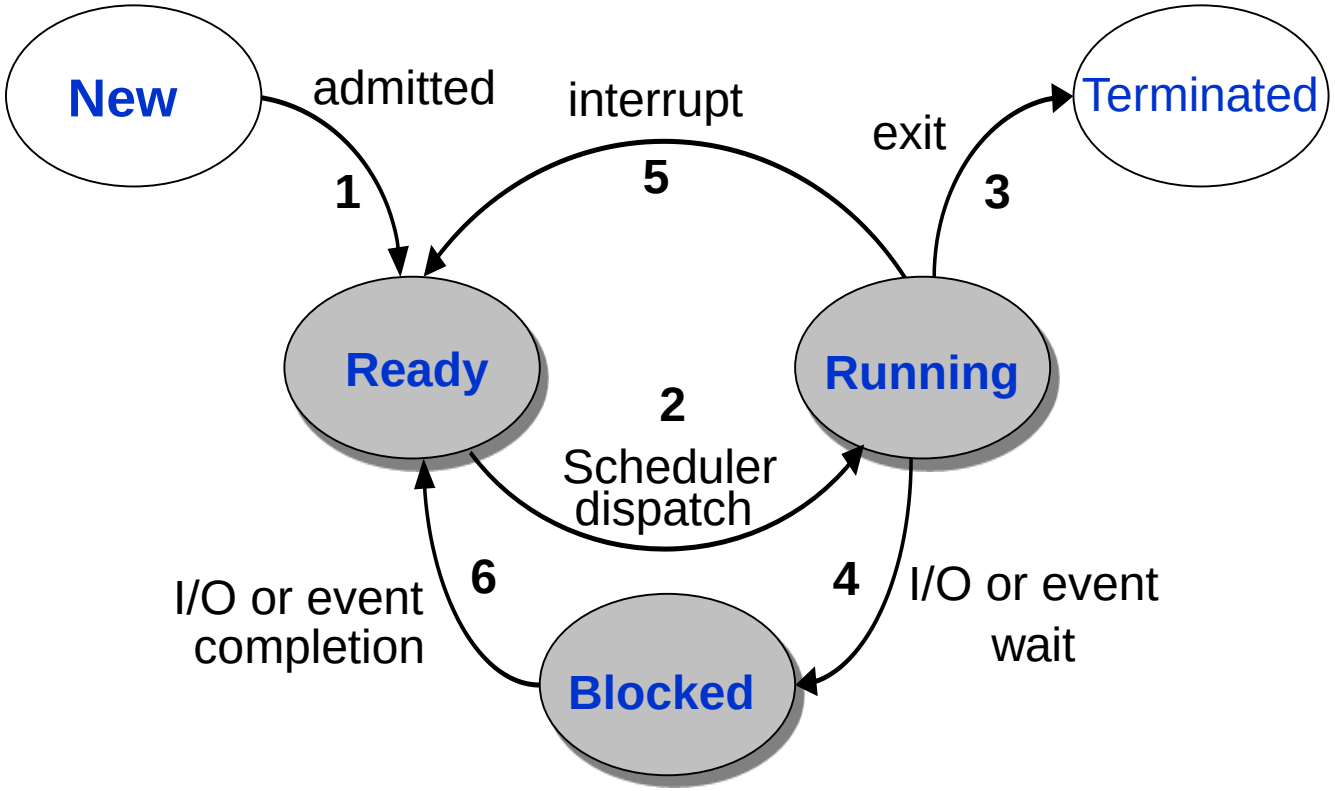
- Trạng thái của tiến trình tại mỗi thời điểm được xác định bởi hoạt động hiện thời của nó bao gồm:

1) **New:** tiến trình đang được tạo lập
2) **Running:** tiến trình đang được xử lý
3) **Blocked / Waiting**: tiến trình tạm dừng và chờ vì thiếu tài nguyên hay chờ một sự kiện nào đó
4) **Ready:** tiến trình đã sẵn sàng, đang chờ cấp CPU
5) **Terminated:** Tiến trình hoàn tất xử lý

➢ Tại một thời điểm, chỉ có một tiến trình có thể nhận trạng thái running trên một bộ xử lý bất kỳ. Trong khi đó nhiều tiến trình có thể ở trạng thái blocked hay ready.

4.1. TIẾN TRÌNH (PROCESS)

Mô tả chuyển trạng thái của tiến trình



(1)Tiến trình mới được đưa vào hệ thống
(2) Bộ điều phối cấp phát thời gian sử dụng CPU cho tiến trình
(3) kết thúc tiến trình
(4) Tiến trình chờ cấp phát tài nguyên hay nhập/xuất
(5) Bộ điều phối chọn một tiến trình khác để xử lý
(6) Tài nguyên đã sẵn sàng, chờ cấp CPU

4.1. TIẾN TRÌNH (PROCESS)

4. CHẾ ĐỘ XỬ LÝ CỦA TIẾN TRÌNH (1)

- Để đảm bảo hệ thống hoạt động đúng đắn, **HĐH** cần được bảo vệ khỏi sự xâm phạm của các tiến trình. **Bản thân các tiến trình và dữ liệu** cũng cần được bảo vệ để tránh các ảnh hưởng sai lạc lẫn nhau.
- **Cơ chế bảo vệ:** phân biệt hai chế độ xử lý cho các tiến trình, chế độ đặc quyền và chế độ không đặc quyền nhờ vào sự trợ giúp của cơ chế phần cứng.
- **Tập lệnh của CPU** được phân chia thành các lệnh đặc quyền và lệnh không đặc quyền. Cơ chế phần cứng chỉ cho phép các lệnh đặc quyền được thực hiện trong chế độ đặc quyền.
- Thông thường **chỉ có HĐH hoạt động trong chế độ đặc quyền**, các tiến trình của người dùng hoạt động trong chế độ không đặc quyền, không thực hiện được các lệnh có nguy cơ ảnh hưởng đến hệ thống. Như vậy hệ điều hành sẽ được bảo vệ.

4.1. TIẾN TRÌNH (PROCESS)

4. CHẾ ĐỘ XỬ LÝ CỦA TIẾN TRÌNH (2)

Hai chế độ xử lý của tiến trình

4.1. TIẾN TRÌNH (PROCESS)

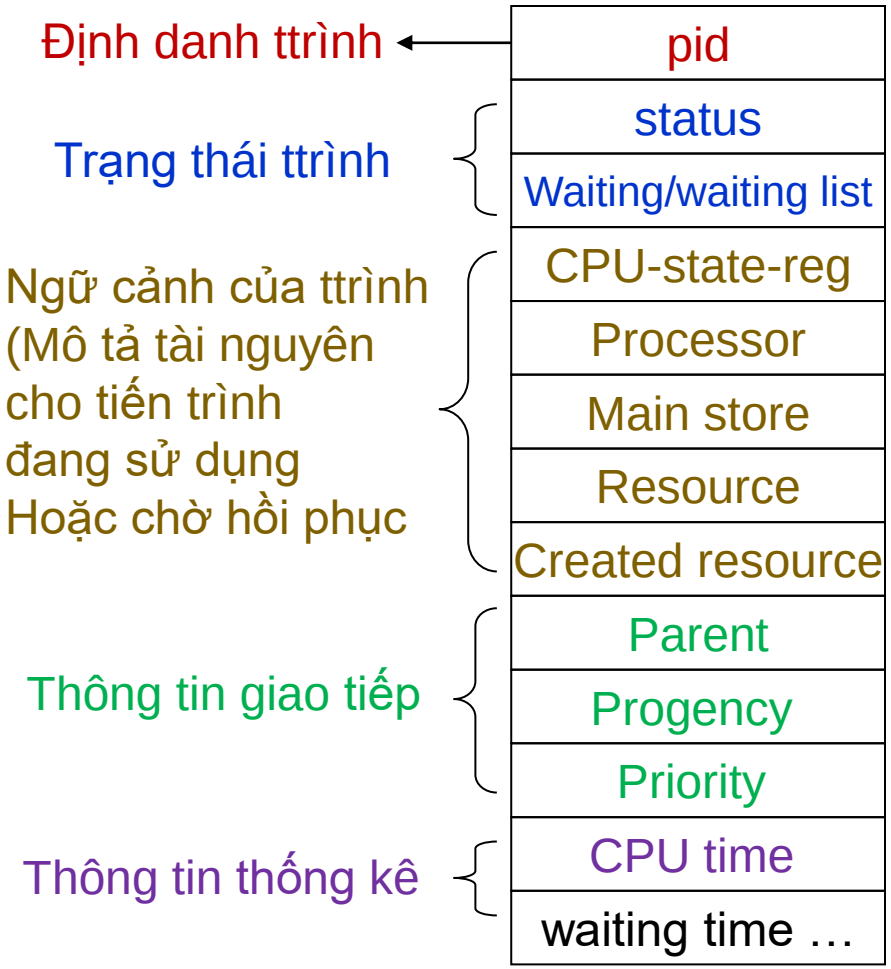
5. CẤU TRÚC DỮ LIỆU KHỐI QUẢN LÝ TIẾN TRÌNH (PCB: Process Control Block)

➢**PCB là vùng nhớ lưu trữ thông tin mô tả tiến trình bao gồm:**

1) **Định danh:** phân biệt các tiến trình
2) **Trạng thái của tiến trình:** xác định hoạt động hiện hành
3) **Ngữ cảnh:** mô tả các tài nguyên dành cho tiến trình như
   - **Trạng thái CPU:** con trỏ lệnh IP/PC (Program Counter: lưu địa chỉ lệnh kế tiếp), nội dung các thanh ghi (accumulators, general-purpose registers, thanh ghi chỉ mục, con trỏ ngăn xếp), các thông tin này cần được lưu trữ khi xảy ra một ngắt, nhằm có thể phục hồi hoạt động của tiến trình đúng như trước lúc bị ngắt.
   - **Bộ xử lý:** dùng trong hệ thống nhiều CPU, xác định CPU mà tiến trình đang dùng,
   - **Bộ nhớ chính, tài nguyên sử dụng, tài nguyên tạo lập,**…
4) **Thông tin giao tiếp (với TT khác):** tiến trình cha, tiến trình con, độ ưu tiên
5) **Thông tin thống kê** về hoạt động của tiến trình (tgian sử dụng CPU, tgian chờ,…)

4.1. TIẾN TRÌNH (PROCESS)

5. Cấu trúc dữ liệu khối quản lý tiến trình



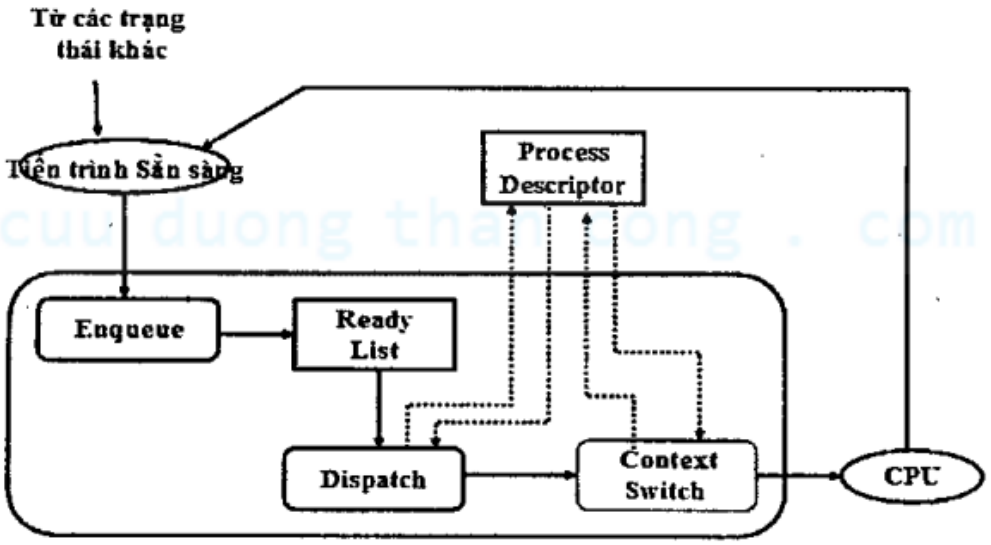
4.1. TIẾN TRÌNH (PROCESS)

• Cấu trúc bộ điều phối tiến trình (*)

Bộ điều phối tiến trình có 3 thành phần cơ bản là:

- **Enqueuer:** đưa tiến trình vào hang đợi.
- **Dispitcher:** bộ điều vận.
- **Context Switcher:** bộ chuyển ngữ cảnh

➢ Khi tiến trình chuyển sang trang thái sẵn sàng, khối PCB tương ứng được cập nhật đồng thời Enqueuer đặt co trỏ để khối PCB này nằm trong danh sách sẵn sang và tính chỉ số ưu tiên cấp CPU

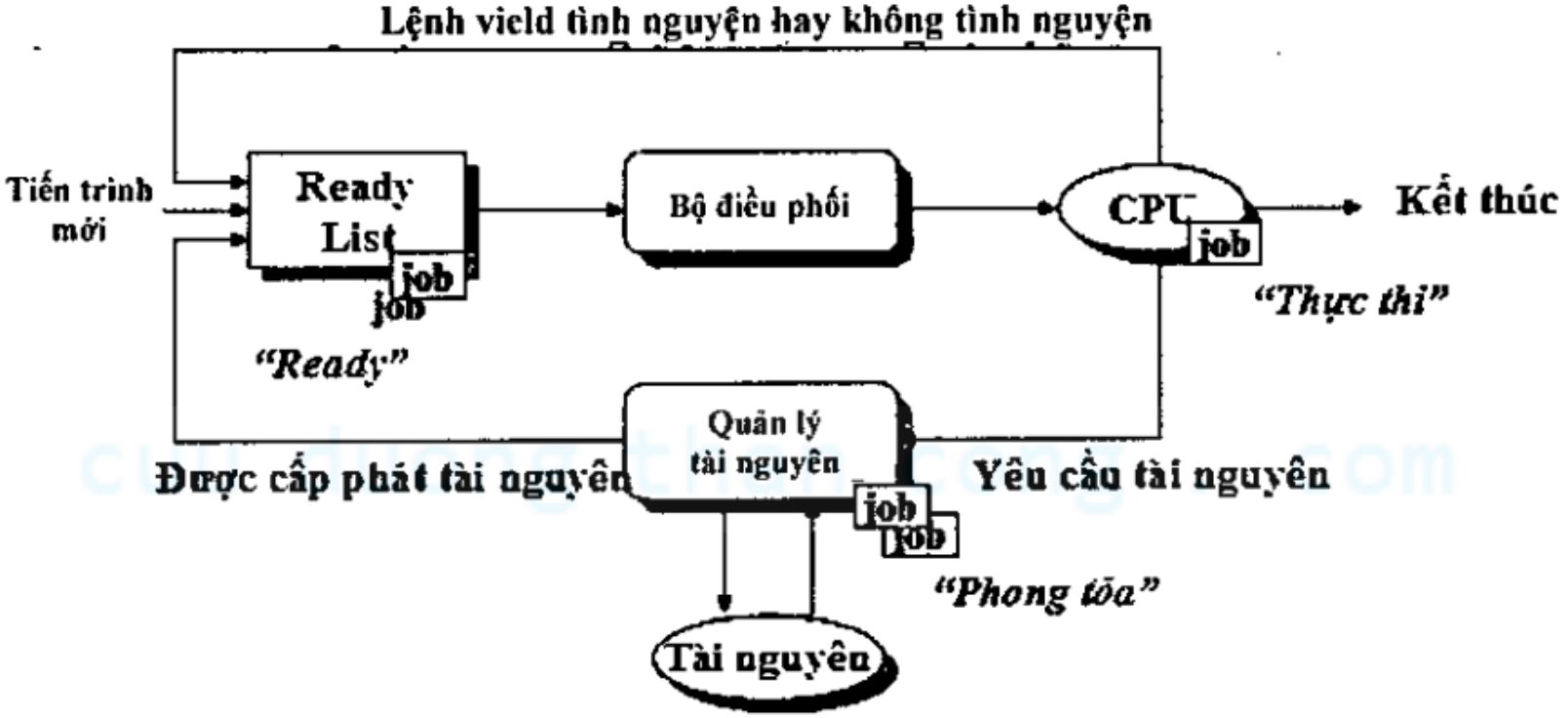


➢ Context Switcher lưu giá trị các thanh ghi vào PCB của tiến trình hiện hành để chuyển quyền sử dụng CPU cho tiến trình khác.

➢ Dispitcher thực thi lựa chọn cấp phát CPU cho tiến trình sẵn sang khác

4.1. TIẾN TRÌNH (PROCESS)

Hình sau đây mô tả luồng đi của tiến trình trong hệ thống có 1 CPU. Tiến trình tự động giải phóng CPU khi nó yêu cầu tài nguyên hoặc bị cưỡng đoạt quyền sử dụng CPU.(*)



4.1. TIẾN TRÌNH (PROCESS)

6. Thao tác trên tiến trình

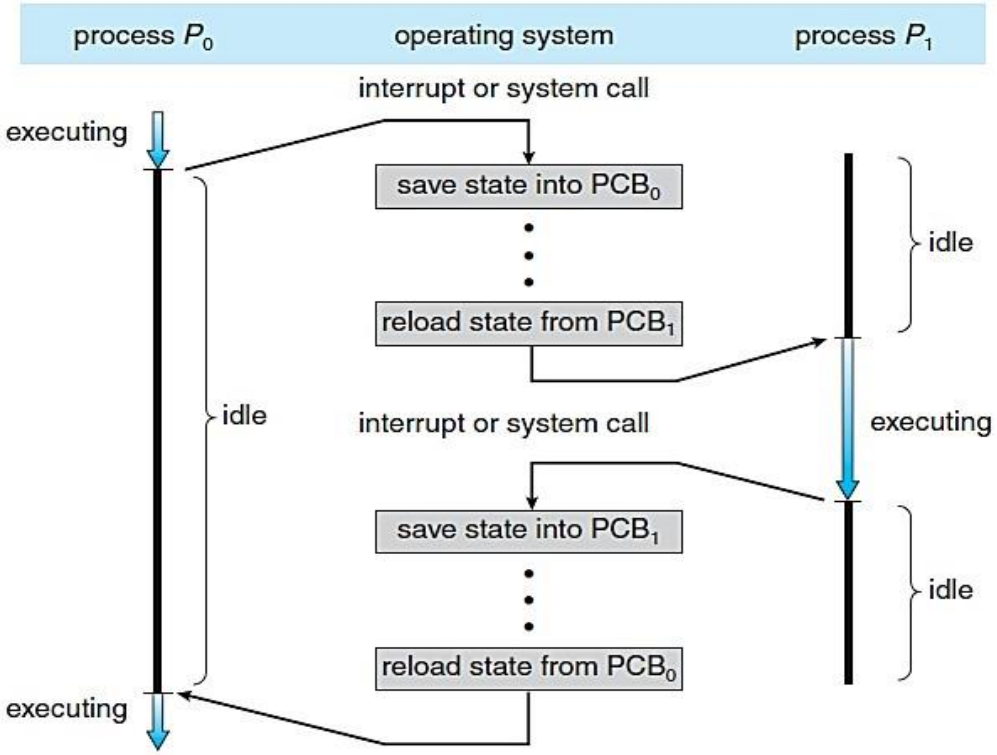
Hệ điều hành cung cấp các thao tác chủ yếu sau đây trên một tiến trình:

- Tạo lập tiến 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

4.1. TIẾN TRÌNH (PROCESS)

7. Chuyển đổi ngữ cảnh

- Là quá trình chuyển đổi CPU từ tiến trình này sang tiến trình khác, HĐH phải thực hiện lưu trữ trạng thái của tiến trình đang thực hiện và phục hồi trạng thái của tiến trình cần chuyển đến.
- Thời gian chuyển đổi ngữ cảnh mặc dù thường rất ngắn (vài ms), nhưng đây là khoảng thời gian lãng phí vì CPU không làm việc có ích.



4.1. TIẾN TRÌNH (PROCESS)

8. Cấp phát tài nguyên cho tiến trình (1)

➢**Vấn đề:** Khi có nhiều người sử dụng đồng thời làm việc, HĐH phải cấp phát các tài nguyên theo yêu cầu. Do tài nguyên hệ thống có giới hạn nên hiếm khi tất cả các yêu cầu tài nguyên đồng thời đều được thỏa mãn → cần phải nghiên cứu một phương pháp để chia sẻ một số tài nguyên hữu hạn giữa nhiều tiến trình người dùng đồng thời.

➢Mỗi tài nguyên được biểu diễn thông qua một cấu trúc dữ liệu khác nhau, nhưng cơ bản chứa đựng các thông tin sau:

- **Định danh tài nguyên**
- **Trạng thái tài nguyên:** phần nào đã cấp phát cho tiến trình, phần nào còn có thể sử dụng?
- **Hàng đợi trên một tài nguyên:** danh sách các tiến trình đang chờ được cấp phát tài nguyên tương ứng.
- **Bộ cấp phát:** là đoạn code cấp phát một tài nguyên đặc thù. Một số tài nguyên đòi hỏi các giải thuật đặc biệt (CPU, bộ nhớ chính, hệ thống tập tin), trong khi những tài nguyên khác (các thiết bị nhập xuất,…) có thể cần các giải thuật cấp phát và giải phóng tổng quát hơn.

4.1. TIẾN TRÌNH (PROCESS)

8. Cấp phát tài nguyên cho tiến trình (2)

Các mục tiêu của giải thuật cấp phát tài nguyên:

1) Bảo đảm một số lượng hợp lệ các tiến trình truy xuất đồng thời đến các tài nguyên không chia sẻ được.
2) Cấp phát tài nguyên cho tiến trình có yêu cầu trong một khoảng thời gian trì hoãn có thể chấp nhận được.
3) Tối ưu hóa sự sử dụng tài nguyên.

➢Để thỏa mãn các mục tiêu kể trên, cần phải giải quyết các vấn đề nảy sinh khi có nhiều tiến trình đồng thời yêu cầu một tài nguyên không thể chia sẻ.

4.2. TIỂU TRÌNH (THREAD)

1. Mô hình tiểu trình (Thread)
2. Ví dụ

4.2. TIỂU TRÌNH (THREAD)

1. Mô hình tiểu trình (Thread)

Vấn đề:

- Thông thường mỗi tiến trình có một không gian địa chỉ và một dòng xử lý.
- Mong muốn có nhiều dòng xử lý cùng chia sẻ một không gian địa chỉ và các dòng xử lý hoạt động song song như các tiến trình độc lập.
- Xuất hiện cơ chế mới cho HĐH gọi là tiểu trình.

- **Thread là một đơn vị xử lý cơ bản**. Mỗi thread xử lý tuần tự đoạn code của nó, **sở hữu một con trỏ lệnh, tập các thanh ghi và một vùng nhớ stack riêng**.
- **Các thread chia sẻ CPU với nhau giống như cách chia sẻ giữa các tiến trình:** một thread chờ xử lý trong khi các thread khác chờ đến lượt. Một thread cũng có thể tạo lập các thread con và nhận các trạng thái khác nhau như một tiến trình thật sự. Một tiến trình có thể sở hữu nhiều thread.

4.2. TIẾU TRÌNH (THREAD)

1. Mô hình tiểu trình (2)

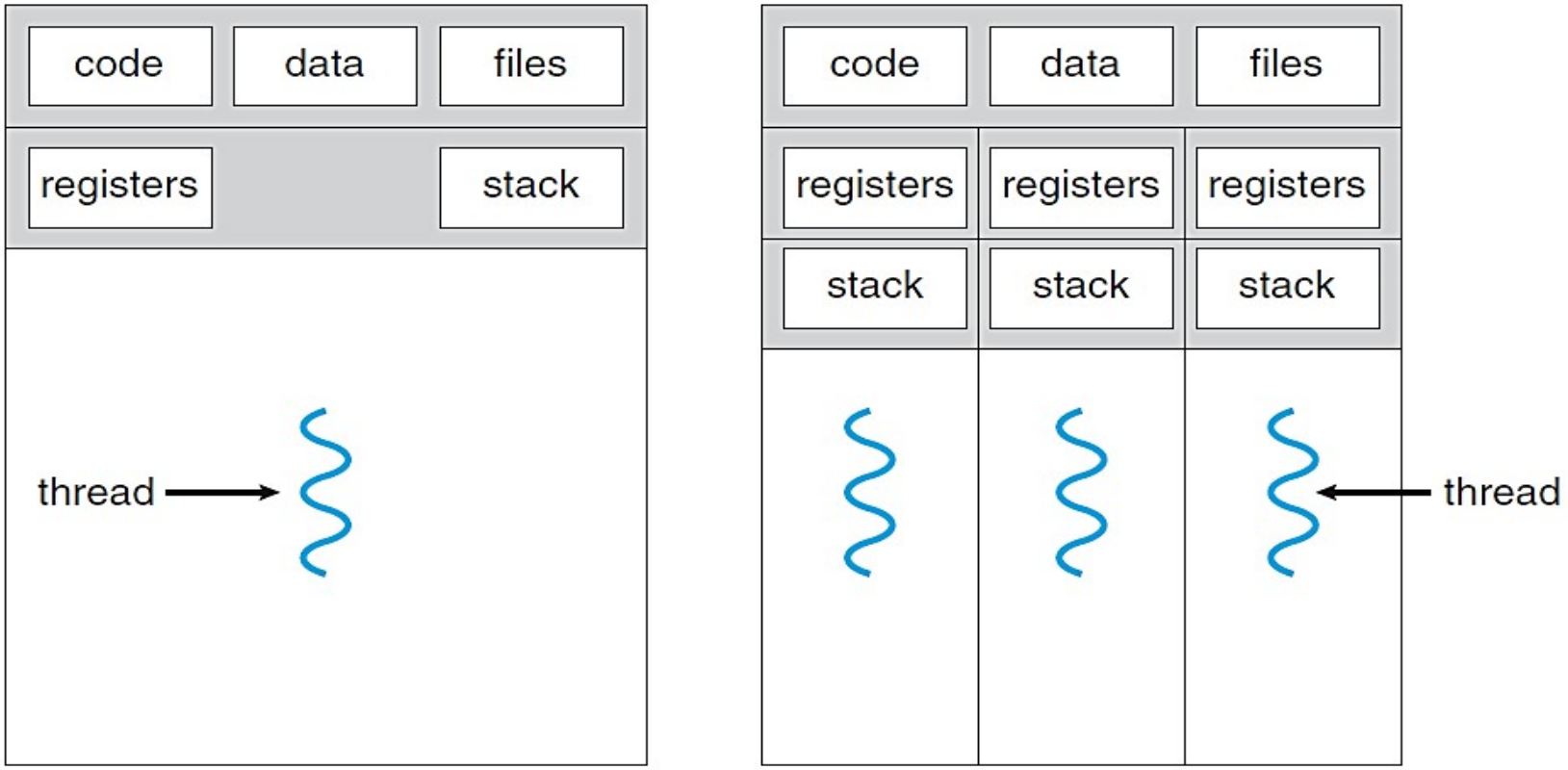
➢Mỗi tiến trình có một **tập tài nguyên** và một **môi trường riêng** (*một con trỏ lệnh, một stack, các thanh ghi và không gian địa chỉ*). Các tiến trình hoàn toàn độc lập với nhau, chỉ có thể liên lạc thông qua các cơ chế thông tin giữa các tiến trình mà HĐH cung cấp.

➢ Ngược lại, các thread trong cùng một tiến trình lại chia sẻ một không gian địa chỉ chung, điều này có nghĩa là các thread chia sẻ các biến toàn cục của tiến trình.

→ **Một thread có thể truy xuất đến tất cả các stack (Heap?) của những thread khác trong cùng một tiến trình (?)**. Cấu trúc này **không đề nghị một cơ chế bảo vệ nào**, và điều này cũng không thật sự cần thiết vì các thread trong cùng một tiến trình thuộc về cùng một sở hữu chủ đã tạo ra chúng trong ý định cho phép chúng hợp tác với nhau.

4.2. TIỂU TRÌNH (THREAD)

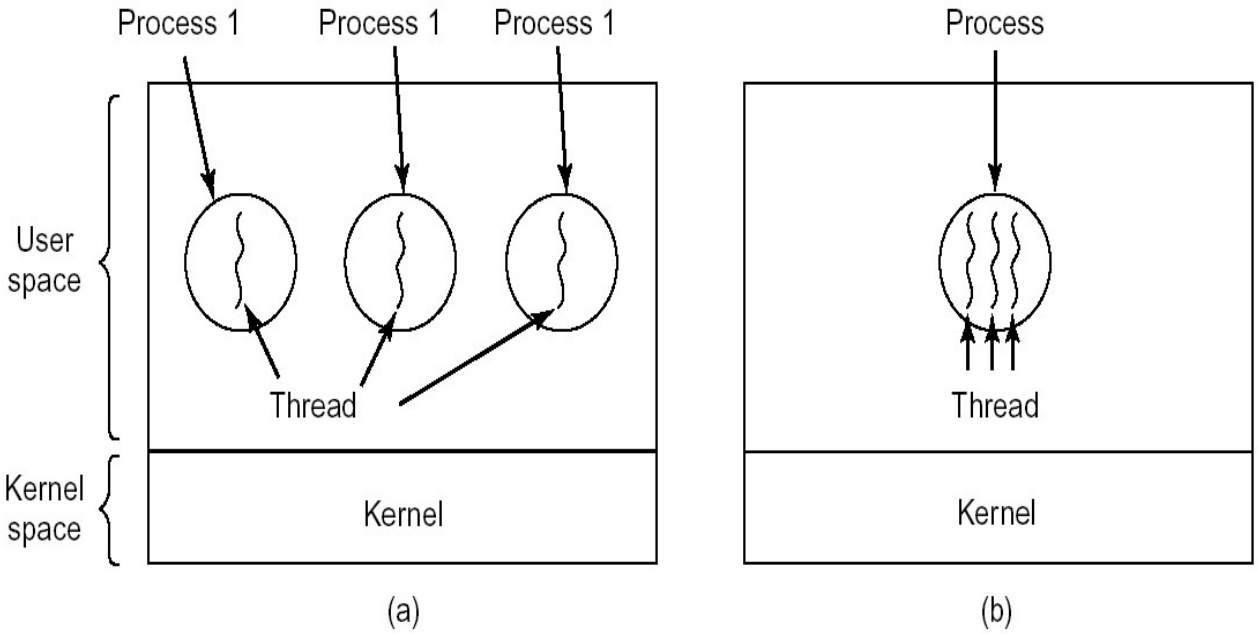
1. Mô hình tiểu trình (3)



Mô hình tiến trình đơn luồng và đa luồng

4.2. TIỂU TRÌNH (THREAD)

1. Mô hình tiểu trình (4)



a) 3 tiến trình với 1 thread b) 1 tiến trình với 3 thread

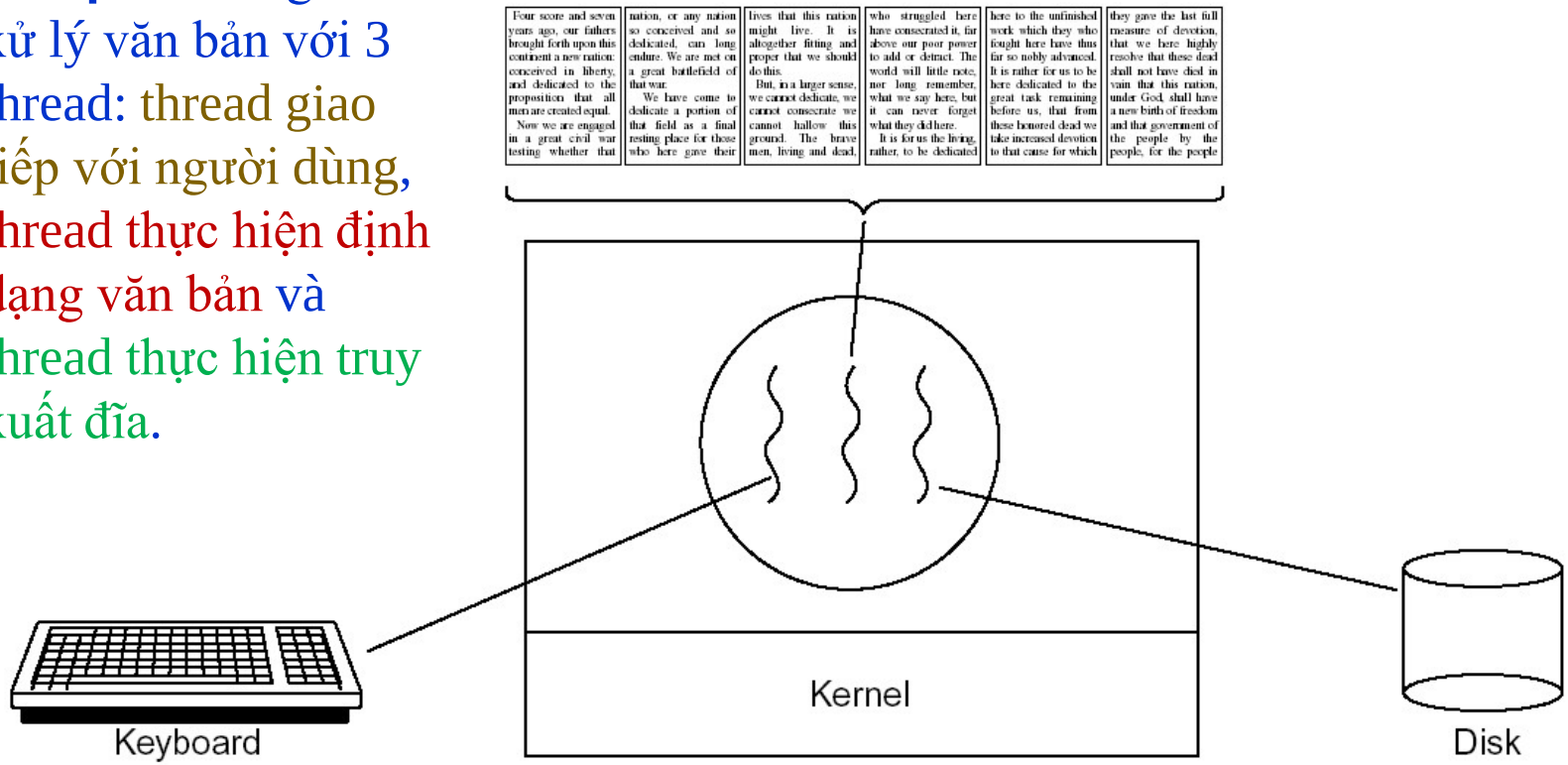
4.2. TIỂU TRÌNH (THREAD)

- **1. Mô hình tiểu trình (5)**

Thông tin chia sẻ giữa các thread trong cùng tiến trình	Các thông tin riêng của thread
Không gian địa chỉ	Bộ đếm chương trình
Các biến toàn cục	Các thanh ghi
Các tập tin mở	Ngăn xếp
Các tiến trình con	Trạng thái
Các cảnh báo	
Các tín hiệu và các bộ xử lý tín hiệu	
Thông tin tài khoản	

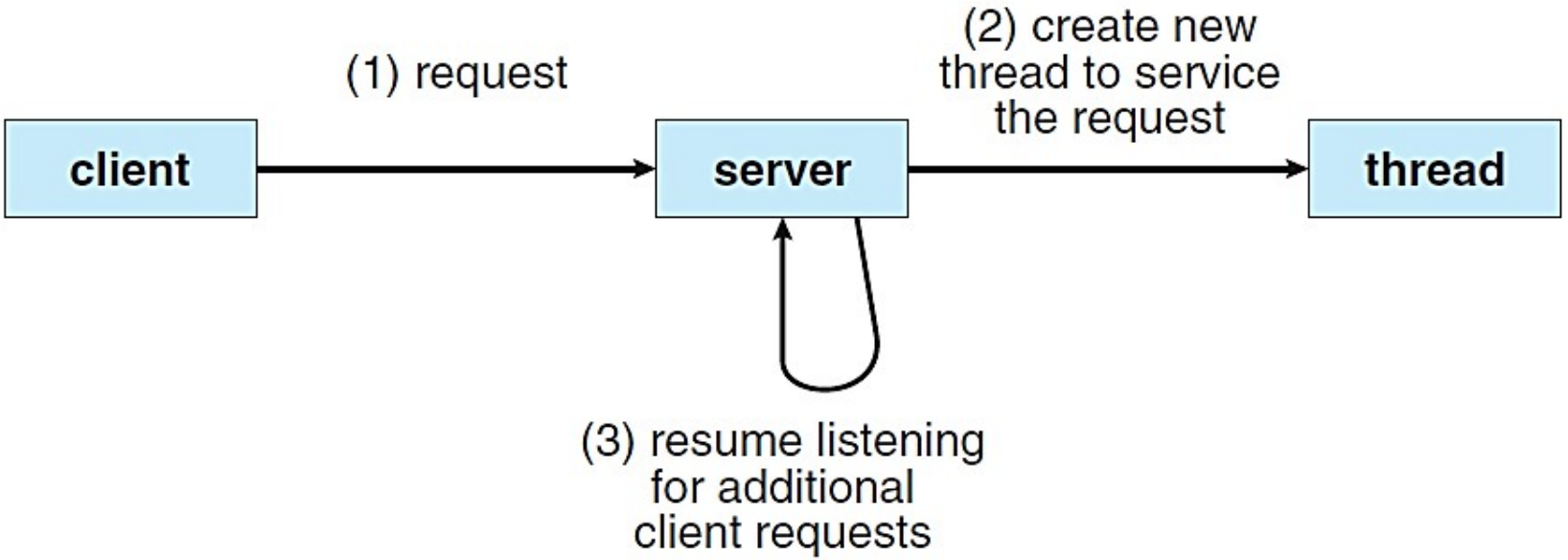
4.2. TIỂU TRÌNH (THREAD)

Ví dụ: Chương trình xử lý văn bản với 3 thread: thread giao tiếp với người dùng, thread thực hiện định dạng văn bản và thread thực hiện truy xuất đĩa.



4.2. TIỂU TRÌNH (THREAD)

Ví dụ: Một tiến trình dịch vụ (file, web, ftp, …) trên máy chủ (server) có thể sử dụng mô hình đa luồng để đáp ứng yêu cầu từ các máy con. Tiến trình gồm: 1 luồng nghe (Listener) chuyên nhận lời yêu cầu từ client, khi yêu cầu được chấp nhận, 1 luồng mới sẽ được tạo ra để phục vụ cho yêu cầu đó.



4.3. ĐIỀU PHỐI TIẾN TRÌNH

5.3. ĐIỀU PHỐI TIẾN TRÌNH

1. Giới thiệu
2. Các chiến lược điều phối

(1) Chiến lược FIFO (First In First Out)

(2) Chiến lược Round Robin

(3) Phân phối với độ ưu tiên (Priority)

(4) Chiến lược SIF (Shortest Job First)

(5) Chiến lược điều phối với nhiều mức độ ưu tiên

4.3. ĐIỀU PHỐI TIẾN TRÌNH

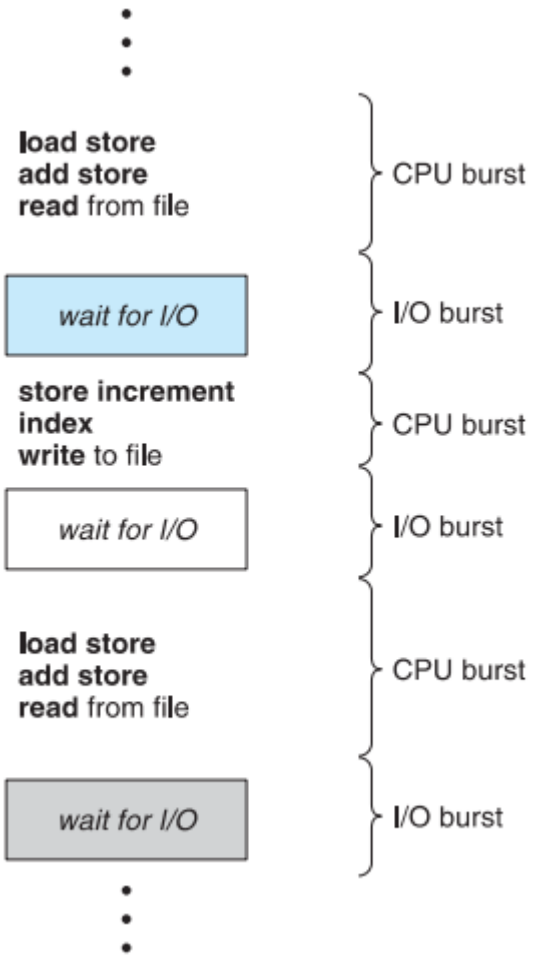
1. GIỚI THIỆU (1)

- Kỹ thuật đa chương giúp việc sử dụng CPU đạt hiệu quả cao nhất.
- **Chu kỳ CPU-I/O**

➢ Sự thực thi của tiến trình bao gồm nhiều chu kỳ CPU-I/O

➢ Một chu kỳ CPU-I/O bao gồm chu kỳ thực thi CPU (CPU burst) và theo sau bởi chu kỳ chờ đợi vào/ra (I/O burst).

➢ Việc phân phối chu kỳ CPU-I/O là một đặt điểm quan trọng để chọn lựa giải thuật định thời phù hợp.



4.3. ĐIỀU PHỐI TIẾN TRÌNH

1. GIỚI THIỆU (2)

Mục tiêu chung:

- Công bằng (Fairness): chia sẻ CPU công bằng, không xảy ra "chờ vô hạn" để được cấp phát CPU.
- Tôn trọng chính sách (Policy enforcement): tuân thủ chính sách đã đặt ra của hệ thống.
- Cân bằng (Balance): cân bằng hoạt động của các bộ phận

4.3. ĐIỀU PHỐI TIẾN TRÌNH

1. GIỚI THIỆU (3)

➢Mục tiêu trong hệ thống theo lô (Batch system)

- Thông lượng (Throughput): cực đại hóa số công việc trong 1 đơn vị thời gian.
  - Thời gian lưu lại trong hệ thống: cực tiểu hóa thời gian hoàn tất 1 tác vụ.
  - Tận dụng CPU: khai thác tối đa, hạn chế thời gian rỗi.

➢Mục tiêu trong hệ thống tương tác (Interactive system)

- Thời gian đáp ứng: cực tiểu hóa thời gian đáp ứng của tác vụ.
- Cân đối: cân đối thời gian đáp ứng của các tác vụ theo yêu cầu của người sử dụng.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

1. GIỚI THIỆU (4)

Đặc tính của tiến trình

- **Tiến trình hướng nhập xuất:** thực hiện nhiều yêu cầu I/O, ít dùng CPU (sử dụng CPU trong những khoảng thời gian rất ngắn).
- **Tiến trình hướng xử lý:** sử dụng CPU nhiều, ít yêu cầu nhập xuất.
- **Tiến trình tương tác hay xử lý theo lô:** tương tác cần hồi đáp tức thời các yêu cầu, xử lý theo lô có thể trì hoãn trong thời gian có thể.
- **Độ ưu tiên của tiến trình:** có thể được phân cấp ưu tiên theo một số tiêu chuẩn nào đó.
- **Thời gian đã sử dụng CPU của tiến trình:** Ưu tiên tiến trình đã sử dụng CPU nhiều/ít tùy theo quan điểm.
- **Thời gian còn lại để hoàn tất:** cho tiến trình cần ít thời gian nhất để hoàn tất thực hiện trước! Tuy nhiên, thực tế ít khi biết được tiến trình cần bao nhiêu tgian nữa để kết thúc.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

1. GIỚI THIỆU (5)

Cơ chế điều phối

Hệ thống máy tính thường có một đồng hồ ngắt giờ (Interval timer / Interrupt clock), sau những khoảng thời gian *t* sẽ phát sinh một ngắt (interrupt), lúc đó quyền điều khiển được trả về cho chương trình xử lý ngắt của hệ điều hành. Có 2 kiểu điều phối:

- **Điều phối không ưu tiên (Non-preemptive)-Độc quyền:** khi CPU được bộ điều phối cấp cho tiến trình, TT sẽ được quyền sử dụng CPU đến khi kết thúc hoặc khi nó chuyển sang trạng thái Blocked. Việc điều phối không được thực hiện khi ngắt đồng hồ được kích hoạt (Tiến trình **độc quyền** sử dụng CPU, quyền điều phối của bộ điều phối **không được ưu tiên**).
- **Điều phối ưu tiên (Preemptive)-Không độc quyền:** khi bộ điều phối cấp CPU cho tiến trình nó sẽ quy định thời gian sử dụng CPU của tiến trình. Khi đồng hồ ngắt được kích hoạt, bộ điều phối sẽ kiểm tra thời gian sử dụng CPU của tiến trình, nếu hết thời gian mà tiến trình chưa giải phóng CPU (kết thúc hoặc chuyển sang trạng thái Blocked) thì tiến trình sẽ được chuyển sang trạng thái Ready, nhường CPU cho tiến trình khác. (Tiến trình **không độc quyền** sử dụng CPU, quyền điều phối của bộ điều phối được **ưu tiên**).

4.3. ĐIỀU PHỐI TIẾN TRÌNH

2. Các chiến lược điều phối

(1) Chiến lược FIFO (First In First Out) : Tiến trình nào được đưa vào danh sách ready trước sẽ được cấp Processor trước. Độc chiếm CPU. Khi CPU được cấp phát cho 1 tiến trình, CPU chỉ được tiến trình tự nguyện giải phóng khi kết thúc xử lý hay khi có 1 yêu cầu I/O.

Ví dụ:

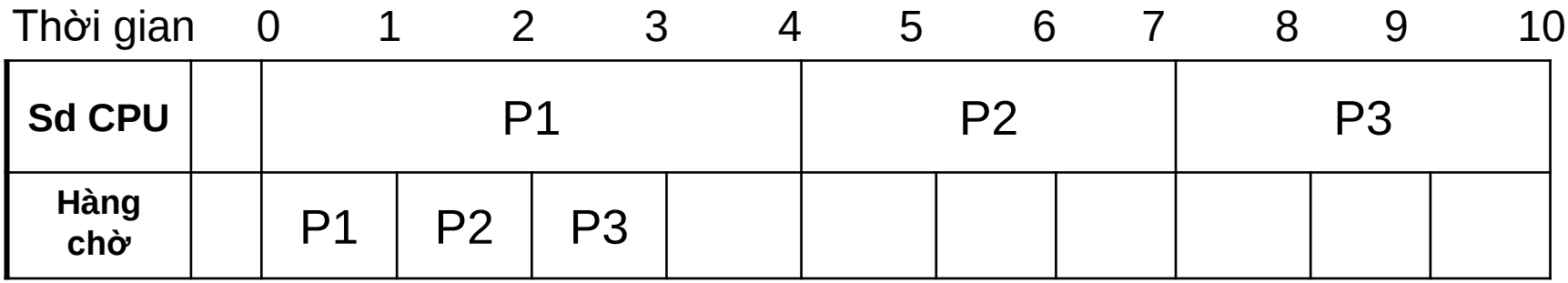
Tiến trình	Thời điểm vào	T/g xử lý
P1	0	4
P2	1	3
P3	2	3

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình



P1P1P1P1P2P2P2P3P3P3

b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 4 – 0 = 4, P2: 7 – 1= 6, P3: 10 – 2 = 8.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = thời gian không sử dụng CPU

P1: 0 – 0 = 0, P2: 4 – 1= 3, P3: 7 – 2 = 5.

Thời gian chờ đợi trung bình của các tiến trình: (0 + 3 + 5)/3 = 2.67

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(1) Chiến lược FIFO (First In First Out) (3)

Ưu & khuyết điểm:

- Đơn giản, dễ triển khai.
- Không phù hợp với hệ thống tương tác do CPU không được cấp phát đều đặn cho các tiến trình.
- Xảy ra hiện tượng tích lũy thời gian chờ, các tiến trình ngắn phải đợi tiến trình có yêu cầu CPU thời gian dài kết thúc xử lý.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(2) Chiến lược Round Robin

- Danh sách sẵn sàng được xử lý như một danh sách vòng, bộ điều phối lần lượt cấp phát cho từng tiến trình trong danh sách một khoảng thời gian sử dụng CPU gọi là **quantum**.
- Đây là một giải thuật điều phối preemptive: khi một tiến trình sử dụng CPU đến hết thời gian quantum dành cho nó, HĐH thu hồi CPU và cấp cho tiến trình kế tiếp trong danh sách.
- Nếu tiến trình bị khóa hay kết thúc trước khi sử dụng hết thời gian quantum, HĐH cũng lập tức cấp phát CPU cho tiến trình khác. Khi tiến trình tiêu thụ hết thời gian CPU dành cho nó mà chưa hoàn tất, tiến trình được đưa trở lại vào cuối danh sách sẵn sàng để đợi được cấp CPU trong lượt kế tiếp.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(2) Chiến lược Round Robin (2)

Ví dụ

Tiến trình	Thời điểm vào	T/g xử lý
P1	0	4
P2	1	3
P3	2	3
Quantum=2		

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình

Thời gian	0	1	2	3	4	5	6	7	8	9	10
Sd CPU		P1		P2		P3		P1		P2	P3
Hàng chờ		P1	P2	P3/P1		P2		P3			

P1P1P2P2P3P3P1P1P2P3

b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 8 – 0 = 8, P2: 9 – 1= 8, P3: 10 – 2 = 8.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = TG không sử dụng CPU

P1= 0+ (6-2) = 4, P2= (2-1)+(8-4)= 5, P3= (4-2)+ (9-6) =5

Thời gian chờ đợi trung bình của các tiến trình: (4 + 5 + 5)/3 = 4.67

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(2) Chiến lược Round Robin (4)

Ưu & khuyết điểm:

- Phù hợp với hệ thống chia sẻ thời gian.
- Khó trong việc xác định quantum hợp lý: nếu quantum quá dài thì trở thành giải thuật FIFO, nếu quantum quá bé sẽ phát sinh nhiều thời gian chuyển đổi ngữ cảnh giữa các tiến trình làm cho việc sử dụng CPU kém hiệu quả.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(3) Chiến lược điều phối với độ ưu tiên (Priority-scheduling algorithm)

- Nguyên tắc:

1) Mỗi tiến trình được gán cho một độ ưu tiên
2) Tiến trình có độ ưu tiên cao được cấp phát CPU trước
3) Các tiến trình có độ ưu tiên bằng nhau sẽ được định thời theo chiến thuật FIFO.
4) Độ ưu tiên có thể được định nghĩa bên trong hay bên ngoài. →Định nghĩa bên trong thường dùng những định lượng có thể đo để tính toán độ ưu tiên (các giới hạn tgian, yêu cầu bộ nhớ, số lượng tập tin đang mở và tỉ lệ của chu kỳ I/O trung bình với tỉ lệ của chu kỳ CPU trung bình,…) → Định nghĩa bên ngoài được thiết lập bởi các tiêu chuẩn bên ngoài đối với HĐH (như sự quan trọng của quá trình, loại và lượng chi phí đang được trả cho việc dùng máy tính,…)
5) Có thể theo nguyên tắc đặc quyền hay không đặc quyền.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(3) Chiến lược điều phối với độ ưu tiên (2)

- Điều phối với độ ưu tiên và không đặc quyền sẽ thu hồi CPU từ tiến trình hiện hành để cấp cho tiến trình mới vào nếu độ ưu tiên của tiến trình mới vào cao hơn
- Điều phối với độ ưu tiên và đặc quyền sẽ chỉ chèn tiến trình mới vào danh sách Ready tại vị trí phù hợp

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(3) Chiến lược điều phối với độ ưu tiên (3)

Ví dụ: điều phối với độ ưu tiên theo chế độ **đặc quyền**

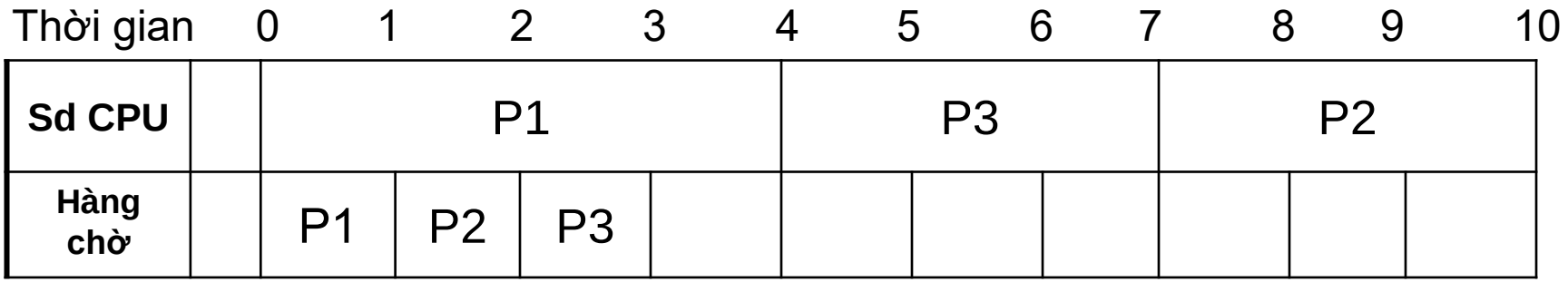
Tiến trình	Thời điểm vào	Độ ưu tiên	t/g xử lý
P1	0	1	4
P2	1	3	3
P3	2	2	3

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình



b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 4 – 0 = 4, P2: 10 – 1= 9 P3: 7 – 2 = 5.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = TG không sử dụng CPU

P1: 0 – 0 = 0, P2: 7 – 1= 6, P3: 4 – 2 = 2.

Thời gian chờ đợi trung bình của các tiến trình: (0 + 6 + 2)/3 = 2.67

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(3) Chiến lược điều phối với độ ưu tiên (5)

Ví dụ: điều phối với độ ưu tiên theo chế độ **không đặc quyền (Tiến trình vào sau có độ ưu tiên hơn tiến trình đang chạy, nó sẽ chiếm CPU).**

Tiến trình	Thời điểm vào	Độ ưu tiên	t/g xử lý
P1	0	3	4
P2	1	2	3
P3	2	1	3

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình

Thời gian	0	1	2	3	4	5	6	7	8	9	10
Sd CPU		P1	P2	P3			P2		P1		
Hàng chờ		P1	P2/ P1	P3/ P2/ P1							

b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 10 – 0 = 10, P2: 7 – 1= 6, P3: 5 – 2 = 3.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = TG không sử dụng CPU

P1: 0 + (7-1) = 6, P2: (1-1) +(5-2)= 3, P3: 0 .

Thời gian chờ đợi trung bình của các tiến trình: (6 + 3 + 0)/3 = 3

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(3) Chiến lược điều phối với độ ưu tiên (7)

Nhược điểm: Tiến trình có độ ưu tiên thấp dễ rơi vào trạng thái chờ vô hạn (nghẽn không định hạn – intefinite blocking hay đói CPU – starvation)

→Cần tăng độ ưu tiên của tiến trình sau mỗi lần cấp phát processor

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(4) Chiến lược điều phối SJF (shortest job first - SJF)

- Đây là một trường hợp đặc biệt của giải thuật điều phối với độ ưu tiên được gán cho mỗi tiến trình là nghịch đảo của thời gian xử lý t ($p = 1/t$) → Ưu tiên các tiến trình yêu cầu ít thời gian
- Giải thuật này cũng có thể theo nguyên tắc preemptive hay non-preemptive.
- Sự chọn lựa xảy ra khi có một tiến trình mới được đưa vào danh sách sẵn sàng trong khi một tiến trình khác đang xử lý. Nếu thời gian xử lý của tiến trình mới ngắn hơn thời gian yêu cầu xử lý còn lại của tiến trình hiện hành, giải thuật SJF preemptive sẽ dừng hoạt động của tiến trình hiện hành, trong khi giải thuật non-preemptive sẽ cho phép tiến trình hiện hành tiếp tục xử lý.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(4) Chiến lược điều phối SJF (2)

Ví dụ: điều phối với SJF theo chế độ đặc quyền (non-preemptive)

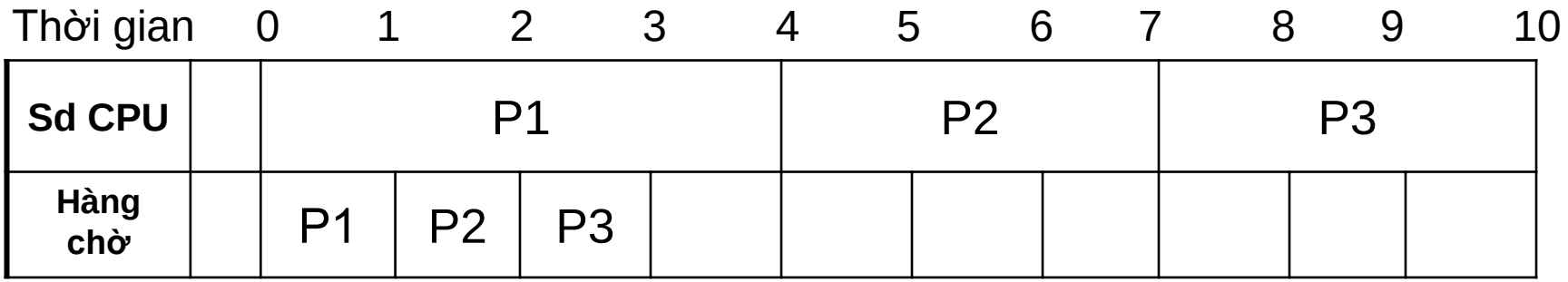
Tiến trình	Thời điểm vào	t/g xử lý
P1	0	4
P2	1	3
P3	2	3

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình



b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 4 – 0 = 4, P2: 7 – 1= 6, P3: 10 – 2 = 8.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = TGBĐSD CPU – TG bắt đầu vào

P1: 0 – 0 = 0, P2: 4 – 1= 3, P3: 7 – 2 = 5.

Thời gian chờ đợi trung bình của các tiến trình: (0 + 3 + 5)/3 = 2.67

5.3. ĐIỀU PHỐI TIẾN TRÌNH

(4) Chiến lược điều phối với SJF (4)

Ví dụ: điều phối với SJF theo chế độ không đặc quyền (Reemptive).

Tiến trình	Thời điểm vào	t/g xử lý
P1	0	4
P2	1	2
P3	2	2

Yêu cầu:

a. Xác định kết quả điều phối hoạt động của các tiến trình.
b. Tính thời gian lưu lại hệ thống của các tiến trình.
c. Tính thời gian chờ trong hệ thống của từng tiến trình.

5.3. ĐIỀU PHỐI TIẾN TRÌNH

a. Kết quả điều phối hoạt động của từng tiến trình

Thời gian	0	1	2	3	4	5	6	7	8	9
Sd CPU		P1	P2	P2	P3	P3	P1	P1	P1	
Hàng chờ		P1	P2/P1	P3/P2						

b. Thời gian lưu lại hệ thống của từng tiến trình:

TGLLHT = TGSD xong CPU – TG bắt đầu vào

P1: 8 – 0 = 9, P2: 3 – 1= 2, P3: 5 – 2 = 3.

c. Thời gian chờ trong hệ thống của từng tiến trình:

TGCHT = Tg không sử dụng CPU

P1: 0+(5 – 1) = 4, P2: (1-1)= 0, P3: (3-2)=1.

Thời gian chờ đợi trung bình của các tiến trình: (4 + 0 + 1)/3 = 1.66

4.3. ĐIỀU PHỐI TIẾN TRÌNH

➢Ưu và nhược điểm:

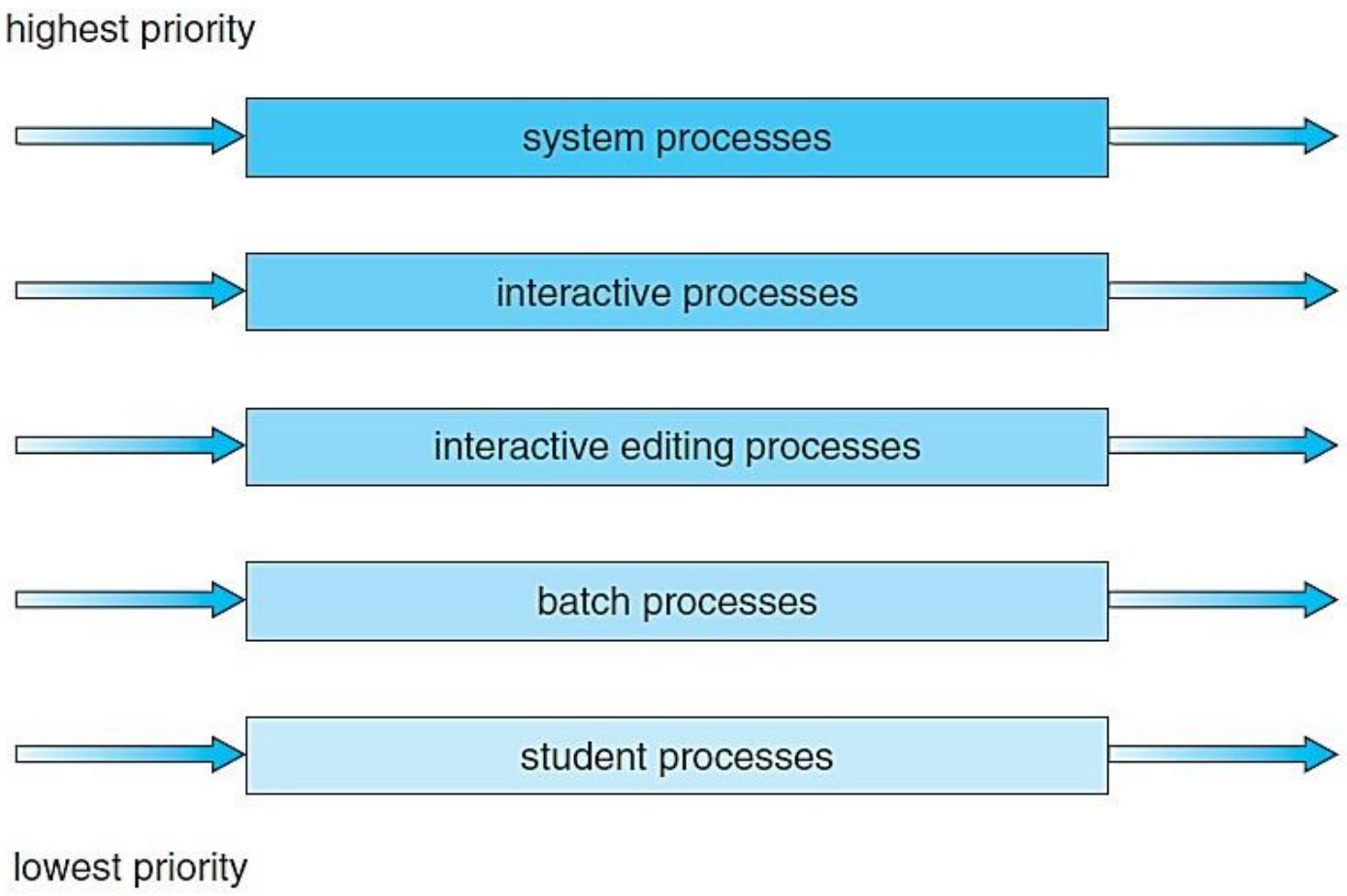
- **G**iải thuật này cho phép đạt được thời gian chờ trung bình cực tiểu.
- Khó khăn lớn nhất của giải thuật là xác định thời gian yêu cầu xử lý còn lại của tiến trình → Thường phải thực hiện dự đoán thông qua các khoảng thời gian đã sử dụng CPU trước kia của tiến trình.

4.3. ĐIỀU PHỐI TIẾN TRÌNH

(5) Chiến lược điều phối với nhiều mức độ ưu tiên

- Phân lớp các tiến trình tùy theo độ ưu tiên
- Mỗi danh sách bao gồm các tiến trình có cùng độ ưu tiên và được áp dụng một giải thuật điều phối trong nhóm
- Điều phối giữa các nhóm bằng giải thuật độc quyền (preemptive) và sử dụng độ ưu tiên cố định
- Một tiến trình thuộc về danh sách ở cấp ưu tiên *i* sẽ chỉ được cấp phát CPU khi các danh sách ở cấp ưu tiên lớn hơn *i* đã trống

4.3. ĐIỀU PHỐI TIẾN TRÌNH



4.3. ĐIỀU PHỐI TIẾN TRÌNH

- **Thảo luận: Có thể xảy ra tình trạng đói CPU của các tiến trình ở danh sách có độ ưu tiên thấp. Giải pháp xử lý?**

➢**Có thể xây dựng giải thuật điều phối nhiều cấp ưu tiên và xoay vòng:** chuyển dần một tiến trình từ danh sách có độ ưu tiên cao xuống danh sách có độ ưu tiên thấp hơn sau mỗi lần sử dụng CPU. Tương tự, một tiến trình chờ quá lâu trong các danh sách có độ ưu tiên thấp cũng có thể được chuyển dần lên các danh sách có độ ưu tiên cao hơn.

➢Khi xây dựng một giải thuật điều phối nhiều cấp ưu tiên và xoay vòng cần quyết định các tham số nào?

- Số lượng các cấp ưu tiên.
- Giải thuật điều phối cho từng danh sách ứng với một cấp ưu tiên.
- Phương pháp xác định thời điểm di chuyển một tiến trình vào danh sách có độ ưu tiên cao hơn.
- Phương pháp xác định thời điểm di chuyển một tiến trình vào danh sách có độ ưu tiên thấp hơn.
- Phương pháp sử dụng để xác đinh một tiến trình mới được đưa vào hệ thống sẽ thuộc danh sách ứng với độ ưu tiên nào.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu (1)

❖Nhu cầu đồng bộ hóa

Trong một hệ thống cho phép các tiến trình liên lạc với nhau, HĐH cần cung cấp kèm theo những cơ chế đồng bộ hóa vì các lý do sau đây:

(1)Yêu cầu độc quyền truy xuất (Mutual exclusion)

- Các tài nguyên trong hệ thống được phân thành hai loại: tài nguyên có thể chia sẻ (cho phép nhiều tiến trình đồng thời truy xuất) và các tài nguyên không thể chia sẻ (chỉ chấp nhận một hay một số lượng hạn chế tiến trình sử dụng tại một thời điểm).
- Tính không chia sẻ của tài nguyên thường có nguồn gốc từ một trong hai nguyên nhân sau đây:
  - Đặc tính cấu tạo phần cứng của tài nguyên không cho phép chia sẻ.
  - Nếu nhiều tiến trình sử dụng tài nguyên đồng thời, có nguy cơ xảy ra các kết quả không dự đoán được do hoạt động của các tiến trình trên tài nguyên ảnh hưởng lẫn nhau.

➢Để giải quyết vấn đề, cần bảo đảm tiến trình độc quyền truy xuất tài nguyên, nghĩa là hệ thống phải kiểm soát sao cho tại một thời điểm, chỉ có một tiến trình được quyền truy xuất một tài nguyên không thể chia sẻ.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu

(1) Nhu cầu đồng bộ hóa

(2) Yêu cầu độc quyền truy xuất (Mutual exclusion)

(3) Yêu cầu phối hợp (Synchronization)

(4) Miền tranh chấp (Critical Section)

1. Các giải pháp

(1) Sử dụng biến cờ hiệu

(2) Sử dụng biến kiểm tra luân phiên

(3) Giải pháp của Peterson

(4) Cấm ngắt

(5) Chỉ thị TSL (Test-and-Set)

(6) Semaphore (Đèn hiệu)

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu (2)

(2) Yêu cầu phối hợp (Synchronization)

- Nhìn chung, mối tương quan về tốc độ thực hiện của hai tiến trình trong hệ thống là không thể biết trước, vì điều này phụ thuộc vào nhiều yếu tố động như tần suất xảy ra các ngắt của từng tiến trình, thời gian tiến trình được cấp phát CPU, …

→ Các tiến trình hoạt động không đồng bộ với nhau.

- Nhưng có những tình huống các tiến trình cần hợp tác trong việc hoàn thành tác vụ → cần phải đồng bộ hóa hoạt động của các tiến trình.

Ví dụ: một tiến trình chỉ có thể xử lý nếu một tiến trình khác đã kết thúc một công việc nào đó.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu (2)

(3) Miền tranh chấp (Critical Section)

- Đoạn chương trình trong đó có khả năng xảy ra các mâu thuẫn truy xuất trên tài nguyên chung được gọi là miền tranh chấp (Critical Section).

Ví dụ: giả sử có hai tiến trình P1 và P2 thực hiện công việc của các kế toán, và cùng chia sẻ một vùng nhớ chung lưu trữ biến **taikhoan** phản ánh thông tin về tài khoản. Mỗi tiến trình muốn rút một khoản tiền **tienrut** từ tài khoản:

```
if (taikhoan – tienrut >=0)  taikhoan = taikhoan - tienrut ;
else   error(”Không thể rút tiền”);
```

Giả sử trong tài khoản hiện còn 800, P1 muốn rút 500 và P2 muốn rút 400. Mô tảtình huống tranh chấp có thể xảu ra?

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu (3)

(3) Miền tranh chấp (Critical Section)

Giả sử tình huống xảy ra như sau:

- Sau khi đã kiểm tra điều kiện (taikhoan - tienrut>=0) và nhận kết quả là True, P1 hết thời gian xử lý mà hệ thống cho phép, hệ điều hành cấp phát CPU cho P2.
- P2 kiểm tra cùng điều kiện trên và cũng nhận được kết quả là True (do P1 vẫn chưa rút tiền) và rút 400. Giá trị của taikhoan được cập nhật lại là 400.
- Khi P1 được tái kích hoạt và tiếp tục xử lý, nó thực hiện tiếp câu lệnh rút tiền và cập nhật lại taikhoan là -100 -> Tình huống lỗi xảy ra.

- Các tình huống tương tự như thế có thể xảy ra khi có nhiều tiến trình đọc và ghi dữ liệu trên vùng nhớ chung, và kết quả phụ thuộc vào sự điều phối tiến trình của hệ thống – **được gọi là các tình huống tranh đoạt điều khiển (race condition).**
- Trong ví dụ, trên đoạn mã: if (taikhoan – tienrut >=0) taikhoan = taikhoan - tienrut ; của mỗi tiến trình tạo thành một miền tranh chấp.

- **Giải quyết vấn đề:** ???

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

1. Giới thiệu (4)

(3) *Miền tranh chấp (Critical Section)*

Một phương pháp giải quyết tốt bài toán miền tranh chấp cần thỏa mãn 4 điều kiện sau:

- Không có hai tiến trình cùng ở trong miền tranh chấp cùng một lúc.
- Không có giả thiết nào đặt ra cho sự liên hệ về tốc độ của các tiến trình, cũng như về số lượng bộ xử lý trong hệ thống. (????)
- Một tiến trình tạm dừng bên ngoài miền tranh chấp không được ngăn cản các tiến trình khác vào miền tranh chấp.
- Không có tiến trình nào phải chờ vô hạn để được vào miền tranh chấp.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

2. Các giải pháp (1)

(1) Sử dụng biến cờ hiệu

- Biến lock (chốt cửa) được các tiến trình chia sẻ là một biến chung, biến này được khởi động là 0.
- Một tiến trình muốn vào miền tranh chấp trước tiên phải kiểm tra giá trị của biến lock. Nếu lock=0, tiến trình đặt lại giá trị cho lock=1 và đi vào miền tranh chấp. Nếu lock đang nhận giá trị 1, tiến trình phải chờ bên ngoài miền tranh chấp cho đến khi lock có giá trị 0.
- Như vậy giá trị 0 của lock mang ý nghĩa là không có tiến trình nào đang ở trong miền tranh chấp, và lock=1 khi có một tiến trình đang ở trong miền tranh chấp.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

Ví dụ: Cấu trúc một chương trình sử dụng lock để đồng bộ:

```
 while (TRUE)
{ while (lock==1) ; //wait
        lock=1;
                critical-section();
        lock=0;
                noncritical-section();          }
```

- Giải pháp này có thể vi phạm điều kiện thứ nhất: hai tiến trình có thể cùng ở trong miền tranh chấp tại một thời điểm.
- Giả sử một tiến trình nhận thấy lock=0 và chuẩn bị vào miền tranh chấp, nhưng trước khi nó có thể đặt lại giá trị cho lock là 1, nó bị tạm dừng để một tiến trình khác hoạt động.
- Tiến trình thứ hai này thấy lock vẫn là 0 thì vào miền tranh chấp và đặt lại lock=1. Sau đó tiến trình thứ nhất được tái kích hoạt, nó gán lock=1 lần nữa rồi vào miền tranh chấp. Như vậy tại thời điểm đó cả hai tiến trình đều ở trong miền tranh chấp.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(2) Sử dụng biến kiểm tra luân phiên

- Đây là một giải pháp đề nghị cho hai tiến trình. Hai tiến trình này sử dụng chung biến **turn** (phản ánh tiến trình nào được vào miền tranh chấp) được khởi động với giá trị 0.
- **Nếu turn=0**, tiến trình A được vào miền tranh chấp.
- **Nếu turn=1**, tiến trình A đi vào một vòng lặp chờ đến khi turn nhận giá trị 0.
- **Khi tiến trình A rời khỏi miền tranh chấp, nó đặt giá trị turn về 1** để **cho phép tiến trình B đi vào miền tranh chấp**.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

Ví dụ: *Sử dụng biến kiểm tra luân phiên*

Cấu trúc tiến trình A:

```
 while (TRUE)
{ while (turn !=0) ; //wait
        critical-section();
        turn=1;
        noncritical-section();
}
```

Cấu trúc tiến trình B:

```
 while (TRUE)
{ while (turn !=1) ; //wait
        critical-section();
        turn=0;
        noncritical-section();
}
```

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(2) Sử dụng biến kiểm tra luân phiên (3)

- Giải pháp này dựa trên việc thực hiện sự kiểm tra nghiêm ngặt đến lượt tiến trình nào được vào miền tranh chấp. Do đó có thể ngăn chặn tình trạng cả hai tiến trình cùng vào một lúc.
- Nhưng lại có thể vi phạm điều kiện thứ ba: *một tiến trình có thể bị ngăn chặn vào miền tranh chấp bởi một tiến trình khác không ở trong miền tranh chấp.*

Giả sử tiến trình B ra khỏi miền tranh chấp rất nhanh chóng. Cả hai tiến trình đều ở ngoài miền tranh chấp và turn=0. Tiến trình A vào miền tranh chấp và ra khỏi nhanh chóng, đặt lại giá trị của turn là 1, rồi lại xử lý đoạn lệnh ngoài miền tranh chấp lần nữa. Sau đó, tiến trình A lại kết thúc nhanh chóng đoạn lệnh ngoài miền tranh chấp của nó và muốn vào lại một lần nữa. Tuy nhiên, lúc này B vẫn còn mãi xử lý đoạn lệnh ngoài miền tranh chấp của mình, và turn lại mang giá trị 1.

➢ ***Như vậy, giải pháp này không có giá trị khi có sự khác biệt lớn về tốc độ thực hiện của hai tiến trình, nó vi phạm cả điều kiện thứ hai.***

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(3) Giải pháp của Peterson

➢Peterson đưa ra giải pháp kết hợp ý tưởng của cả hai giải pháp trên. Các tiến trình chia sẻ hai biến chung:

- int turn; //đến phiên ai
- int interesse[2]; //khởi động là FALSE

- Nếu interesse[i]=TRUE có nghĩa là tiến trình Pi muốn vào miền tranh chấp.
- Khởi đầu, interesse[0]=interesse[1]=FALSE và giá trị của turn được khởi động là 0 hay 1.
- Để có thể vào được miền tranh chấp, trước tiên tiến trình Pi đặt giá trị interesse[i]=TRUE (xác định rằng tiến trình muốn vào miền tranh chấp), sau đó đặt turn=j (đề nghị thử tiến trình khác vào miền tranh chấp). Nếu tiến trình Pj không quan tâm đến việc vào miền tranh chấp (interesse[j]=FALSE), thì Pi có thể vào miền tranh chấp, nếu không, Pi phải chờ đến khi interesse[j]=FALSE.
- Khi tiến trình Pi rời khỏi miền tranh chấp, nó lại đặt lại giá trị cho interesse[i]=FALSE.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

Ví dụ: ***Giải pháp của Peterson***

Cấu trúc của tiến trình Pi trong giải pháp Peterson:

```
 while (TRUE)
{
        int j=1-i;     //j là tiến trình còn lại
        interesse[i]=TRUE;
        turn=j;
        while (turn ==j && interesse[j]==TRUE) ; //wait
                 critical-section();
                interesse[i]=FALSE;
                noncritical-section();          }
```

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(3) Giải pháp của Peterson (3)

- Giải pháp này ngăn chặn được tình trạng mâu thuẫn truy xuất: mỗi tiến trình Pi chỉ có thể vào miền tranh chấp khi interesse[j]=FALSE hoặc turn=i.
- Nếu cả hai tiến trình đều muốn vào miền tranh chấp thì interesse[i]=interesse[j]=TRUE nhưng giá trị của turn chỉ có thể hoặc là 0 hoặc là 1, do vậy chỉ có một tiến trình được vào miền tranh chấp

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(4) Cấm ngắt

- Phần cứng cho phép tiến trình cấm tất cả các ngắt trước khi vào miền tranh chấp và phục hồi ngắt khi ra khỏi miền tranh chấp. Khi đó, ngắt đồng hồ cũng không xảy ra, do vậy hệ thống không thể tạm dừng hoạt động của tiến trình đang xử lý để cấp phát CPU cho tiến trình khác, nhờ đó tiến trình hiện hành yên tâm thao tác trên miền tranh chấp mà không sợ bị tiến trình nào khác tranh chấp.
- Giải pháp này không được ưa chuộng vì rất thiếu thận trọng khi cho phép tiến trình người dùng được phép thực hiện lệnh cấm ngắt. Hơn nữa, nếu hệ thống có nhiều bộ xử lý, lệnh cấm ngắt chỉ có tác dụng trên bộ xử lý đang xử lý tiến trình, còn các tiến trình hoạt động trên các bộ xử lý khác vẫn có thể vào được miền tranh chấp.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(5) Chỉ thị TSL (Test-and-Set)

- Giải pháp đòi hỏi sự trợ giúp của cơ chế phần cứng. Nhiều máy tính cung cấp một chỉ thị đặc biệt cho phép kiểm tra và cập nhật nội dung một vùng nhớ trong một thao tác không thể phân chia, gọi là chỉ thị Test-and-Set Lock (TSL) và được định nghĩa như sau:

```
Test-and-Setlock (boolean target)
        {
                Test-and-Setlock = target;
                Target = TRUE;
        }
```

- Nếu có hai chỉ thị TSL xử lý đồng thời (trên hai bộ xử lý khác nhau), chúng sẽ được xử lý tuần tự.
- Có thể cài đặt giải pháp truy xuất độc quyền với TSL bằng cách sử dụng thêm một biến lock, được khởi gán là FALSE. Tiến trình phải kiểm tra giá trị của biến lock trước khi vào miền tranh chấp, nếu lock=FALSE, tiến trình có thể vào miền tranh chấp.

5.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

Ví dụ: ***Chỉ thị TSL (Test-and-Set)***

Cấu trúc một chương trình trong giải pháp TSL:

```
while (TRUE)        {
        while (Test-and-Setlock(lock));
                critical-section();
        lock = FALSE;
        Noncritical-section();     }
```

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

- Cũng giống như các giải pháp phần cứng khác, chỉ thị TSL chỉ giảm nhẹ công việc lập trình để giải quyết vấn đề, nhưng lại không dễ dàng để cài đặt chỉ thị TSL sao cho được xử lý một cách không thể phân chia, nhất là trên máy với cấu hình nhiều bộ xử lý.

➢ *Tất cả các giải pháp đã trình bày đều phải thực hiện một vòng lặp để kiểm tra liệu tiến trình có được phép vào miền tranh chấp.*

➢ *Nếu điều kiện chưa cho phép, tiến trình phải chờ tiếp tục trong vòng lặp kiểm tra này.*

➢ *Các giải pháp buộc tiến trình phải liên tục kiểm tra điều kiện để phát hiện thời điểm thích hợp để vào miền tranh chấp như thế được gọi là các giải pháp **"busy waiting"**. Lưu ý rằng việc kiểm tra như thế tiêu thụ rất nhiều thời gian sử dụng CPU, do vậy tiến trình đang chờ vẫn chiếm dụng CPU. Xu hướng giải quyết vấn đề đồng bộ hóa là nên tránh các giải pháp "busy waiting".*

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(6) Semaphore (Được Dijkstra đề xuất vào năm 1965)

Một semaphore **S** là một biến có các thuộc tính sau:

- Một giá trị nguyên dương e(s).
- Một hàng đợi f(s) lưu danh sách các tiến trình đang bị khóa (chờ) trên semaphore s.

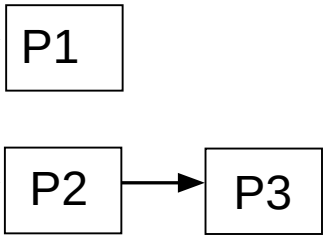
Chỉ có hai thao tác được định nghĩa trên semaphore

- **Down(s) :** giảm giá trị của semaphore s đi 1 đơn vị. Nếu semaphore có trị e(s)>0 thì tiếp tục xử lý. Ngược lại, nếu e(s) ≤ 0, tiến trình phải chờ cho đến khi e(s)>0.
- **Up(s) :** tăng giá trị của semaphore s lên 1 đơn vị. Nếu có một hoặc nhiều tiến trình đang chờ trên semaphore s, bị khóa bởi thao tác Down, thì hệ thống sẽ chọn một trong các tiến trình này để kết thúc thao tác Down và cho tiếp tục xử lý.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

Ví dụ: Semaphore

Semaphore	e(s)	f(s)
S1	-1	
S2	0	Null
S3	-2	



Cài đặt: gọi p là tiến trình thực hiện thao tác Down(s) hay Up(s).

Down(s)

```
        e(s) = e(s) – 1;
        if e(s) <0 { status(P) = blocked;   enter(P, f(s)); }
```

Up(s)

```
        e(s) = e(s) + 1;
         if e(s) <= 0 {  exit(Q, f(s));
        // Q là tiến trình đang chờ trên s status(Q) = ready;
         enter(Q, ready-list); }
```

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(6) Semaphore (3)

Lưu ý:

- Cài đặt này có thể đưa đến một **giá trị âm** cho semaphore, khi đó trị tuyệt đối của semaphore cho biết số (thứ tự ?)tiến trình đang chờ trên semaphore.
- Điều quan trọng là các thao tác này cần thực hiện một cách không bị chia tách, không bị ngắt nửa chừng, có nghĩa là không một tiến trình nào được phép truy xuất đến semaphore nếu tiến trình đang thao tác trên semaphore này chưa kết thúc xử lý hay chuyển sang trạng thái blocked.

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(6) Semaphore (4)

Sử dụng: có thể dùng semaphore để giải quyết vấn đề truy xuất độc quyền hay tổ chức phối hợp giữa các tiến trình.

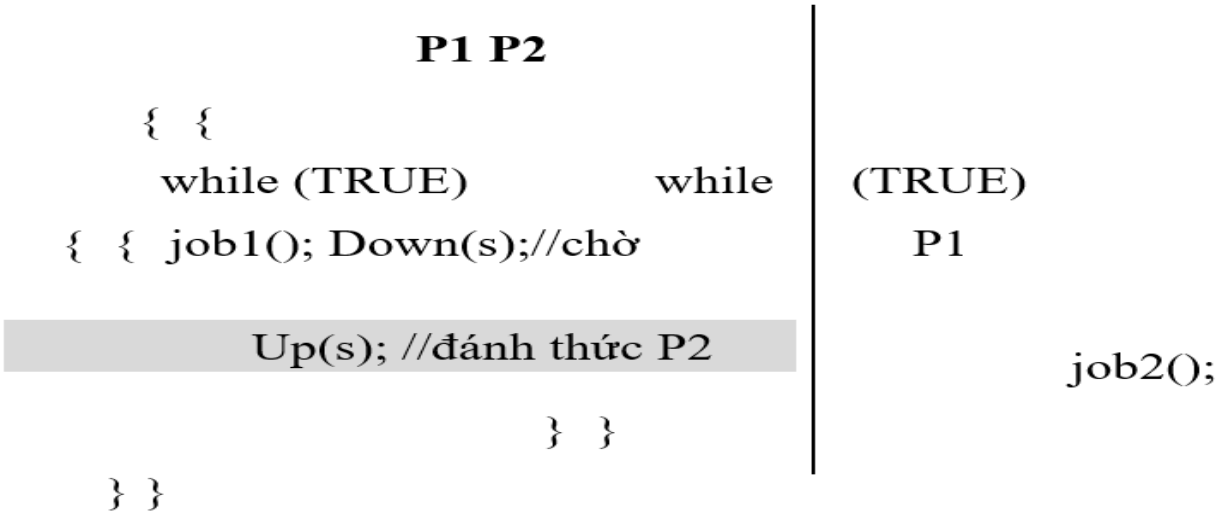
- **Tổ chức truy xuất độc quyền với semaphore:** cho phép bảo đảm nhiều tiến trình cùng truy xuất đến miền tranh chấp mà không có sự mâu thuẫn truy xuất. n tiến trình cùng sử dụng một semaphore s, e(s) được khởi gán là 1. Để thực hiện đồng bộ hóa, tất cả các tiến trình cần phải áp dụng cùng cấu trúc chương trình sau đây:

```
while (TRUE)        {
        Down(s);
        critical-section();
        Up(s);
        Noncritical-section();
                }
```

4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(6) Semaphore (5)

Tổ chức đồng bộ hóa với semaphore: ta có thể đồng bộ hóa hoạt động của hai tiến trình trong tình huống một tiến trình phải đợi một tiến trình khác hoàn tất thao tác nào đó mới có thể bắt đầu hay tiếp tục xử lý. Hai tiến trình chia sẻ một semaphore s, khởi gán e(s) là 0. Cả hai tiến trình có cấu trúc như sau:



4.4. ĐỒNG BỘ HÓA TIẾN TRÌNH

(6) Semaphore (6)

Thảo luận:

- Nếu lập trình viên vô tình đặt các primitive Down và Up sai vị trí, thứ tự trong chương trình thì tiến trình có thể bị khóa vĩnh viễn. Vì thế, việc sử dụng đúng cách semaphore để đồng bộ hóa phụ thuộc hoàn toàn vào lập trình viên và đòi hỏi lập trình viên phải hết sức thận trọng.

4.5. TẮC NGHẼN (deadlock)

4.5. TẮC NGHẼN (deadlock)

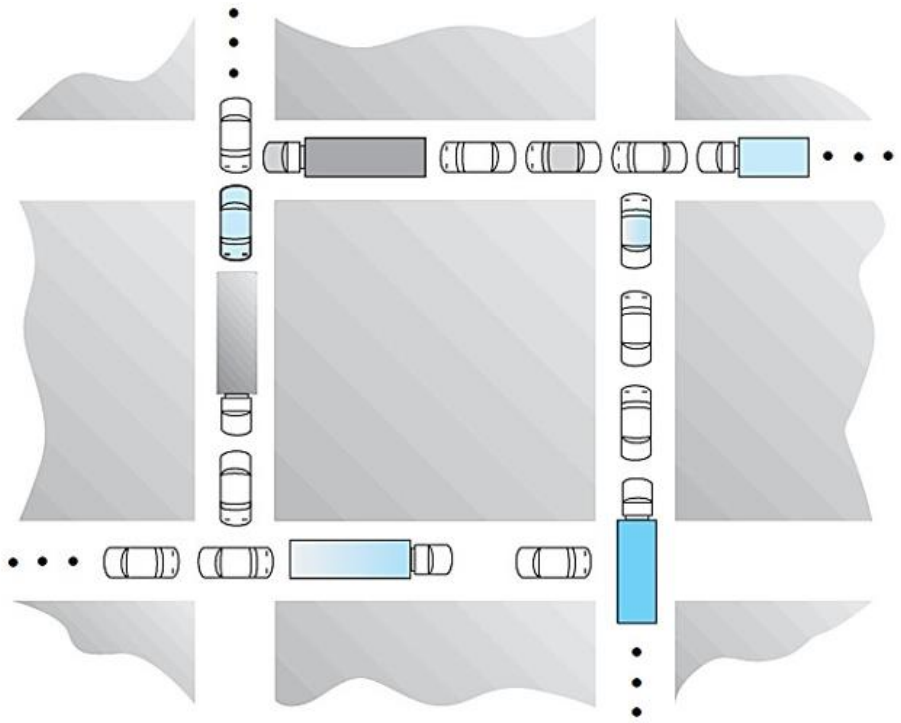
1. Định nghĩa
2. Điều kiện xuất hiện tắc nghẽn
3. Đồ thị cấp phát tài nguyên
4. Các phương pháp xử lý tắc nghẽn
5. Tránh tắc nghẽn

4.5. TẮC NGHẼN (deadlock)

1. Định nghĩa

- Một tập hợp các tiến trình được định nghĩa là **ở trong tình trạng tắc nghẽn** khi mỗi tiến trình trong tập hợp đều chờ đợi một sự kiện mà chỉ có một tiến trình khác trong tập hợp mới có thể phát sinh được.
- Nói cách khác mỗi tiến trình trong tập hợp đều chờ đợi được cấp phát một tài nguyên hiện đang bị một tiến trình khác cũng ở trạng thái blocked chiếm giữ.
- Như vậy **không có tiến trình nào có thể tiếp tục xử lý**, cũng như giải phóng tài nguyên cho tiến trình khác sử dụng, tất cả các tiến trình trong tập hợp đều bị khóa vĩnh viễn.

4.5. TẮC NGHẼN (deadlock)



Một tình huống giao thông tắc nghẽn

4.5. TẮC NGHẼN (deadlock)

2. Điều kiện xuất hiện tắc nghẽn

Để xảy ra tắc nghẽn cần có đủ 4 điều kiện sau đây:

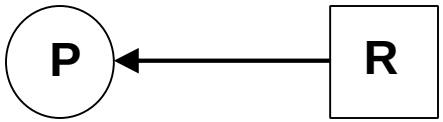
1) **Có sử dụng tài nguyên không thể chia sẻ** (Mutual exclusion): mỗi thời điểm, một tài nguyên không thể chia sẻ được hệ thống cấp phát chỉ cho một tiến trình, khi tiến trình sử dụng xong tài nguyên này, hệ thống mới thu hồi và cấp phát tài nguyên cho tiến trình khác.
2) **Sự chiếm giữ và yêu cầu thêm tài nguyên** (Wait for): các tiến trình tiếp tục chiếm giữ các tài nguyên đã cấp phát cho nó trong khi chờ được cấp phát thêm một số tài nguyên mới.
3) **Không thu hồi tài nguyên từ tiến trình đang giữ chúng** (No preemption): tài nguyên không thể được thu hồi từ tiến trình đang chiếm giữ chúng trước khi tiến trình này sử dụng chúng xong.
4) **Tồn tại một chu kỳ trong đồ thị cấp phát tài nguyên** (Circular wait): có ít nhất hai tiến trình chờ đợi lẫn nhau: tiến trình này chờ được cấp phát tài nguyên đang bị tiến trình kia chiếm giữ và ngược lại.

4.5. TẮC NGHẼN (deadlock)

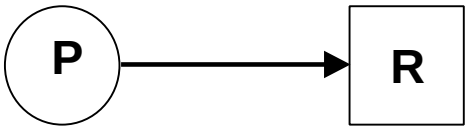
3. Đồ thị cấp phát tài nguyên

- Có thể sử dụng một đồ thị để mô hình hóa việc cấp phát tài nguyên.
- Đồ thị này có 2 loại nút: các tiến trình được biểu diễn bằng hình tròn và mỗi tài nguyên được biểu thị bằng hình vuông.

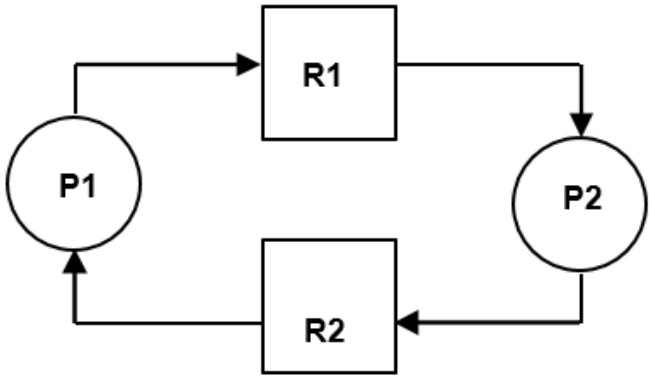
➢Tiến trình P đang giữ tài nguyên R



➢Tiến trình P đang yêu cầu tài nguyên



➢Một tình huống tắc nghẽn



4.5. TẮC NGHẼN (deadlock)

4. Các phương pháp xử lý tắc nghẽn

Chủ yếu có ba hướng tiếp cận để xử lý tắc nghẽn:

1) Sử dụng một nghi thức (Protocol) để đảm bảo rằng hệ thống không bao giờ xảy ra tắc nghẽn.
2) Cho phép xảy ra tắc nghẽn và tìm cách sửa chữa tắc nghẽn.
3) Hoàn toàn bỏ qua việc xử lý tắc nghẽn, xem như hệ thống không bao giờ xảy ra tắc nghẽn

4.5. TẮC NGHẼN (deadlock)

5. Tránh tắc nghẽn

➢Ngăn cản là một mối bận tâm lớn khi sử dụng tài nguyên. Cần phải thực hiện những cơ chế phức tạp để thực hiện ý định này.

(1) Trạng thái an toàn

- Trạng thái A là an toàn nếu hệ thống có thể thỏa mãn các nhu cầu tài nguyên (cho đến tối đa) của mỗi tiến trình theo một thứ tự nào đó mà vẫn ngăn chặn được tắc nghẽn. *Một chuỗi cấp phát an toàn*
- Một thứ tự của các tiến trình <P1, P2, …, Pn> là an toàn đối với tình trạng cấp phát hiện hành nếu với mỗi tiến trình Pi nhu cầu tài nguyên của Pi có thể được thỏa mãn với các tài nguyên còn tự do của hệ thống, cộng với các tài nguyên đang bị chiếm giữ bởi các tiến trình P_j khác, với j<i.
- Một trạng thái an toàn không thể là một trạng thái tắc nghẽn. Ngược lại, một trạng thái không an toàn có thể dẫn đến tình trạng tắc nghẽn.

4.5. TẮC NGHẼN (deadlock)

5. Tránh tắc nghẽn (2)

(2) Chiến lược cấp phát

- Chỉ thỏa mãn các yêu cầu tài nguyên của tiến trình khi trạng thái kết quả là an toàn.

(3) Giải thuật xác định trạng thái an toàn

Cần sử dụng các cấu trúc dữ liệu sau:

```
int Available[NumResources];
// Available[r] = số lượng các thể hiện còn tự do của tài nguyên r int Max[NumProcs, NumResources];
// Max[p, r] = nhu cầu tối đa của tiến trình p về tài nguyên r int Allocation[NumProcs, NumResources];
// Allocation[p, r] = số lượng tài nguyên r thực sự cấp phát cho p int Need[NumProcs, NumResources];
// Need[p, r] = Max[p, r] – Allocation[p, r]
```

4.5. TẮC NGHẼN (deadlock)

Bước 1: giả sử có các mảng int Work = Available;

int Finish[NumProcs] = false;

Bước 2:

Tìm i sao cho:

Finish[i] == false

Need[i] <=Work[i]

Nếu không có i như thế, đến bước 4

Bước 3:

Work = Work + Allocation[i]; Finish[i] = true;

Đến bước 2

Bước 4:

Nếu Finish[i] == true với mọi i, thì hệ thống ở trạng thái an toàn.

4.5. TẮC NGHẼN (deadlock)

Ví dụ

Giả sử tình trạng hiện hành của hệ thống được mô tả như sau:

Tiến trình	Max			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0			
P2	6	1	3	2	1	1			
							4	1	2
P3	3	1	4	2	1	1			
P4	4	2	2	0	0	2			

4.5. TẮC NGHẼN (deadlock)

Ví dụ

- Nếu tiến trình P2 yêu cầu 4 cho R1, 1 cho R3. Hãy cho biết yêu cầu này có thể đáp ứng mà bảo đảm không xảy ra tình trạng deadlock hay không?
- Nhận thấy Available[1]=4, Available[3]=2 đủ để thỏa mãn yêu cầu của P2, ta có:

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	0	1	1
P2	0	0	1	6	1	2			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

4.5. TẮC NGHẼN (deadlock)

Ví dụ

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	2	2	1	0	0	6	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

4.5. TẮC NGHẼN (deadlock)

Ví dụ

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	7	2	3
P2	0	0	0	0	0	0			
P3	1	0	3	2	1	1			
P4	4	2	0	0	0	2			

4.5. TẮC NGHẼN (deadlock)

Ví dụ

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	4
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	4	2	0	0	0	2			

4.5. TẮC NGHẼN (deadlock)

Ví dụ

Tiến trình	Need			Allocation			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0			
P4	0	0	0	0	0	0			

-> Kết luận: Trạng thái kết quả là an toàn, có thể cấp phát.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (1)

- ❖Bộ nhớ chính là thiết bị lưu trữ duy nhất thông qua đó CPU trao đổi dữ liệu với môi trường ngoài.
- ➢Quản lý bộ nhớ là một trong những nhiệm vụ trọng tâm của HĐH.
- ❖Bộ nhớ chính được tổ chức như một mảng một chiều các từ nhớ (word), mỗi từ nhớ có một địa chỉ.
- ❖Các HĐH hiện đại đều cho phép chế độ đa nhiệm → nảy sinh nhu cầu chia sẻ bộ nhớ giữa các tiến trình.
- ➢HĐH có trách nhiệm cấp phát vùng nhớ cho các tiến trình có yêu cầu.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (2)

❖**Vấn đề:**

- **Sự tương ứng giữa địa chỉ logic và địa chỉ vật lý (physic):** làm cách nào để chuyển đổi một địa chỉ tượng trưng (symbolic) trong chương trình thành một địa chỉ thực trong bộ nhớ chính?
- **Quản lý bộ nhớ vật lý:** làm cách nào để mở rộng bộ nhớ có sẵn nhằm lưu trữ được nhiều tiến trình đồng thời?
- **Chia sẻ thông tin:** làm thế nào để cho phép hai tiến trình có thể chia sẻ thông tin trong bộ nhớ?
- **Bảo vệ:** làm thế nào để ngăn chặn các tiến trình xâm phạm đến vùng nhớ được cấp phát cho tiến trình khác?

❖ Các địa chỉ trong chương trình nguồn là địa chỉ tượng trưng → phải chuyển đổi các địa chỉ này thành các địa chỉ tuyệt đối trong bộ nhớ chính.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (3)

- Có thể thực hiện *kết buộc* các lệnh và dữ liệu với các địa chỉ bộ nhớ vào một trong những thời điểm sau:
  - Thời điểm biên dịch.
  - Thời điểm nạp
  - Thời điểm xử lý.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (4)

- Thời điểm biên dịch:
  - Nếu tại thời điểm biên dịch, có thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ.
  - Nếu về sau có sự thay đổi vị trí thường trú lúc đầu của chương trình, cần phải biên dịch lại chương trình.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (5)

- Thời điểm nạp:
  - Nếu tại thời điểm biên dịch, chưa thể biết vị trí mà tiến trình sẽ thường trú trong bộ nhớ, trình biên dịch cần phát sinh mã tương đối (translatable).
  - Sự liên kết địa chỉ được trì hoãn đến thời điểm chương trình được nạp vào bộ nhớ, lúc này các địa chỉ tương đối sẽ được chuyển thành địa chỉ tuyệt đối do đã biết vị trí bắt đầu lưu trữ tiến trình.
  - Khi có sự thay đổi vị trí lưu trữ, chỉ cần nạp lại chương trình để tính toán lại các địa chỉ tuyệt đối mà không cần biên dịch lại.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (6)

- Thời điểm xử lý:
  - Nếu có nhu cầu di chuyển tiến trình từ vùng nhớ này sang vùng nhớ khác trong quá trình tiến trình xử lý, thì thời điểm kết buộc địa chỉ phải trì hoãn đến tận thời điểm xử lý.
  - Để thực hiện kết buộc địa chỉ vào thời điểm xử lý cần sử dụng cơ chế phần cứng đặc biệt.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (7)

Một vài khái niệm quan trọng:

1) **Địa chỉ logic: (**hay địa chỉ ảo) là tất cả các địa chỉ do bộ xử lý tạo ra.
2) **Địa chỉ vật lý:** là địa chỉ thực tế mà trình quản lý bộ nhớ nhìn thấy và thao tác được.
3) **Không gian địa chỉ:** là vùng địa chỉ mà chương trình có thể sử dụng được.
4) **Không gian vật lý:** là tập hợp tất cả các địa chỉ vật lý trong các địa chỉ ảo.

4.6. QUẢN LÝ BỘ NHỚ - 4.6.1. Mở đầu (8)

Chú ý:

- Địa chỉ ảo và địa chỉ vật lý là như nhau trong phương thức kết buộc địa chỉ vào thời điểm biên dịch và thời điểm nạp. Nhưng có sự khác biệt trong phương thức kết buộc vào thời điểm xử lý.
- MMU (Memory Management Unit) là một cơ chế phần cứng được sử dụng để thực hiện chuyển đổi địa chỉ ảo thành địa chỉ vật lý vào thời điểm xử lý.
- Chương trình của người sử dụng chỉ thao tác trên các địa chỉ ảo, không bao giờ nhìn thấy các địa chỉ vật lý. Địa chỉ thật sự ứng với vị trí của dữ liệu trong bộ nhớ chỉ được xác định khi thực hiện truy xuất đến dữ liệu.

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (paging)

Khái niệm:

- Phân bộ nhớ vật lý thành các khối (block) có kích thước cố định và bằng nhau, gọi là khung trang (page frame).
- Không gian địa chỉ cũng được chia thành các khối có cùng kích thước với khung trang, và được gọi là trang (page).
- Khi cần nạp một tiến trình để xử lý, các trang của tiến trình sẽ được nạp vào những khung trang còn trống.
- Một tiến trình kích thước N trang sẽ yêu cầu N khung trang tự do.

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (2)

❖ Cơ chế MMU trong kỹ thuật phân trang

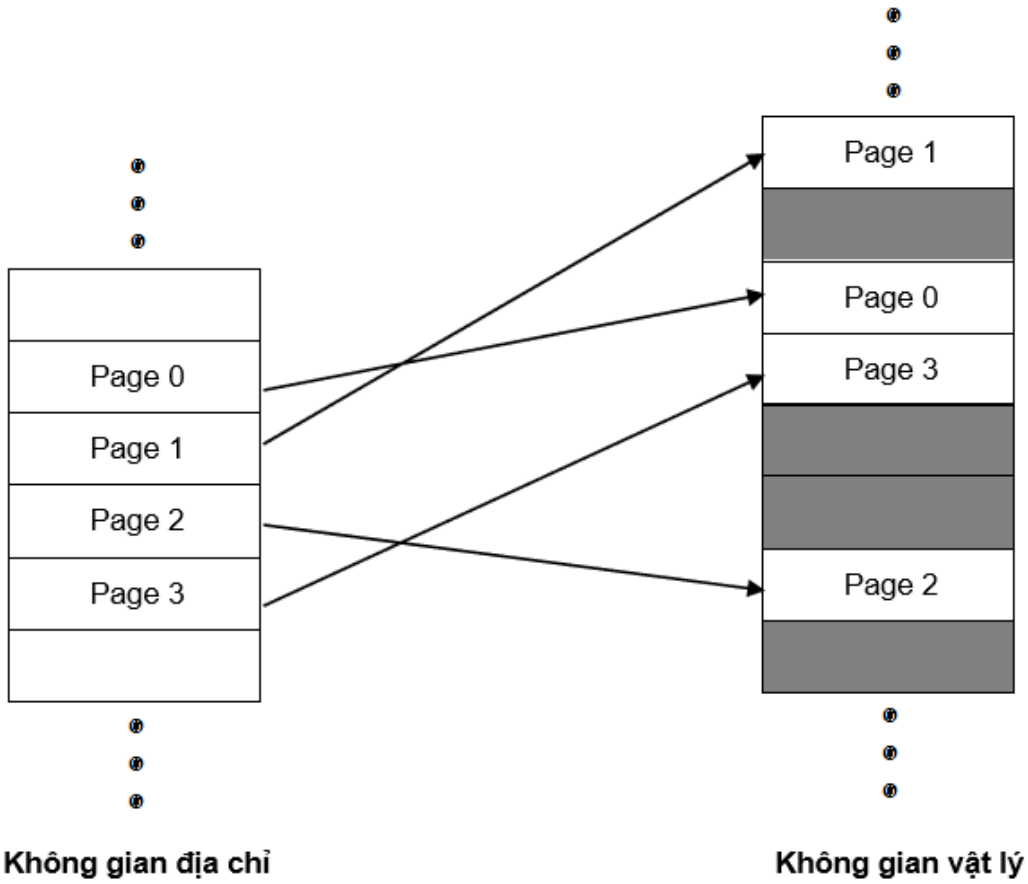
➢ Cơ chế phần cứng hỗ trợ thực hiện chuyển đổi địa chỉ trong cơ chế phân trang là **bảng trang** (pages table). Mỗi phần tử trong bảng trang cho biết các địa chỉ bắt đầu của vị trí lưu trữ trang tương ứng trong bộ nhớ vật lý (số hiệu khung trang trong bộ nhớ vật lý đang chứa trang).

❖ Chuyển đổi địa chỉ

➢ Mỗi địa chỉ phát sinh bởi CPU được chia thành hai phần:

- **Số hiệu trang (p):** sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang.
- **Địa chỉ tương đối trong trang (d):** kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

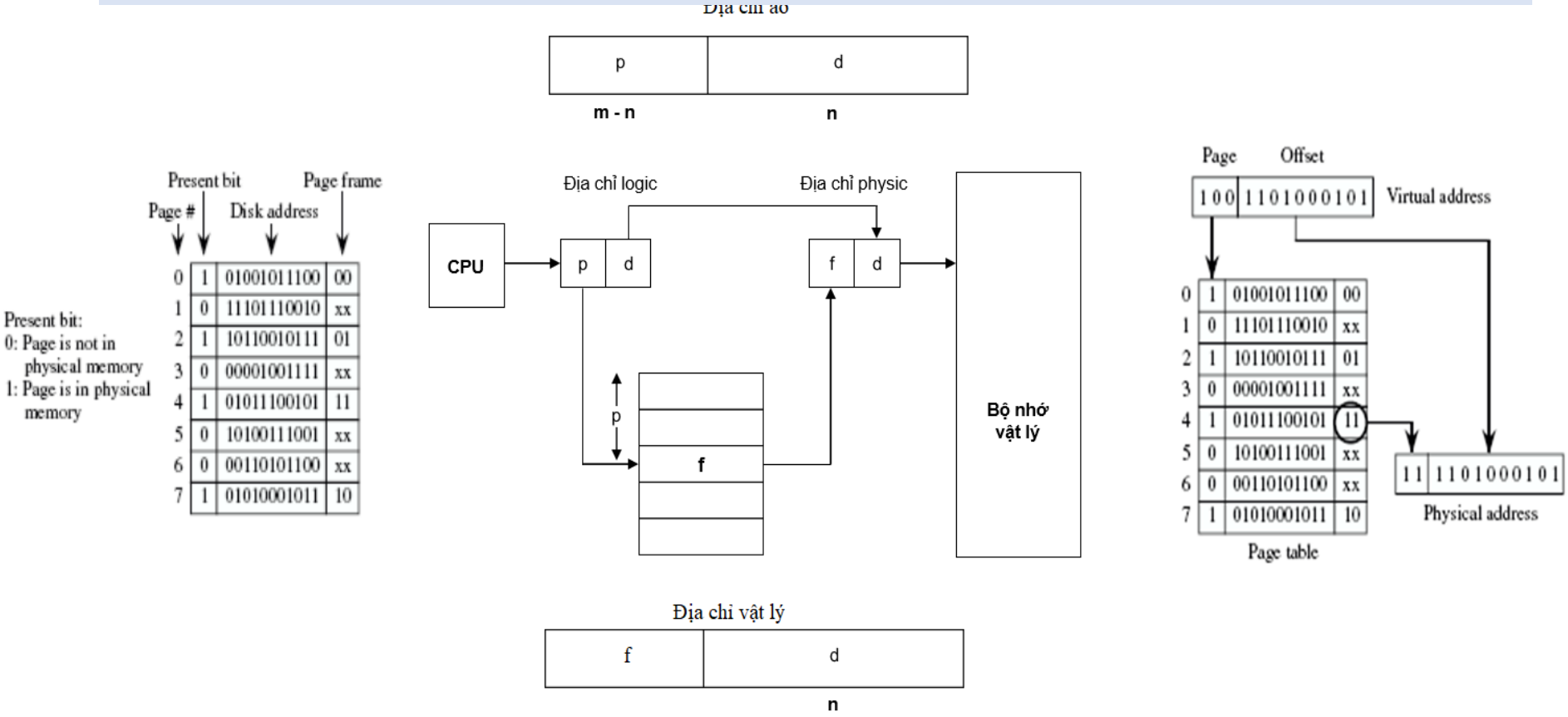
4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (3)



Mô hình bộ nhớ phân trang

- Kích thước của trang do phần cứng quy định.
- Kích thước của một trang thông thường là một lũy thừa của 2 (biến đổi trong phạm vi 2^9=512 bytes và 2^{13}=8192 bytes).

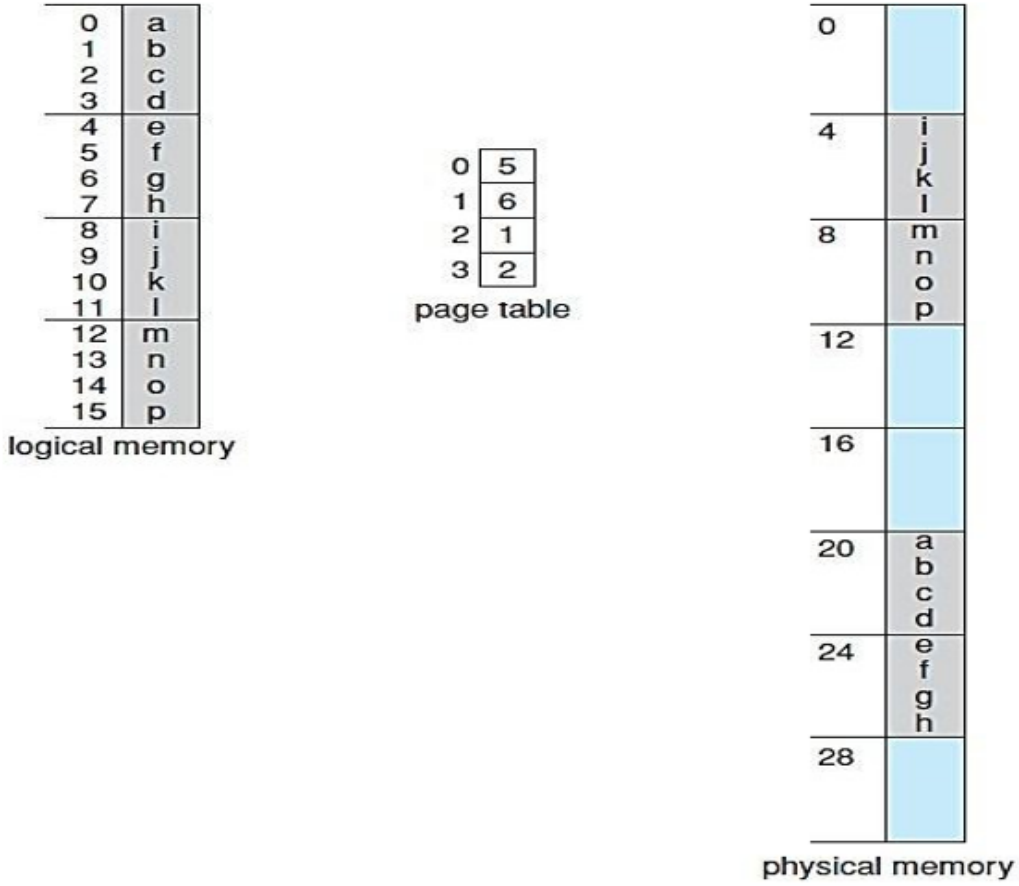
4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (4)



Quan hệ giữa địa chỉ ảo và địa chỉ vật lý

Nếu kích thước của không gian địa chỉ là 2^m và kích thước trang là 2^n thì m-n bits cao của địa chỉ ảo sẽ biểu diễn số hiệu trang, và n bits thấp cho biết địa chỉ tương đối trong trang.

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (5)



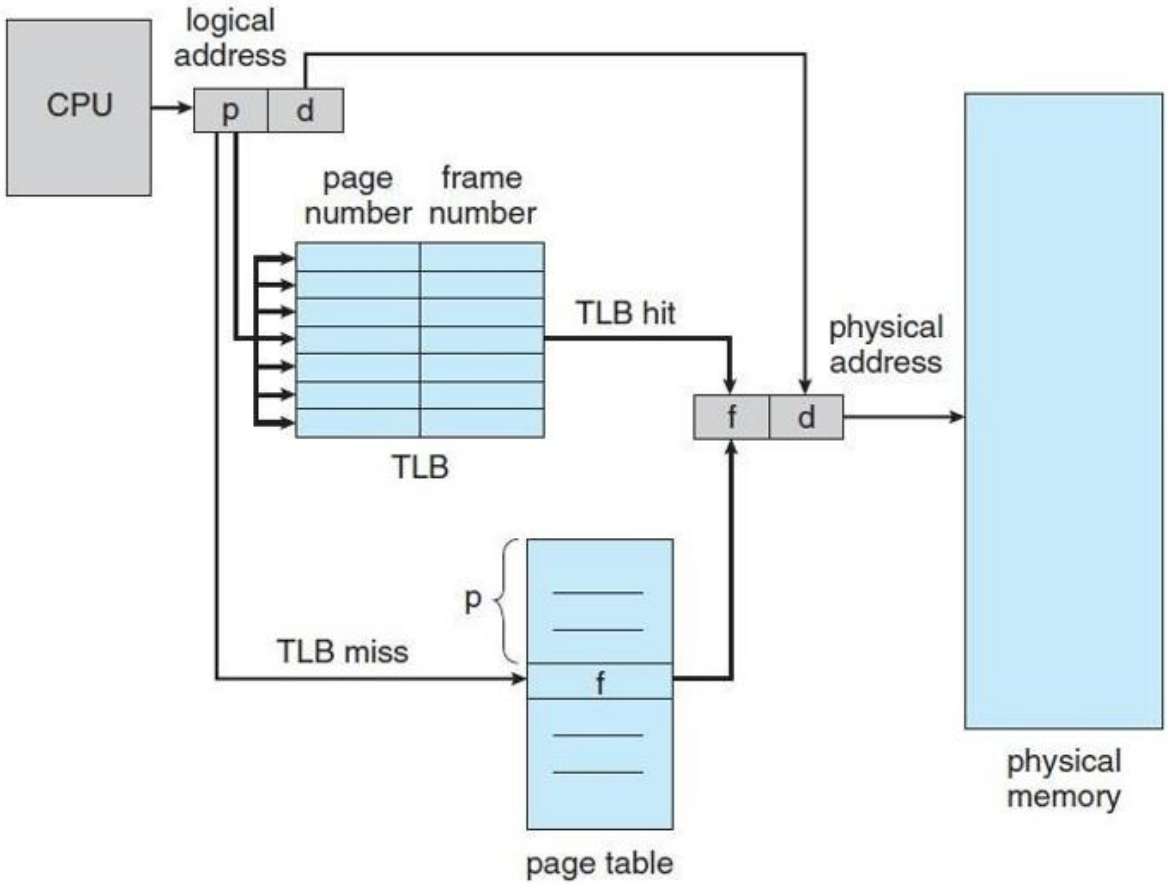
Hình 5.13. Ví dụ phân trang với bộ nhớ 32 byte, kích thước trang là 4 byte

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (6)

❖Cài đặt bảng trang

- Sử dụng một tập các thanh ghi để cài đặt bảng trang → chỉ phù hợp với các bảng trang có kích thước nhỏ.
- Nếu bảng trang có kích thước lớn, nó phải được lưu trữ trong bộ nhớ chính và sử dụng một thanh ghi để lưu địa chỉ bắt đầu lưu trữ bảng trang. → Theo cách tổ chức này, mỗi lần truy xuất đến dữ liệu hay chỉ thị đều đòi hỏi hai lần truy xuất bộ nhớ: *một cho truy xuất đến bảng trang* và *một cho bản thân dữ liệu*. → tránh việc truy xuất bộ nhớ hai lần bằng cách sử dụng *bộ nhớ kết hợp* (TBLs: Translation Lookaside Buffers).
- TBLs được sử dụng để lưu trữ các trang bộ nhớ được truy cập gần hiện tại nhất. Khi CPU phát sinh một địa chỉ, số hiệu trang của địa chỉ sẽ được so sánh với các phần tử trong TBLs, nếu có trang tương ứng trong TBLs thì sẽ xác định được ngay số hiệu khung trang tương ứng, nếu không mới cần thực hiện thao tác tìm kiếm trong bảng trang.

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (7)



Hình 5.14. Phần cứng hỗ trợ phân trang sử dụng TBLs

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (8)

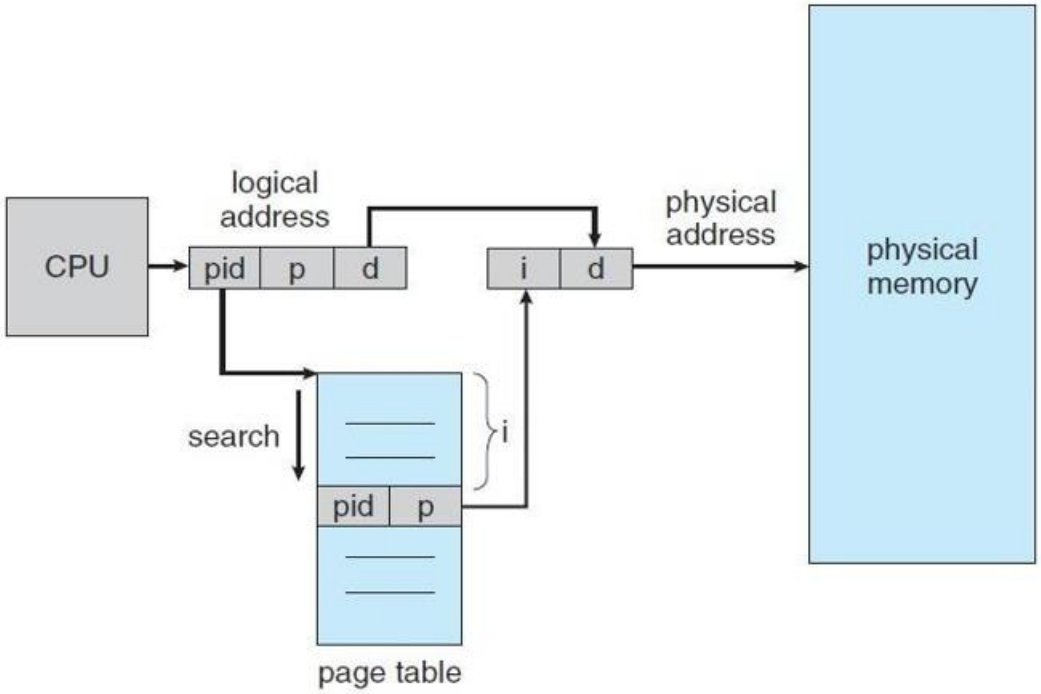
❖ Tổ chức bảng trang

Đa số các HĐH cấp cho mỗi tiến trình một bảng trang. Tuy nhiên, phương pháp này không thể chấp nhận được nếu HĐH quản lý một không gian địa chỉ có dung ượng quá lớn (2^{32}, 2^{64}) vì bảng trang cần vùng nhớ qua lớn!

Cách giải quyết:

- **Phân trang đa cấp:** phân chia bảng trang thành các phần nhỏ, bản thân bảng trang cũng sẽ được phân trang.
- **Bảng trang nghịch đảo (Inverted page table):** sử dụng duy nhất một bảng trang nghịch đảo cho tất cả các tiến trình. Mỗi ptử trong bảng trang nghịch đảo phản ánh một khung trang trong bộ nhớ bao gồm địa chỉ logic của một trang đang được lưu trữ trong bộ nhớ vật lý tại khung trang này, cùng với thông tin về tiến trình đang được sở hữu trang. Mỗi địa chỉ ảo là một bộ ba <idp, p, d>: (1) idp là định danh của tiến trình. (2) p là số hiệu trang (3) d là địa chỉ tương đối trong trang.

4.6. QUẢN LÝ BỘ NHỚ 4.6.2. Phân trang (9)



Hình 5.15. Bảng trang nghịch đảo

Mỗi phần tử trong bảng trang nghịch đảo là một cặp <idp, p>. Khi một tham khảo đến bộ nhớ được phát sinh, một phần địa chỉ ảo là <idp, p> được đưa đến cho trình quản lý bộ nhớ để tìm phần tử tương ứng trong bảng trang nghịch đảo. Nếu tìm thấy địa chỉ vật lý <i,d> sẽ được phát sinh. Trong các trường hợp khác, xem như tham khảo bộ nhớ đã truy xuất một địa chỉ bất hợp lệ

4.6. QUẢN LÝ BỘ NHỚ 4.6.3. Phân đoạn (1)

4.6.3. Phân đoạn

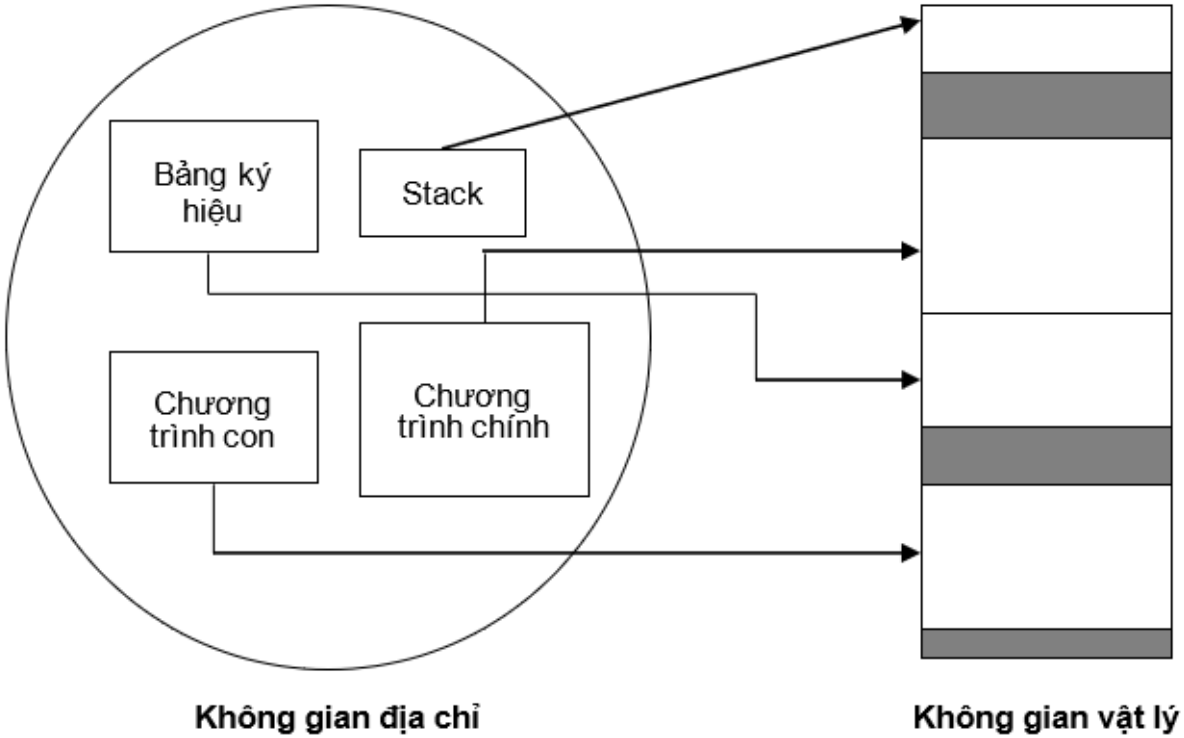
❖ Ý tưởng

- Quan niệm không gian địa chỉ là một tập các phân đoạn (segments).
- Các phân đoạn là những phần bộ nhớ kích thước *khác* nhau và có liên hệ logic với nhau.
- Mỗi phân đoạn có một tên gọi (số hiệu phân đoạn) và có một độ dài → Người dùng sẽ thiết lập mỗi địa chỉ với hai giá trị: <số hiệu phân đoạn, offset>

❖ Cơ chế MMU

- Cần phải xd một ánh xạ để chuyển đổi các địa chỉ 2 chiều được người dùng định nghĩa thành địa chỉ vật lý một chiều.
- Sự chuyển đổi này được thực hiện qua một bảng phân đoạn.
- Mỗi thành phần trong bảng phân đoạn bao gồm một thanh ghi nền và một thanh ghi giới hạn.
- Thanh ghi nền lưu trữ địa chỉ vật lý nơi bắt đầu phân đoạn trong bộ nhớ. Trong khi thanh ghi giới hạn đặc tả chiều dài của phân đoạn.

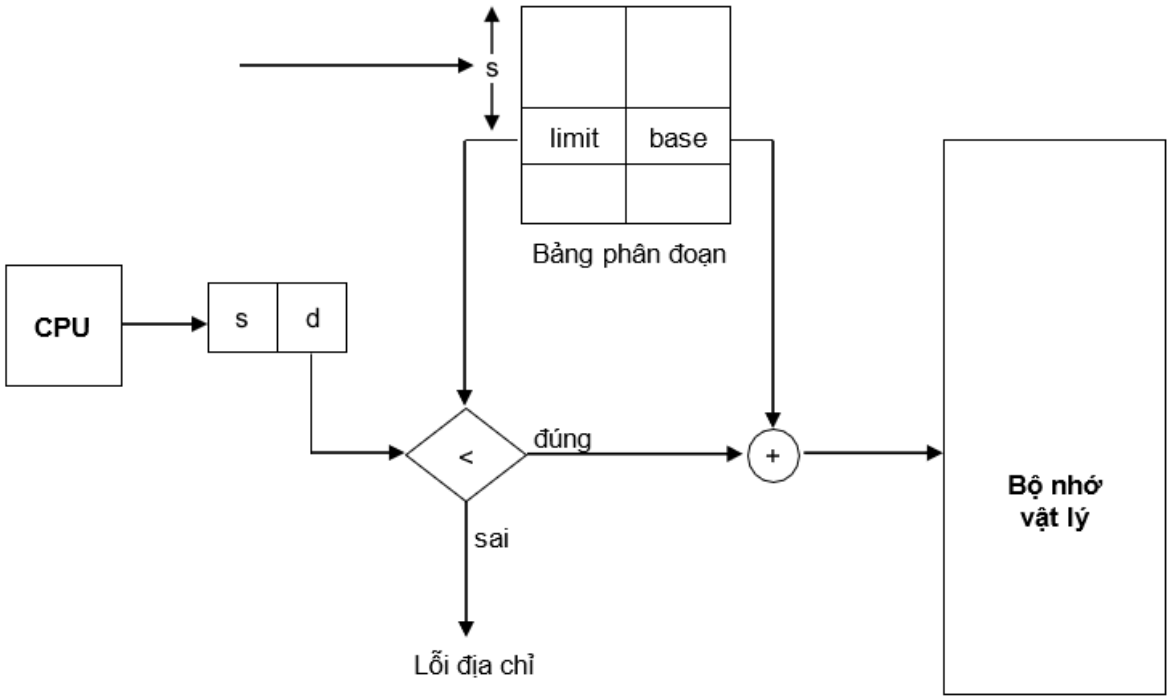
4.6. QUẢN LÝ BỘ NHỚ 4.6.3. Phân đoạn (2)



Hình 5.1.6 Một mô hình phân đoạn bộ nhớ

4.6. QUẢN LÝ BỘ NHỚ 4.6.3. Phân đoạn (3)

❖Chuyển đổi địa chỉ



Hình 5.1.7 Cơ chế phần cứng hỗ trợ phân đoạn

- **Mỗi địa chỉ ảo là một bộ <s, d>:**
  - **Số hiệu phân đoạn s**: được sử dụng như chỉ mục đến bảng phân đoạn.
  - **Địa chỉ tương đối d:** có giá trị từ 0 đến giới hạn chiều dài của phân đoạn.
  - Nếu địa chỉ tương đối hợp lệ, nó sẽ được cộng với giá trị chứa trong thanh ghi nền để phát sinh địa chỉ vật lý tương ứng.

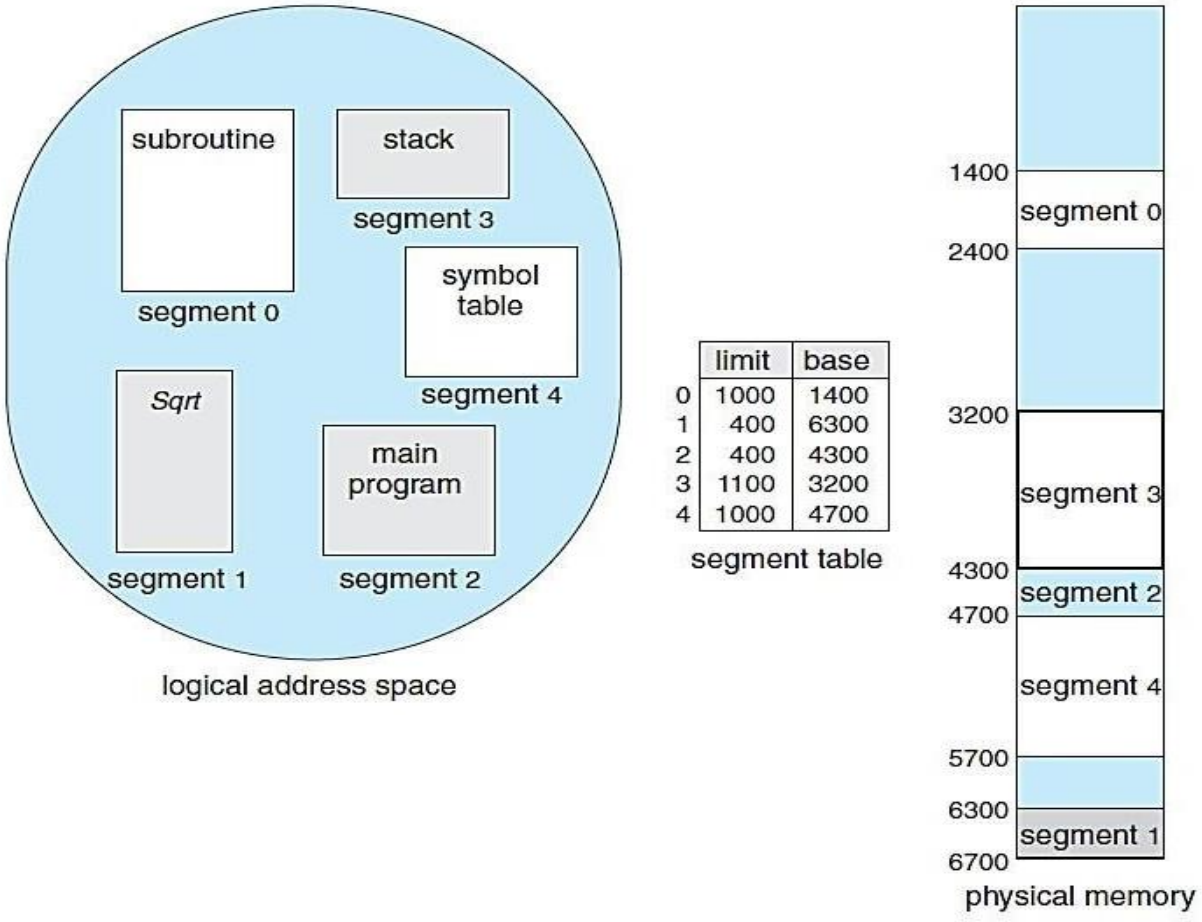
4.6. QUẢN LÝ BỘ NHỚ 4.6.3. Phân đoạn (4)

❖ Cài đặt bảng phân đoạn

- Có thể sử dụng các thanh ghi để lưu trữ bảng phân đoạn nếu số lượng phân đoạn nhỏ. Trong trường hợp chương trình bao gồm quá nhiều phân đoạn, bảng phân đoạn phải được lưu trong bộ nhớ chính. Một thanh ghi nền bảng phân đoạn (STBR) *chỉ đến địa chỉ bắt đầu của bảng phân đoạn*. Vì số lượng phân đoạn sử dụng trong một chương trình biến động, cần sử dụng thêm một thanh ghi đặc tả kích thước bảng phân đoạn (STLR).
- **Với một địa chỉ logic <s, d>,** trước tiên số hiệu phân đoạn s được kiểm tra tính hợp lệ (**s<STLR**). Kế tiếp, cộng giá trị s với STBR để có được địa chỉ của phần tử thứ s trong bảng phân đoạn (STBR + s). Địa chỉ vật lý cuối cùng là (STBR + s +d).

- Cũng như đối với kỹ thuật phân trang, có thể nâng tốc độ truy xuất bộ nhớ bằng cách sử dụng TBLs để lưu trữ các phần tử trong bảng trang được truy cập gần nhất.

4.6. QUẢN LÝ BỘ NHỚ 4.6.3. Phân đoạn (2)



Hình 5.18. Ví dụ một trường hợp phân đoạn

4.6.4. Phân trang kết hợp đoạn

4.6.4. Phân trang kết hợp đoạn

❖ Ý tưởng

- Không gian địa chỉ là một tập các phân đoạn, mỗi phân đoạn được chia thành nhiều trang. Khi một tiến trình được đưa vào hệ thống, HĐH sẽ cấp phát cho tiến trình các trang cần thiết để chứa đủ các phân đoạn của tiến trình.

❖ Cơ chế MMU

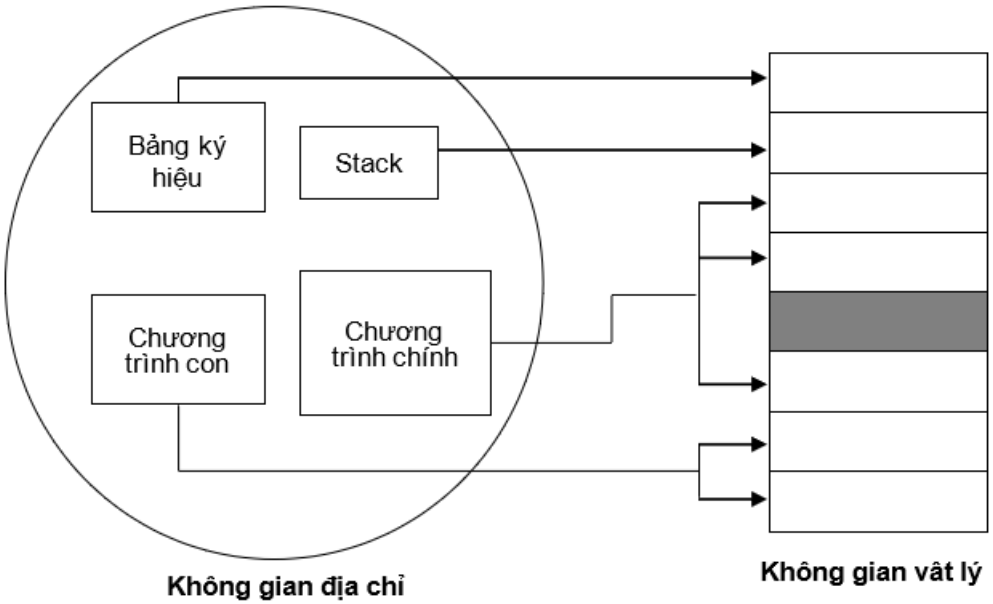
Để hỗ trợ kỹ thuật phân đoạn, cần phải có một bảng phân đoạn, nhưng giờ đây mỗi phân đoạn cần có một bảng trang riêng biệt.

▪ Chuyển đổi địa chỉ

Mỗi địa chỉ logic là một bộ ba: <s, p, d>

- Số hiệu phân đoạn s : sử dụng như chỉ mục đến phần tử tương ứng trong bảng phân đoạn.
- Số hiệu trang p : sử dụng như chỉ mục đến phần tử tương ứng trong bảng trang của phân đoạn.
- Địa chỉ tương đối trong trang d : kết hợp với địa chỉ bắt đầu của trang để tạo ra địa chỉ vật lý mà trình quản lý bộ nhớ sử dụng.

4.6. QUẢN LÝ BỘ NHỚ 4.6.4. Phân trang kết hợp đoạn (1)



Hình 5.19. Mô hình phân đoạn kết hợp phân trang

- *Tất cả các mô hình tổ chức bộ nhớ ở trên đều có khuynh hướng cấp phát cho tiến trình toàn bộ các trang yêu cầu trước khi thật sự xử lý. Vì bộ nhớ vật lý có kích thước rất giới hạn, điều này dẫn đến hai điểm bất tiện:*
- Kích thước tiến trình bị giới hạn bởi kích thước của bộ nhớ vật lý.
- Khó có thể duy trì nhiều tiến trình cùng lúc trong bộ nhớ, và như vậy khó nâng cao mức độ đa chương của hệ thống.

4.6.5. Bộ nhớ ảo (virtual memory)

❖ Giới thiệu

- Nếu đặt toàn thể không gian địa chỉ vào bộ nhớ vật lý, thì kích thước của chương trình bị giới hạn bởi kích thước bộ nhớ vật lý.
- Một giải pháp được đề xuất là cho phép thực hiện một chương trình chỉ được nạp từng phần vào bộ nhớ vật lý.
- Một cách thể hiện ý tưởng của giải pháp này là kỹ thuật overlay. Kỹ thuật overlay không đòi hỏi bất kỳ sự trợ giúp đặc biệt nào của hệ điều hành. Nhưng trái lại lập trình viên phải biết cách lập trình theo cấu trúc overlay và điều này đòi hỏi khá nhiều công sức.
- Để giải phóng lập trình viên khỏi các khó khăn cho công việc lập trình của họ, người ta nghĩ đến các kỹ thuật tự động cho phép xử lý một chương trình có kích thước lớn chỉ với một vùng nhớ có kích thước nhỏ.
  → Bộ nhớ ảo (Virtual Memory).

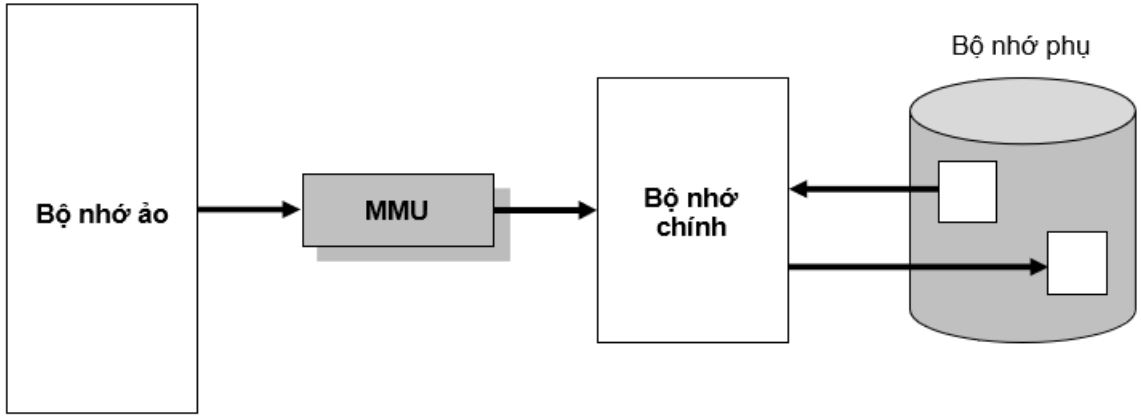
4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (2)

❖ **Khái niệm bộ nhớ ảo:**

Bộ nhớ ảo là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý. Bộ nhớ ảo mô hình hóa bộ nhớ như một bảng lưu trữ rất lớn và đồng nhất, tách biệt hẳn khái niệm không gian địa chỉ và không gian vật lý. Người sử dụng chỉ nhìn thấy và làm việc trong không gian địa chỉ ảo, việc chuyển đổi sang không gian vật lý do hệ điều hành thực hiện với sự trợ giúp của các cơ chế phần cứng cụ thể.

- Cần kết hợp kỹ thuật swapping để chuyển các phần của chương trình vào ra giữa bộ nhớ chính và bộ nhớ phụ khi cần thiết.
- Nhờ việc tách biệt bộ nhớ ảo và bộ nhớ vật lý, có thể tổ chức một bộ nhớ ảo có kích thước lớn hơn bộ nhớ vật lý.
- Bộ nhớ ảo cho phép giảm nhẹ công việc của lập trình viên vì họ không cần bận tâm đến giới hạn của vùng nhớ vật lý, cũng nhọ không cần tổ chức chương trình theo cấu trúc overlay.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. bộ nhớ ảo (3)



Hình 5.20. Mô hình bộ nhớ ảo

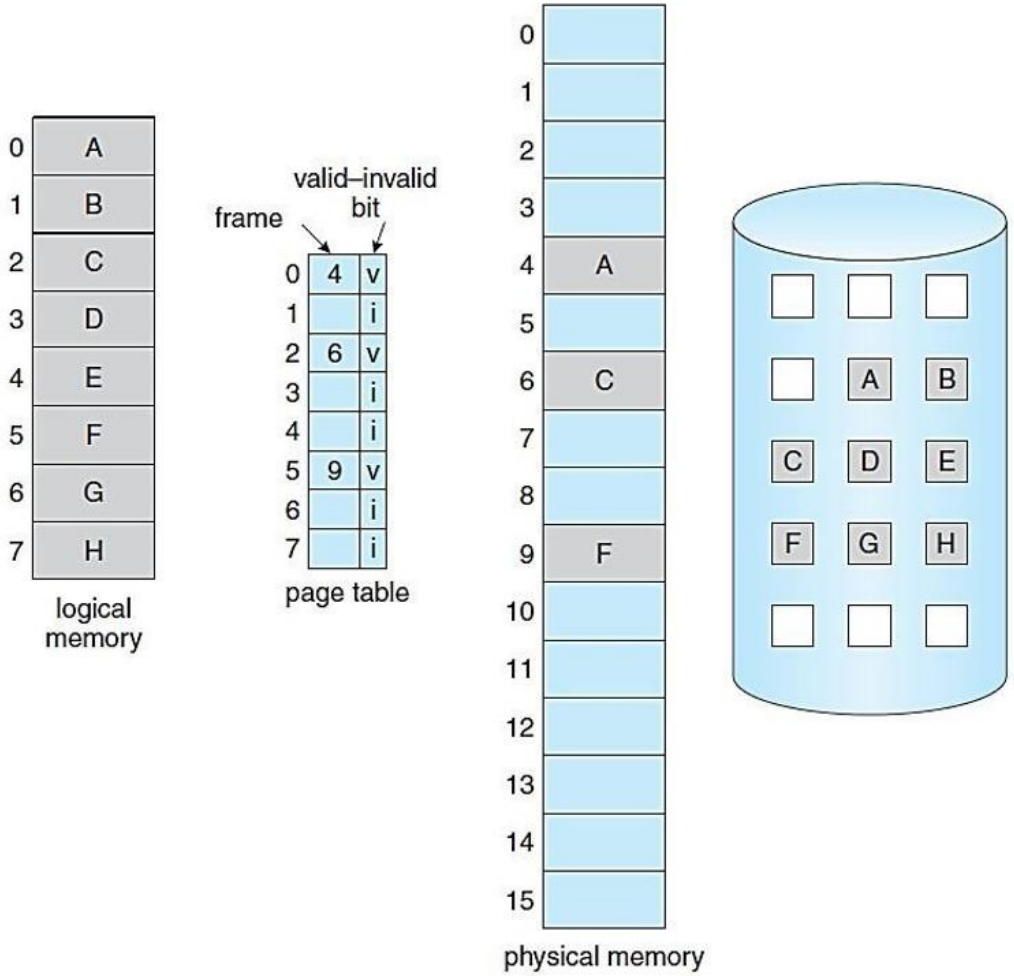
- **Bộ nhớ ảo thường được thực hiện với kỹ thuật phân trang theo yêu cầu (demand paging).**
- Cũng có thể sử dụng kỹ thuật phân đoạn theo yêu cầu (demand segmentation) để cài đặt bộ nhớ ảo. Tuy nhiên, việc cấp phát và thay thế các phân đoạn phức tạp hơn thao tác trên trang vì kích thước không bằng nhau của các đoạn.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (4)

Phân trang theo yêu cầu (demand paging)

- Hệ thống phân trang theo yêu cầu là hệ thống sử dụng kỹ thuật swapping. Một tiến trình được xem như một tập các trang, thường trú trên bộ nhớ phụ (thường là đĩa). Khi cần xử lý, tiến trình sẽ được nạp vào bộ nhớ chính. Nhưng thay vì nạp toàn bộ chương trình, chỉ những trang cần thiết trong thời điểm hiện tại mới được nạp vào bộ nhớ. Như vậy, một trang chỉ được nạp vào bộ nhớ chính khi có yêu cầu.
- Với mô hình này, cần cung cấp một cơ chế phần cứng giúp phân biệt các trang ở trong bộ nhớ chính và các trang trên đĩa → sử dụng lại bit valid-invalid:
  - **valid:** trang tương ứng là hợp lệ và đang ở trong bộ nhớ chính.
  - **invalid:** hoặc trang bất hợp lệ (không thuộc về không gian địa chỉ của tiến trình) hoặc trang hợp lệ nhưng đang được lưu trên bộ nhớ phụ.
- Một phần tử trong bảng trang mô tả cho một trang không nằm trong bộ nhớ chính sẽ được đánh dấu invalid và chứa địa chỉ của trang trên bộ nhớ phụ.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (5)



Hình 5.21. Bảng trang của một trường hợp phân trang theo yêu cầu

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (6)

❖ *Cơ chế MMU*

Cơ chế phần cứng hỗ trợ kỹ thuật phân trang theo yêu cầu là sự kết hợp của cơ chế hỗ trợ **kỹ thuật phân trang** và **kỹ thuật swapping**.

- **Bảng trang:** cấu trúc bảng trang phải cho phép phản ánh tình trạng của một trang là đang nằm trong bộ nhớ chính hay bộ nhớ phụ.
- **Bộ nhớ phụ:** bộ nhớ phụ lưu trữ những trang không được nạp vào bộ nhớ chính. Bộ nhớ phụ thường được sử dụng là đĩa và là vùng không gian đĩa dùng để lưu trữ tạm các trang trong kỹ thuật swapping được gọi là không gian swapping.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (7)

❖ ***Lỗi trang***

Truy xuất đến một trang được đánh dấu bất hợp lệ → lỗi trang (page fault).

Phát sinh một ngắt để báo cho HĐH → HĐH sẽ xử lý lỗi trang như sau:

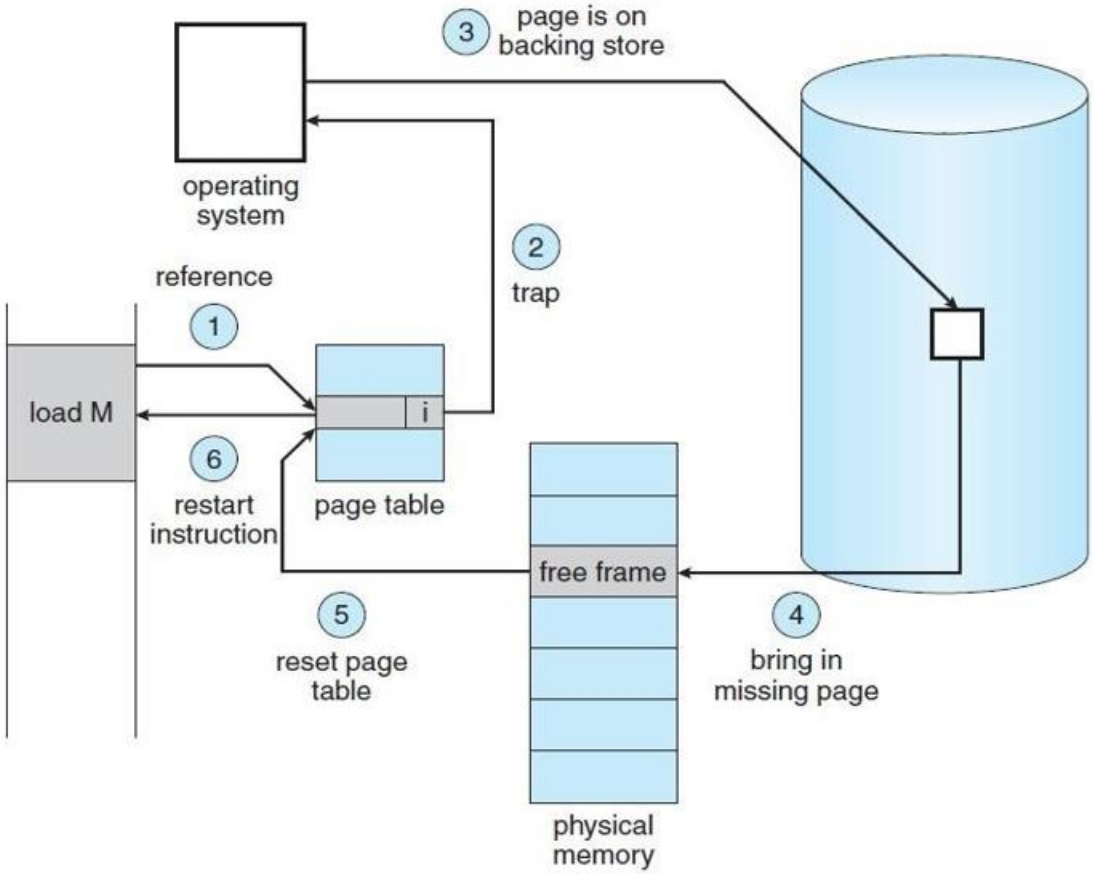
1. Kiểm tra truy xuất đến bộ nhớ có hợp lệ hay không?
2. Nếu truy xuất bất hợp lệ thì kết thúc tiến trình. Ngược lại thì đến bước 3
3. Tìm vị trí chứa trang muốn truy xuất trên đĩa
4. Tìm một khung trang trống trong bộ nhớ chính

- Nếu tìm thấy: đến bước 5
- Nếu không còn khung trang trống, chọn một khung trang "nạn nhân" và chuyển trang "nạn nhân" ra bộ nhớ phụ, cập nhật bảng trang tương ứng rồi đến bước 5

5. Chuyển trang muốn truy xuất từ bộ nhớ phụ vào bộ nhớ chính: nạp trang cần truy xuất vào khung trang trống đã chọn (hay vừa mới làm trống), cập nhật nội dung bảng trang, bảng khung trang tương ứng.

6. Tái kích hoạt tiến trình người sử dụng

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. bộ nhớ ảo (8)



Hình 5.22. Sơ đồ các bước xử lý lỗi trang

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (9)

❖ Thay thế trang

- Khi xảy ra một lỗi trang, cần phải mang trang cần thiết nhưng vắng mặt vào bộ nhớ. Nếu không có một khung trang nào trống, HĐH cần thực hiện công việc thay thế: chuyển một trang ra bộ nhớ phụ và nạp một trang khác vào bộ nhớ chính.
- Có thể giảm bớt số lần chuyển trang bằng cách sử dụng thêm một bit cập nhật (dirty bit). Bit này được gắn với mỗi trang để phản ánh tình trạng trang có bị cập nhật hay không. Giá trị của bit được cơ chế phần cứng đặt là 1 mỗi lần có một từ được ghi vào trang để ghi nhận nội dung trang có bị sửa đổi. Khi cần thay thế một trang, nếu bit cập nhật có giá trị là 1 thì trang cần được lưu lại trên đĩa, ngược lại thì không cần lưu trữ trang trở lại đĩa.
- Vấn đề chính khi thay thế trang là chiến thuật chọn trang để thay thế.

Ví dụ 1: giả sử theo vết xử lý của một tiến trình và nhận thấy tiến trình thực hiện truy xuất các địa chỉ theo thứ tự sau:

0100, 0423, 0101, 0162, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

- Nếu có kích thước của một trang là 100 bytes, có thể viết lại chuỗi truy xuất trên giản lược hơn như sau: 1, 4, 1, 6, 1, 6, 1, 6, 1

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (10)

Ví dụ 2: Để xác định số lỗi trang xảy ra khi sử dụng một thuật toán thay thế trang nào đó trên một chuỗi truy xuất cụ thể, còn cần phải biết số lượng khung trang sử dụng trong hệ thống.

- **Để minh họa các thuật toán thay thế trang sẽ trình bày, chuỗi truy xuất được sử dụng là: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1**

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
	0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
		1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

Kí hiệu * là có lỗi trang và có 15 lỗi trang

1. Thuật toán FIFO: Ghi nhận thời điểm một trang được mang vào bộ nhớ chính. Khi cần thay thế trang, trang ở trong bộ nhớ lâu nhất sẽ được chọn.

Ví dụ: sử dụng 3 khung trang, ban đầu cả 3 đều trống

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (11)

❖ **Thảo luận:**

- Để áp dụng thuật toán FIFO, thực tế không nhất thiết phải ghi nhận thời điểm mỗi trang được nạp vào bộ nhớ, mà chỉ cần tổ chức quản lý các trang trong bộ nhớ trong một danh sách FIFO, khi đó trang đầu danh sách sẽ được chọn để thay thế.
- Thuật toán thay thế trang FIFO dễ hiểu, dễ cài đặt. Tuy nhiên kém hiệu quả: trang được chọn để thay thế có thể là trang chứa nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm. Do vậy, khi bị thay thế sẽ nhanh chóng gây ra lỗi trang.
- Số lượng lỗi trang xảy ra sẽ tăng lên khi số lượng khung trang sử dụng tăng. Hiện tượng này gọi là **nghịch lý Belady**.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (12)

2. Thuật toán tối ưu: Thay thế trang lâu được sử dụng nhất trong tương lai.

Ví dụ: Cũng sử dụng choỗi truy xuất như VD trước, sử dụng 3 khung trang, khởi đầu đều trống.

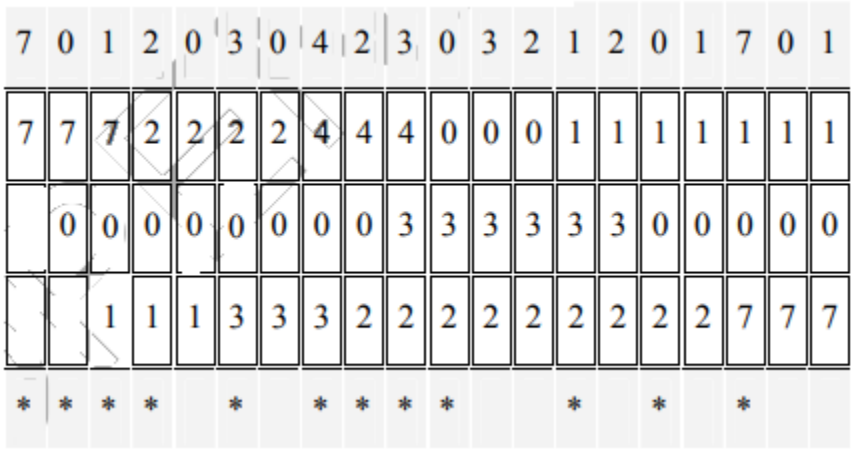
7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
	0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
		1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
*	*	*	*		*		*			*			*				*		

Thảo luận: thuật toán này đảm bảo số lượng lỗi trang phát sinh là thấp nhất, nó cũng không gánh chịu nghịch lý Belady. Tuy nhiên, đây là một thuật toán không khả thi trong thực tế, vì không thể biết trước chuỗi truy xuất của tiến trình.

4.6. QUẢN LÝ BỘ NHỚ 4.6.5. Bộ nhớ ảo (13)

3. Thuật toán LRU (Least-Rece ntly-Used): với mỗi trang, ghi nhận thời điểm cuối cùng trang được truy cập, trang được chọn để thay thế sẽ là trang lâu nhất chưa được truy xuất.

Ví dụ: Cũng sử dụng choỗi truy xuất như VD trước, sử dụng 3 khung trang, khởi đầu đều trống.



Thuật toán này đòi hỏi phải được cơ chế phần cứng hỗ trợ để xác định một thứ tự cho các trang theo thời điểm truy xuất cuối cùng. Có thể cài đặt theo một trong hai cách:

1. **Sử dụng bộ đếm**
2. **Sử dụng Stack**

4.6. QUẢN LÝ BỘ NHỚ
4.6.6. Cấp phát khung – Thang thế trang (1)

4.6.6. Cấp phát khung – Thang thế trang

- Với mỗi tiến trình, cần phải cấp phát một số lượng khung trang tối thiểu nào đó để tiến trình có thể hoạt động. Số khung trang tối thiểu này được quy định bởi kiến trúc của máy tính.
- Số khung trang tối thiểu được quy định bởi kiến trúc máy tính, trong khi số khung trang tối đa được xác định bởi dung lượng bộ nhớ vật lý có thể sử dụng.

1. Cấp phát số khung trang

Có ba cách cấp phát số lượng khung trang là:

- Cấp phát ngang bằng (Equal Allocation)
- Cấp phát theo tỷ lệ kích thước (Proportional Allocation)
- Cấp phát theo độ ưu tiên (Priority Allocation)

4.6. QUẢN LÝ BỘ NHỚ

4.6.6. Cấp phát khung – Thang thế trang (2)

- ***Cấp phát ngang bằng:*** Nếu có m khung trang và n tiến trình, mỗi tiến trình được cấp m/n khung trang. Phương pháp này đơn giản nhưng không hiệu quả.
- ***Cấp phát theo tỷ lệ kích thước:*** Tùy vào kích thước của tiến trình để cấp phát số khung trang. Gọi:
  - s_i là kích thước của tiến trình p_i
  - $S = \sum si$ là tổng kích thước của tất cả tiến trình
  - m là số lượng khung trang có thể sử dụng
  - Khi đó, tiến trình p_i sẽ được cấp phát a_i khung trang: $a_i = \frac{s_i}{S} \times m$
- ***Cấp phát theo độ ưu tiên:*** Số lượng khung trang cấp cho tiến trình phụ thuộc vào độ ưu tiên của tiến trình. Tiến trình có độ ưu tiên cao sẽ được cấp nhiều khung hơn để tăng tốc độ thực hiện.

4.6. QUẢN LÝ BỘ NHỚ
4.6.6. Cấp phát khung – Thang thế trang (3)

❖ **Thay thế trang**

Có hai cách để thay thế trang:

- ***Thay thế toàn cục*** (Global Replacement): Chọn trang "nạn nhân" từ tập tất cả các khung trang trong hệ thống, khung trang đó có thể đang được cấp phát cho một tiến trình khác.
  - Ví dụ: có thể chọn trang của tiến trình có độ ưu tiên thấp hơn làm trang nạn nhân.
  - Thuật toán thay thế toàn cục cho phép hệ thống có nhiều khả năng lựa chọn hơn, số khung trang cấp cho một tiến trình có thể thay đổi, nhưng các tiến trình không thể kiểm soát được tỷ lệ phát sinh lỗi trang của mình.
- ***Thay thế cục bộ:*** Chỉ chọn trang thay thế trong tập các khung trang được cấp cho tiến trình phát sinh lỗi trang.
  - Khi đó số khung trang cấp cho một tiến trình sẽ không thay đổi.

4.6. QUẢN LÝ BỘ NHỚ
4.6.6. Cấp phát khung – Thang thế trang (4)

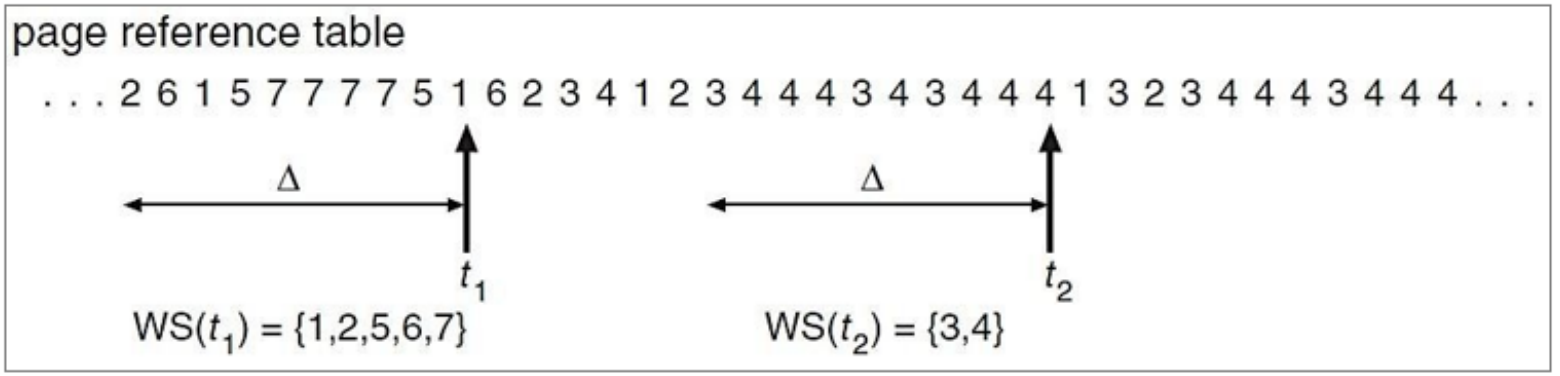
❖ Trì trệ hệ thống (Thrashing)

- Khi tiến trình không có đủ các khung trang để chứa những trang cần thiết cho việc xử lý, nó sẽ thường xuyên phát sinh các lỗi trang.
- Vì thế phải dùng đến rất nhiều thời gian sử dụng CPU để thực hiện thay thế trang.
- HĐH thấy hiệu quả sử dụng CPU thấp sẽ tăng mức độ đa chương dẫn đến trì trệ toàn bộ hệ thống.
- Để ngăn cản tình trạng trì trệ này xảy ra, cần phải cấp cho tiến trình đủ các khung trang cần thiết để hoạt động.
- Vấn đề là làm sao biết được mỗi tiến trình cần bao nhiêu trang?

4.6. QUẢN LÝ BỘ NHỚ
4.6.6. Cấp phát khung – Thang thế trang (5)

❖ Mô hình tập làm việc (Working-Set Model)

- Tập làm việc của tiến trình tại thời điểm t là tập các trang được tiến trình truy xuất đến trong Δ lần truy cập cuối cùng tính tại thời điểm t.



Hình 5.23. Mô hình tập làm việc

4.6. QUẢN LÝ BỘ NHỚ
4.6.6. Cấp phát khung – Thang thế trang (6)

❖Mô hình tập làm việc

Gọi:

- WSSi(Δ, t) là số phần tử của tập working-set của tiến trình Pi tại thời điểm t ;
- m là số khung trang trống;
- ΣWSSi là tổng số khung trang yêu cầu cho toàn hệ thống

➢ HĐH giám sát working-set của mỗi tiến trình Pi và tại thời điểm t sẽ cấp phát cho tiến trình Pi số khung trang bằng với số phần tử trong tập làm việc $(WSS_i)(\Delta, t - 1)$

➢ Nếu tổng số khung trang yêu cầu của các tiến trình trong hệ thống vượt quá các khung trang có thể sử dụng (D>m), thì sẽ xảy ra tình trạng hệ thống trì trệ. Khi đó HĐH chọn một tiến trình để tạm dừng, giải phóng các khung trang của tiến trình đã chọn để các tiến trình khác có đủ khung trang hoàn tất công việc.

4.6. QUẢN LÝ BỘ NHỚ

4.6.6. Cấp phát khung – Thang thế trang (6)

❖Cấu trúc chương trình

- Số lỗi trang có khi phụ thuộc vào ngôn ngữ lập trình, nên khi lập trình ta cần chú ý để chương trình có thể thực hiện nhanh hơn.

  Ví dụ: xét chương trình sau

```
int a[128][128];
for (i=0; i<128;i++)
for (j=0; j<128; j++)
a[i][j]=0;
```

- Giả sử trang có kích thước 128 byte và tiến trình được cấp 2 khung trang: khung trang thứ nhất chứa mã tiến trình, khung trang còn lại được khởi động ở trạng thái trống.
- Trong Pascal, C thì mảng lưu theo hàng, mỗi hàng chiếm 1 trang bộ nhớ nên tổng số lỗi trang phát sinh là 128.
- Nhưng trong Fortran thì mảng lưu theo cột, do đó số lỗi trang sẽ là 128 x 128 = 16384

HẾT CHƯƠNG 4