BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ I NĂM HỌC 2023 – 2024

ĐOÀN KHOA
CÔNG NGHỆ PHẦN MỀM
BAN HỌC TẬP

Sharing is learning

BAN HỌC TẬP

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

CONTACT

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit

TRAINING

HỆ ĐIỀU HÀNH

Thời gian: 19:30 thứ 4 ngày 03/01/2024

Địa điểm: Microsoft Teams

Trainers: Lê Dương Minh Thiên – KHMT2022.4

Nguyễn Hoài Như – KTPM2022.2

Sharing is learning

Nội dung

Chương 5. ĐỒNG BỘ

Chương 6. DEADLOCK

Chương 7. QUẢN LÍ BỘ NHỚ

Chương 8. BỘ NHỚ ẢO

Chương 9. HỆ ĐIỀU HÀNH LINUX VÀ WINDOWS

Chương 5. ĐỒNG BỘ

Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

2. Phân loại giải pháp:

Busy – waiting:

- Giải thuật Peterson
- Giải thuật Bakery
- Giải thuật Swap
- Cấm ngắt
- Chỉ thị TSL

Sleep & wake up:

- Semaphore
- Monitors
- Message

Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

Vấn đề cần đồng bộ:

- Khảo sát các process/thread thực thi đồng thời và chia sẻ dữ liệu (qua shared memory, file).
- Nếu không có sự kiểm soát khi truy cập các dữ liệu chia sẻ thì có thể đưa đến ra trường hợp không nhất quán dữ liệu (data inconsistency).
- Để duy trì sự nhất quán dữ liệu, hệ thống cần có cơ chế bảo đảm sự thực thi có trật tự của các process đồng thời.

Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

- Giới thiệu bài toán Producer-Consumer

Producer

```
while (true) {
     /* produce an item in next_produced */

     while (count == BUFFER_SIZE)
        ; /* do nothing */

     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
     count++;
}
```

Consumer

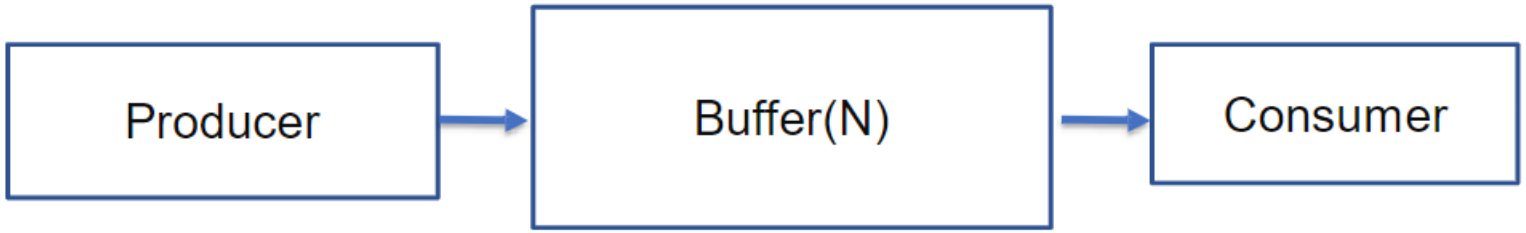
```
while (true) {
     while (count == 0)
        ; /* do nothing */

     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;
     count--;

     /* consume the item in next_consumed */
}
```

Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

- Giới thiệu bài toán Producer-Consumer
- Vấn đề đặt ra:

- P không được ghi dữ liệu vào buffer đã đầy
- C không được đọc dữ liệu từ buffer đang trống
- P và C không được thao tác trên buffer cùng lúc



Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

- Giới thiệu bài toán Producer-Consumer
- Vấn đề đặt ra:
- Vấn đề critical section:

• Trong mỗi process có những đoạn code có chứa các thao tác lên dữ liệu chia sẻ . Đoạn code này được gọi là **vùng tranh chấp** (critical section, CS).

• **Vấn đề Critical Section**: phải bảo đảm sự loại trừ tương hỗ (mutual exclusion, mutex), tức là khi một process đang thực thi trong vùng tranh chấp, không có process nào khác đồng thời thực thi các lệnh trong vùng tranh chấp.

Chương 5. ĐỒNG BỘ

1. Tổng quan về đồng bộ

- Giới thiệu bài toán Producer-Consumer
- Vấn đề đặt ra:
- Vấn đề critical section
- Yêu cầu lời giải:

- **Loại trừ tương hỗ (Mutual exclusion):** Khi một process P đang thực thi trong vùng tranh chấp (CS) của nó thì không có process Q nào khác đang thực thi trong CS của Q
- **Progress:** Một tiến trình tạm dừng bên ngoài miền giăng không được ngăn cản các tiến trình khác vào miền giăng
- **Chờ đợi giới hạn (Bounded waiting):** Mỗi process chỉ phải chờ để được vào vùng tranh chấp trong một khoảng thời gian có hạn định nào đó. Không xảy ra tình trạng đói tài nguyên (starvation).

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

BUSY waiting	Sleep and Wakeup
• Tiếp tục tiêu thụ CPU trong khi chờ đợi vào miền giăng • Không đòi hỏi sự trợ giúp của Hệ điều hành ➢ Sử dụng các biến cờ hiệu ➢ Sử dụng việc kiểm tra luân phiên ➢ Giải pháp của Peterson ➢ Cấm ngắt ➢ Chỉ thị TSL	• Từ bỏ CPU khi chưa được vào miền giăng • Cần Hệ điều hành hỗ trợ ➢ Semaphore ➢ Monitor ➢ Message

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Busy – waiting:**

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Giải thuật Peterson:** Là kết hợp hai ý tưởng quan trọng trong giải thuật "Luân phiên" và "Cờ hiệu" (áp dụng cho 2 quá trình cùng lúc)

Giải thuật Luân phiên	Giải thuật Cờ hiệu
-Áp dụng cho chỉ hai quá trình cùng một lúc -Biến chia sẻ: *turn*	

do {
while (turn != i);
critical section
turn = j;
remainder section
} **while** (1);
	do {
	flag[i] = true; /* Pi "sẵn sàng" vào CS */
	while (flag[j]); /* Pi "nhường" Pj */
	critical section
	flag[i] = false;
	remainder section
	} **while** (1);

-Thỏa tính chất **Mutual exclution** -Không thỏa **Progress & Bounded Waiting** vì tính chất "strict alternation" của giải thuật	-Thỏa tính chất **Mutual exclution** -Không thỏa tính chất **Progress**

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Giải thuật Peterson:** Là kết hợp hai ý tưởng quan trọng trong giải thuật "Luân phiên" và "Cờ hiệu" (áp dụng cho 2 quá trình cùng lúc)

- Đặc điểm Peterson: dùng cả biến turn và flag

P_0

```
{
      while (true){
         flag[0] = true;
         turn = 1;
         while (flag[1] && turn = = 1);
         /* critical section */
         flag[0] = false;
         /* remainder section */
      }
}
```

P_1

```
{
      while (true){
         flag[1] = true;
         turn = 0;
         while (flag[0] && turn = = 0);
         /* critical section */
         flag[1] = false;
         /* remainder section */
      }
}
```

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Giải thuật Peterson:** Là kết hợp hai ý tưởng quan trọng trong giải thuật "Luân phiên" và "Cờ hiệu" (áp dụng cho 2 quá trình cùng lúc)\

- Đặc điểm Peterson: dùng cả biến turn và flag

```
while (true) {
    flag[i] = true;
    turn = j;
    while (flag[j] && turn == j)
        ;

        /* critical section */

    flag[i] = false;

        /*remainder section */
}
```

Thỏa 3 tính chất:

- Biến turn chỉ có thể là 0 hoặc 1 nên tại mỗi thời điểm chỉ có một process ở trong CS=> thỏa Mutual Exclusion
- Quá trình Pi có thể bị ngăn chặn vào miền tương trục khi flag[j] = true & turn = j .
- Khi Pi ra khỏi miền tương trục thì sẽ gán flag[i] = false

=> thỏa Progress & Bounded Waiting

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Giải thuật Bakery**

```
boolean     choosing[ n ]; /*  initially, choosing[ i ] = false  */
int         num[ n ];      /*  initially, num[ i ] = 0           */

do {
      choosing[ i ] = true;
      num[ i ]      = max(num[0], num[1],..., num[n - 1]) + 1;
      choosing[ i ] = false;
      for (j = 0; j < n; j++) {
          while (choosing[ j ]);
          while ((num[ j ] != 0) && (num[ j ], j) < (num[ i ], i));
      }
         critical section
      num[ i ] = 0;
         remainder section
} while (1);
```

- Mỗi process nhận một con số.Process nào giữ số nhỏ nhất thì được vào CS
- Nếu cùng số thì so sánh số thứ tự

VD: nếu Pi và Pj cùng số thì (i < j) => Pi vào trước

- Khi ra khỏi CS, Pi đặt lại bằng 0
- Cơ chế cấp số cho các process: tăng dần

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Giải thuật Swap**

```
void Swap(boolean *a,
          boolean *b) {
   boolean temp = *a;
   *a = *b;
   *b = temp;
}
```

```
Process Pi
   do {
        key = true;
        while (key == true)
              Swap(&lock, &key);
           critical section
        lock = false;
           remainder section
   } while (1)
```

– **Đặc điểm:**

- Biến chia sẻ lock (khởi tạo false)
- Biến cục bộ key
- Process Pi nào thấy giá trị lock = false thì được vào CS
- Process Pi vào CS sẽ cho lock = true để ngăn các process khác

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Cấm ngắt**

<table>
<tr><th colspan="2">Cấm ngắt</th></tr>
<tr><td>Trong hệ thống uniprocessor</td><td>Trong hệ thống multiprocessor</td></tr>
<tr><td colspan="2">Process P_i:
do {
disable_interrupts();
critical section
enable_interrupts();
remainder section
} while (1);
disable_interrupts(): Tiến trình hiện tại ngăn các tiến trình khác vào vùng tranh chấp
enable_interrupts(): Cho phép những tiến trình khác tiến vào vùng tranh chấp</td></tr>
<tr><td>-Mutual exclution được đảm bảo
-Gây ảnh hưởng đến các thành phần khác của hệ thống có sử dụng ngắt như system clock
-Cần phải liên tục tạm dừng và phục hồi ngắt dẫn đến hệ thống tốn chi phí quản lý và kiểm soát</td><td>-Mutual exclution không được đảm bảo
-Chỉ cấm ngắt tại CPU thực thi lệnh disable_interrupts()
-Các CPU khác vẫn có thể truy cập bộ nhớ chia sẻ</td></tr>
</table>

Chương 5. ĐỒNG BỘ

2. Phân loại giải pháp:

➢ **Chỉ thị TSL (Test and Set Lock)**

```
boolean TestAndSet(boolean *lock) {
    boolean returnValue = *lock;
    *lock = true;
    return returnValue;
}
```

```
do
{
    while(TestAndSet(&Lock));
        CS;
    Lock = false;
        RS;
} while(1);
```

Đặc điểm:

- Đảm bảo được tính chất Mutual Exclution
- Quá trình chọn lựa process Pj vào CS kế tiếp là tùy ý
- Không đảm bảo được điều kiện **Bounded**

Waiting=>Xảy ra tình trạng starvation

Chương 5. ĐỒNG BỘ

Sleep & wake up:

- Ý tưởng:

Hệ điều hành cung cấp hai lệnh SLEEP và WAKEUP

• Nếu tiến trình gọi lệnh SLEEP, hệ điều hành chuyển tiến trình sang ready list, lấy lại CPU cấp cho tiến trình khác

• Nếu tiến trình gọi lệnh WAKEUP, hệ điều hành chọn một tiến trình trong ready list, cho tiến trình đó thực hiện tiếp

• Khi một tiến trình chưa đủ điều kiện vào CS, nó gọi SLEEP để tự khóa, cho đến khi một tiến trình khác gọi WAKEUP để giải phóng cho nó

• Khi ra khỏi CS, tiến trình gọi WAKEUP để đánh thức một tiến trình đang chờ, tạo cơ hội cho tiến trình này vào CS

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

```
typedef struct {
        int value;
        struct process *list;
} semaphore;
```

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Đặc điểm:

- Là công cụ đồng bộ cung cấp bởi OS, không đòi hỏi busy waiting
- Semaphore S là một **biến số nguyên**
- Semaphore chỉ có thể được truy xuất qua hai tác vụ có **tính đơn nguyên** (atomic) và **loại trừ** (mutual exclution)

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Wait(S): giảm giá trị semaphore. Kế đó nếu giá trị này *âm* thì process thực hiện wait sẽ bị *sleep()*

- Dùng để giành tài nguyên
- Hàng đợi là DSLK các **PCB**(Process Control Block)

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Signal(S): tăng giá trị semaphore. Kế đó nếu giá trị này *không dương*, một process *đang sleep* bởi lệnh wait() sẽ được *wakeup()* để thực thi

- Dùng để giải phóng tài nguyên
- Sử dụng cơ chế **FIFO**

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Tránh busy waiting: Khi phải chờ đợi thì process được đặt vào một block queue, trong đó chứa các process đang chờ đợi cùng một sự kiện

Có 2 loại:

- *Counting semaphore*: S là số nguyên
- *Binary semaphore:* S =0 | S = 1

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Semaphore**

- Định nghĩa:

wait

```
wait(semaphore *S) {
        S->value--;
        if (S->value < 0) {
                add this process to S->list;
                sleep();
        }
}
```

signal

```
signal(semaphore *S) {
        S->value++;
        if (S->value <= 0) {
                remove a process P from S->list;
                wakeup(P);
        }
}
```

Hạn chế:

- Dễ xảy ra Deadlock
- Có thể xảy ra Starvation
- Semaphore có thể gây ra hiệu ứng đảo ngược ưu tiên

Chương 5. ĐỒNG BỘ

Sleep & wake up:

➢ **Monitors**

Khái niệm:

• Là một cấu trúc ngôn ngữ cấp cao, có chức năng như semaphore nhưng dễ điều khiển hơn

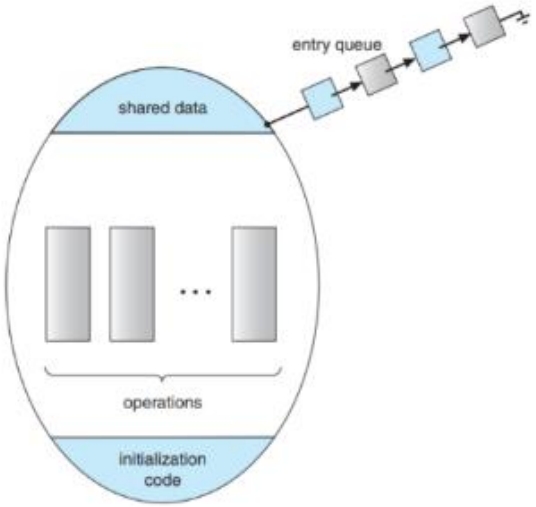
• Có thể hiện thực bằng semaphore

Cấu tạo: Là một module phần mềm, bao gồm:

• Một hoặc nhiều thủ tục (procedure)

• Một đoạn code khởi tạo (initialization code)

• Các biến dữ liệu cục bộ (local data variable)



Mô hình 1 monitor đơn giản

Chương 5. ĐỒNG BỘ

Sleep & wake up:

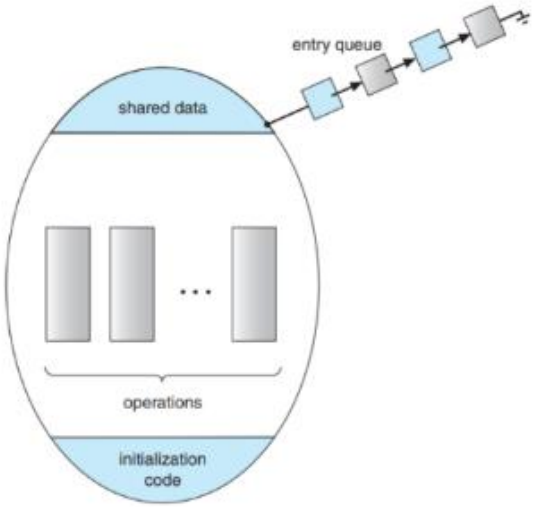
➢ **Monitors**

Đặc tính:

• Dùng các thủ tục của monitor để *truy xuất* local variable

• Process "*vào monitor*" bằng cách gọi một trong các thủ tục đó

• *Chỉ có một* process có thể vào monitor tại *một thời điểm* ⇒ mutualexclusion được bảo đảm.



Mô hình 1 monitor đơn giản

Chương 5. ĐỒNG BỘ

Câu 1: Giải pháp nào sau đây cần sự hỗ trợ của hệ điều hành ?

A. Peterson
B. Bakery
C. Cấm ngắt
D. Semaphore.

Chương 5. ĐỒNG BỘ

Câu 2: Nhóm giải pháp đồng bộ Sleep And Wakeup không có đặc điểm nào dưới đây ?

A. Tiến trình từ bỏ CPU khi chưa được vào vùng tranh chấp
B. Cần sự hỗ trợ từ hệ điều hành
C. Tiến trình rời khỏi vùng tranh chấp sẽ đánh thức tiến trình đã từ bỏ CPU trước đó(nếu có)
D. Được chia thành hai loại Phần mềm và Phần cứng.

Chương 5. ĐỒNG BỘ

Câu 3: Câu nào sau đây đúng ?

A. Loại trừ tương hỗ(Mutal Exclusion – Mutex) là khi một process P đang thực thi trong vùng tranh chấp (CS) của nó thì một process Q nào đó vẫn thực thi được trong vùng tranh chấp (CS) của Q

B. Giải thuật kiểm tra luân phiên thuộc nhóm giải pháp Sleep&Wake up

C. Một tiến trình tạm dừng bên ngoài vùng tranh chấp không được ngăn cản các tiến trình khác vào vùng tranh chấp.

D. Nhóm giải pháp Busy Wating đòi hỏi sự hỗ trợ từ hệ điều hành

Chương 5. ĐỒNG BỘ

Câu 4 : Chọn phát biểu Sai trong các phát biểu dưới đây

A. Giải thuật Peterson và giải thuật Bakery là các giải pháp đồng bộ Busy Waiting sử dụng phần mềm

B. Cấm ngắt là giải pháp đồng bộ busy waiting luôn đảm bảo tính chất loạt trừ tương hỗ.

C. Trong giải thuật Bakery, trước khi vào vùng tranh chấp, mỗi tiến trình sẽ nhận được một con số

D. Trong giải thuật Peterson, tính chất chờ đợi giới hạn luôn được đảm bảo Cấm ngắt không đảm bảo được tính chất loại trừ tương hỗ trên hệ thống đa xử lý

Chương 6. DEADLOCK

Chương 6. DEADLOCK

1. Tổng quan về deadlock
2. Các tính chất của deadlock
3. Phương pháp giải quyết deadlock

Chương 6. DEADLOCK

1. Tổng quan về deadlock

Định nghĩa:

- Một tiến trình gọi là **deadlock** nếu nó đang *đợi một* sự kiện mà sẽ không bao giờ xảy ra (thông thường, có nhiều hơn một tiến trình bị liên quan trong một Deadlock).
- Một tiến trình gọi là **trì hoãn vô hạn định** nếu nó bị trì hoãn một khoảng thời gian dài lặp đi lặp lại trong khi hệ thống đáp ứng cho những tiến trình khác

Ví dụ: Một tiến trình sẵn sàng để xử lý nhưng nó không bao giờ nhận được CPU

Chương 6. DEADLOCK

2. Các tính chất của deadlock

Điều kiện xảy ra deadlocks

- Mutual exclution: một tài nguyên chỉ có thể được giữ bởi một process tại một thời điểm
- Hold & Wait: Một tiến trình đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác giữ
- No preemption: Tài nguyên không thể bị lấy lại mà chỉ có thể được trả lại từ tiến trình đang giữ tài nguyên đó khi nó muốn
- Circular wait: Một tập hợp các process đang chờ đợi lẫn nhau ở dạng vòng tròn (chu trình)

Chương 6. DEADLOCK

2. Các tính chất của deadlock

Đồ thị cấp phát tài nguyên(RAG)

- Là đồ thị có hướng, với tập đỉnh V, và tập cạnh E
- Tập đỉnh V gồm 2 loại:

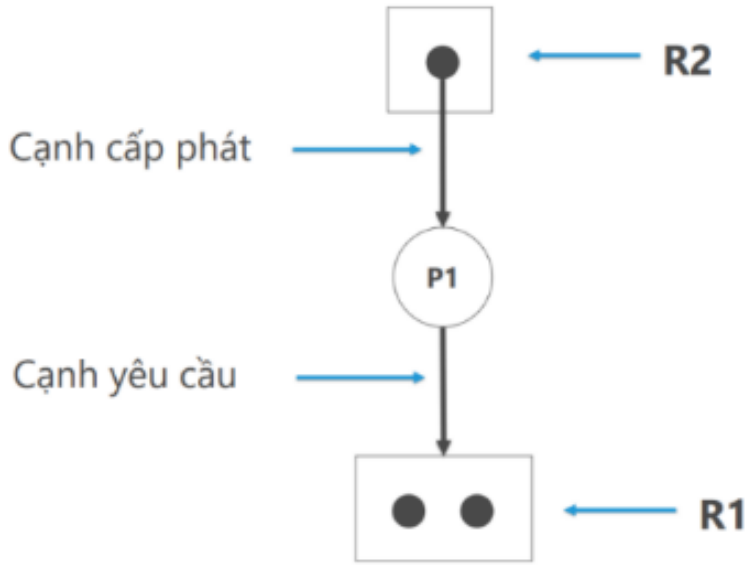
➢ P ={P1,P2,..,Pn} (All process)

➢ R = {R1,R2,...,Rn} (All resource)

- Tập cạnh E gồm 2 loại:

➢ Cạnh yêu cầu: Pi -> Rj

➢ Cạnh cấp phát: Rj > Pi



Chương 6. DEADLOCK

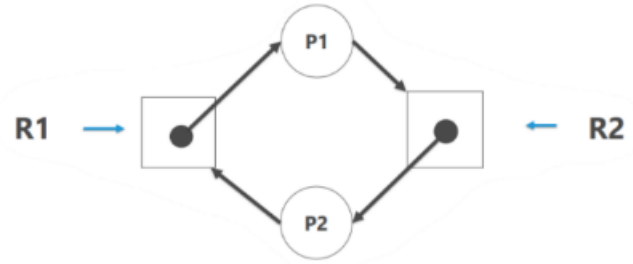
2. Các tính chất của deadlock

Mối liên hệ giữa RAG và deadlocks

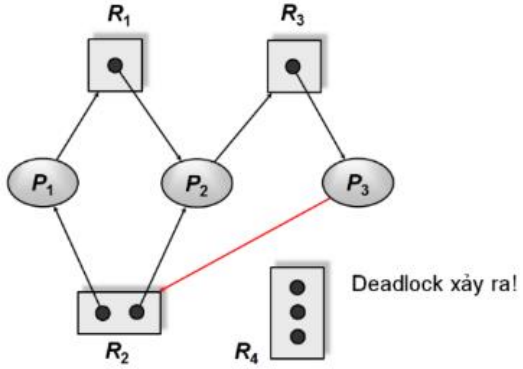
- RAG không chứa chu trình => không có deadlock
- RAG chứa một(hay nhiều) chu trình

➢ Nếu mỗi loại tài nguyên chỉ có một thực thể => deadlock (1)

➢ Nếu mỗi loại tài nguyên có nhiều thực thể => có thể xảy ra deadlock (2)



(2) Một thực thể



(1) Nhiều thực thể

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

- Ngăn chặn deadlocks (deadlocks prevention)
- Tránh deadlocks (deadlocks avoidance)
- Cho phép hệ thống vào trạng thái deadlocks, nhưng sau đó phát hiện deadlocks và phục hồi hệ thống (deadlocks detection and recovery)
- Xem như deadlocks không bao giờ xảy ra trong hệ thống (deadlocks ignorance) (Được sử dụng nhiều nhất trong các cơ chế trên window và Linux,nếu xảy ra deadlocks cần khởi động lại máy).

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

Khác biệt giữa **ngăn** và **tránh** deadlocks:

- **Ngăn deadlocks**: *không cho phép*(ít nhất) một trong 4 điều kiện cần cho deadlocks
- **Tránh deadlock:** Các quá trình cần *cung cấp thông tin về tài nguyên* nó cần để hệ thống cấp phát tài nguyên một cách thích hợp

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.1 Ngăn deadlocks

- **Ngăn Mutual exclution**

- Đối với tài nguyên không chia sẻ (printer): Không làm được
- Đối với tài nguyên chia sẻ (read-only file): Không cần thiết

- **Ngăn Hold&Wait (Hold and Wait)**

- Không giữ: khi yêu cầu tài nguyên thì process không được giữ tài nguyên nào, nếu có thì phải trả lại
- Không chờ: process yêu cầu toàn bộ tài nguyên, nếu đủ OS sẽ cấp phát, nếu không sẽ bị block

Chú ý: Không thể áp dụng trong thực tế vì process không thể xác định được tài nguyên cần thiết trước khi yêu cầu, và process có thể giữ tài nguyên trong một thời gian dài, chưa thể trả ngay

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.1 Ngăn deadlocks

– **Ngăn No preemption**: A có tài nguyên, A yêu cầu tài nguyên khác nhưng chưa được cấp

- Cách 1: Hệ thống lấy lại mọi tài nguyên A đang giữ
- Cách 2: Hệ thống sẽ xem xét tài nguyên A yêu cầu

  - Được giữ bởi một tiến trình khác đang đợi thêm tài nguyên

  => hệ thống lấy lại và cấp cho A

  - Được giữ bởi tiến trình không đợi tài nguyên, A phải đợi

  => tài nguyên của A bị lấy

– **Ngăn Circular wait:** Gán số thứ tự cho tất cả các tài nguyên trong hệ thống. Một process không thể yêu cầu một tài nguyên ít ưu tiên hơn.

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2 Tránh deadlocks

- Tránh deadlocks vẫn ***đảm bảo hiệu suất*** sử dụng tài nguyên **tối đa** đến mức có thể
- Yêu cầu mỗi tiến trình ***khai báo số lượng tài nguyên tối đa*** cần để thực Hiện công việc
- Giải thuật tránh deadlocks sẽ ***kiểm tra trạng thái*** cấp phát tài nguyên để đảm bảo hệ thống không rơi vào deadlocks
- Trạng thái cấp phát tài nguyên được định nghĩa dựa trên ***số tài nguyên còn lại***, ***đã được cấp phát*** và yêu cầu tối đa của các tiến trình

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2 Tránh deadlocks

- Trạng thái safe và unsafe: Một trạng thái của hệ thống gọi là an toàn (safe) nếu tồn tại một chuỗi an toàn

Một chuỗi quá trình <P1,P2,...,Pn> là một **chuỗi an toàn** nếu: Với mọi i = 1,2,...,n yêu cầu tối đa về tài nguyên của Pi có thể được thỏa bởi:

- Tài nguyên mà hệ thống đang có sẵn sàng
- Cùng với tài nguyên mà tất cả các Pj (i<j) đang giữ

➢ *Nếu hệ thống ở trạng thái safe => không deadlocks*

➢ *Nếu hệ thống đang ở trạng thái unsafe => có thể bị deadlocks*

=> Tránh deadlocks bằng cách đảm bảo hệ thống không đi đến trạng thái unsafe

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker

Điều kiện:

- Mỗi tiến trình phải khai báo số lượng thực thể tối đa của mỗi loại tài nguyên mà nó cần
- Khi tiến trình yêu cầu tài nguyên thì có thể phải đợi
- Khi tiến trình đã có được đầy đủ tài nguyên thì phải hoàn trả trong một khoảng thời gian hữu hạn nào đó

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker

1. Tìm Need = Max – Allocation

2. Tìm tiến trình Pi thỏa:

- Pi chưa hoàn thành thực thi

- Needi ≤ Available

- Nếu không còn Pi thỏa => chuyển sang b4

3. Available = Available + Allocationi

Thêm Pi vào chuỗi

Chuyển sang b2

4. Nếu chuỗi có tồn tại đủ hết các P

· Hệ thống tồn tại chuỗi an toàn

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker

Ví dụ:

	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

Available	Chuỗi an toàn
3 3 2	
5 3 2	P_1
7 4 3	P_3
7 4 5	P_4
10 4 7	P_2
10 5 7	P_0

Sharing is learning

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker: Giải thuật Banker yêu cầu tài nguyên cho một tiến trình

1. Nếu requesti ≤ needi thì đến b2. Nếu không, báo lỗi vì tiến trình đã vượt

yêu cầu tối đa

2. Nếu requesti ≤ available thì qua b3. Nếu không, Pi phải chờ vì tài nguyên không còn đủ để cấp phát

3. Giả định cấp phát tài nguyên đáp ứng yêu cầu của Pi bằng cách cập nhật

trạng thái hệ thống như sau:

- available = available – requesti
- allocationi = allocationi+ requesti
- needi = needi – requesti

Sharing is learning

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker:

Ví dụ:

P3 yêu cầu thêm tài nguyên (1,3,1,2) thì hệ thống có đáp ứng không ?

Tiến trình	Allocation				Max			
	R1	R2	R3	R4	R1	R2	R3	R4
P1	3	1	1	2	5	3	4	3
P2	1	1	2	1	3	4	6	1
P3	2	1	4	5	3	5	5	7
P4	3	5	2	2	4	6	4	5
P5	1	3	4	1	1	5	7	2

Available			
R1	R2	R3	R4
4	3	3	5

Giải:

Request P3(1,3,1,2) ≤ Need P3(1,4,1,2)

Request P3(1,3,1,2) ≤ Available P3(4,3,3,5)

Chương 6. DEADLOCK

3. Phương pháp giải quyết deadlock

3.2. Giải thuật Banker:

Ví dụ:

Tiến trình	Allocation				Max				Need				Available			
	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4	R1	R2	R3	R4
P1	3	1	1	2	5	3	4	3	2	2	3	1	3	0	2	3
P2	1	1	2	1	3	4	6	1	2	3	4	0				
P3	3	4	5	7	3	5	5	7	0	1	0	0				
P4	3	5	2	2	4	6	4	5	1	1	2	3				
P5	1	3	4	1	1	5	7	2	0	2	3	1				

Giải:

Request P3(1,3,1,2) ≤ Need P3(1,4,1,2)

Request P3(1,3,1,2) ≤ Available P3(4,3,3,5)

Ta thấy: sau khi cấp phát cho P3 thì **không** thể tìm được bất kì chuỗi an toàn nào trong hệ thống => hệ thống không an toàn. Vậy hệ thống không thể đáp ứng yêu cầu của P3

Chương 6. DEADLOCK

4. Phát hiện và phục hồi Deadlock

Phát hiện Deadlock

- Đối với mỗi loại tài nguyên chỉ có một thực thể

=> Sử dụng wait-for graph

- Đối với mỗi loại tài nguyên có nhiều thực thể

➢ Thực hiện tương tự giải thuật Banker

➢ Thay cột Need bằng cột Request

➢ Liệt kê chuỗi an toàn => Tiến trình không có trong chuỗi an toàn thì deadlock xảy ra tại tiến trình đó

Chương 6. DEADLOCK

4. Phát hiện và phục hồi Deadlock

Phục hồi

- Báo người vận hành
- Hệ thống tự động phục hồi bằng cách bẻ gãy chu trình deadlock
  - Chấm dứt một hay nhiều tiến trình (1)
  - Lấy lại tài nguyên từ một hay nhiều tiến trình (2)

Chương 6. DEADLOCK

Câu 2: Thuật ngữ "deadlock" được hiểu như thế nào là đúng:

A. do thông lượng tiến trình xử lý trên 1 giây quá nhỏ.

B. do xung đột tài nguyên làm treo máy.

C. do thiếu tài nguyên đáp ứng cho các tiến trình cùng yêu cầu

D. là điểm chết của các tiến trình bị khóa.

Chương 6. DEADLOCK

Câu 8: Hệ thống rơi vào trạng thái deadlock khi:

A. tất cả các tiến trình bị deadlock.

B. chỉ cần 1 tiến trình bị deadlock

C. Thỏa một trong bốn điều kiện deadlock.

D. không thu hồi được tài nguyên.

Chương 6. DEADLOCK

Câu 6: Cho các thuật ngữ sau: "Banker","Elphick","Mutual exclusion","No preemption". Thuật ngữ nào là điều kiện gây ra tắc nghẽn?

A. Banker, Mutual exclusion.

B. No preemption, Mutual exclusion

C. Elphick, Banker.

D. No preemption, Elphick, Banker.

Chương 6. DEADLOCK

Câu 12: Chọn câu sai trong các câu sau:

A. Thứ tự quá trình xử lý deadlock: tránh, ngăn ngừa, phát hiện, phục hồi

B. Xác định deadlock có chu trình bằng đồ thị có hướng.

C. Ngăn chặn tắt nghẽn với điều kiện "chiếm giữ và yêu cầu thêm tài nguyên" cả hai giải pháp cấp phát đều vi phạm tính toàn vẹn của dữ liệu.

D. Ngăn chặn tắt nghẽn dạng tài nguyên không thể chia sẻ nhưng cho phép kết xuất người ta dùng spooling điều phối tài nguyên.

Chương 7. QUẢN LÍ BỘ NHỚ

Chương 7. QUẢN LÍ BỘ NHỚ

1. Khái niệm
2. Địa chỉ bộ nhớ
3. Các mô hình quản lý bộ nhớ

Chương 7. QUẢN LÍ BỘ NHỚ

1. Khái niệm

- Một chương trình phải được mang vào trong bộ nhớ chính và đặt nó trong một tiến trình để được xử lý
- Chương trình = tập mã lệnh + dữ liệu
- Trước khi được vào bộ nhớ chính, các tiến trình phải đợi trong một Input Queue
- 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

Chương 7. QUẢN LÍ BỘ NHỚ

1. Khái niệm

Nhiệm vụ của hệ điều hành trong quản lý bộ nhớ:

- 5 nhiệm vụ chính:
  - Cấp phát bộ nhớ cho process
  - Tái định vị (relocation): khi swapping…
  - Bảo vệ: kiểm tra truy xuất có hợp lệ không
  - Chia sẻ: cho phép tiến trình chia sẻ vùng nhớ(code, biến toàn cục…)
  - Kết gán địa chỉ nhớ luận lý vào địa chỉ vật lý
- Mục tiêu: Nạp càng nhiều process vào bộ nhớ càng tốt (tăng tính đa chương)
- Trong hầu hết các hệ thống, **kernel sẽ chiếm một phần cố định trong bộ nhớ,** phần còn lại phần phối cho tiến trình

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

- Địa chỉ vật lý (physical address – địa chỉ thực): là một vị trí thực trong bộ nhớ chính(tính toàn bởi MMU)
- Địa chỉ tuyệt đối(absolute address): địa chỉ tương đương với địa chỉ thực
- Địa chỉ vật lý = địa chỉ frame + offset

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

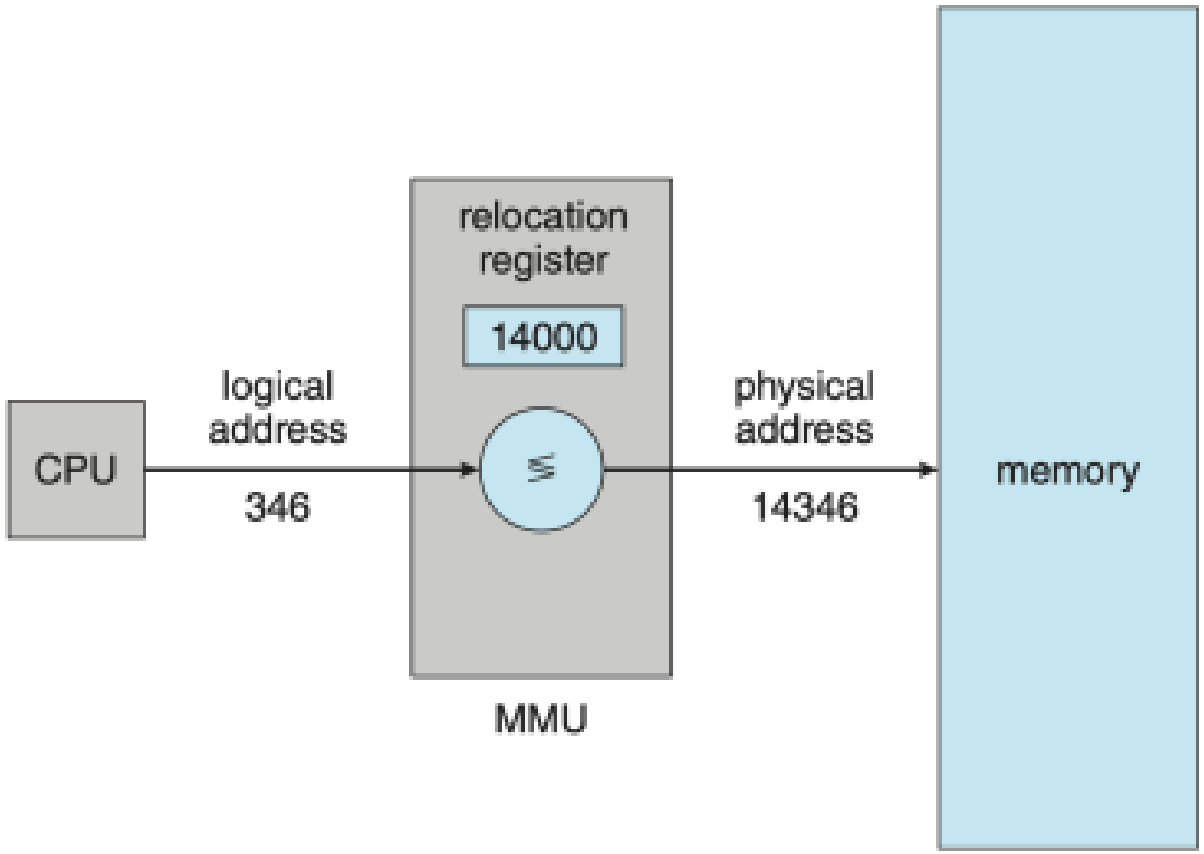
- Địa chỉ luận lý (logical address – virtual address – địa chỉ ảo): vị trí nhớ được diễn tả trong một chương trình(CPU tạo ra)
- Địa chỉ tương đối (relative address): địa chỉ được biểu diễn tương đối so với một vị trí xác định nào đó và không phụ thuộc vào vị trí thực của tiến trình trong bộ nhớ
- Địa chỉ luận lý = địa chỉ page + offset

➔ Để truy cập bộ nhớ, địa chỉ luận lý cần được biến đổi thành địa chỉ vật lý.

Thao tác này cần có phần cứng để đạt hiệu quả cao

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:



Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

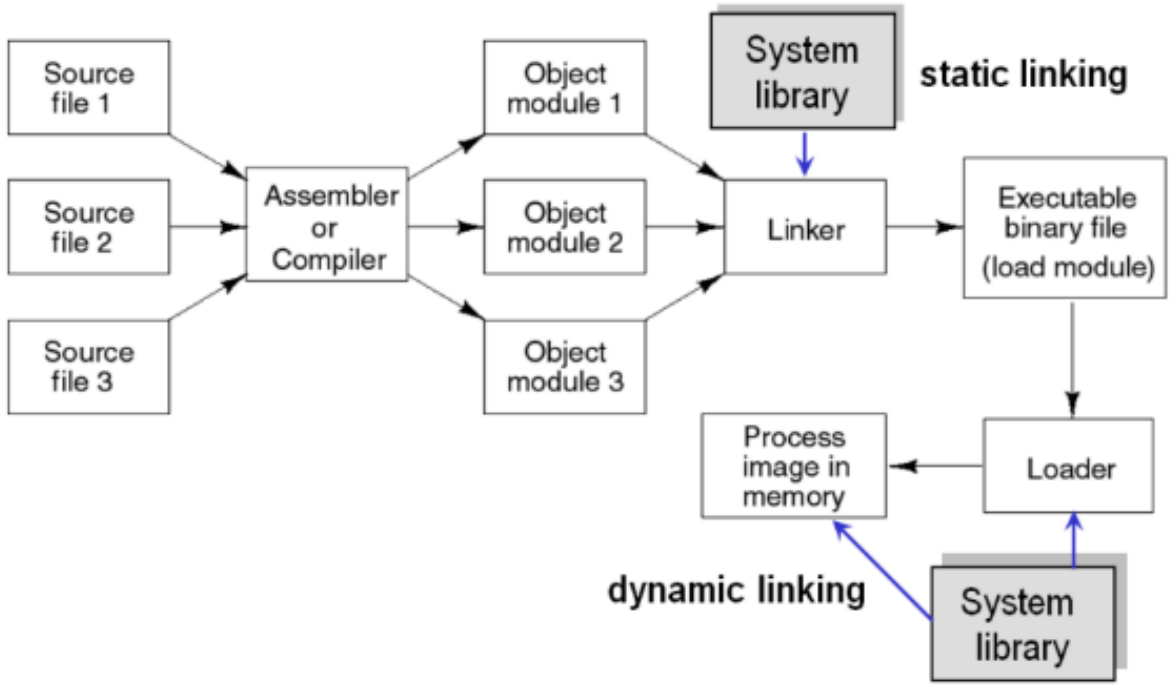
	Địa chỉ logic	Địa chỉ vật lí
Nền tảng	CPU tạo ra	Vị trí trong bộ nhớ
Không gian địa chỉ	Là tập hợp tất cả các ĐCLL do CPU tạo ra liên quan đến một chương trình	Là tập hợp tất cả ĐCVL được ánh xạ tới các ĐCLL tương ứng
Hiển thị	Có thể xem ĐCLL của một chương trình	Không bao giờ có thể xem được ĐCVL của chương trình
Tạo bởi	CPU tạo ra	Tính toán bởi MMU
Truy cập	Có thể dung ĐCLL để truy cập ĐCVL	Có thể gián tiếp truy cập ĐCVL nhưng không thể truy cập trực tiếp
Có thể chỉnh sửa	Có thể chỉnh sửa	Không thay đổi
Còn được gọi là	Địa chỉ ảo	Địa chỉ thực

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ Nạp chương trình vào bộ nhớ

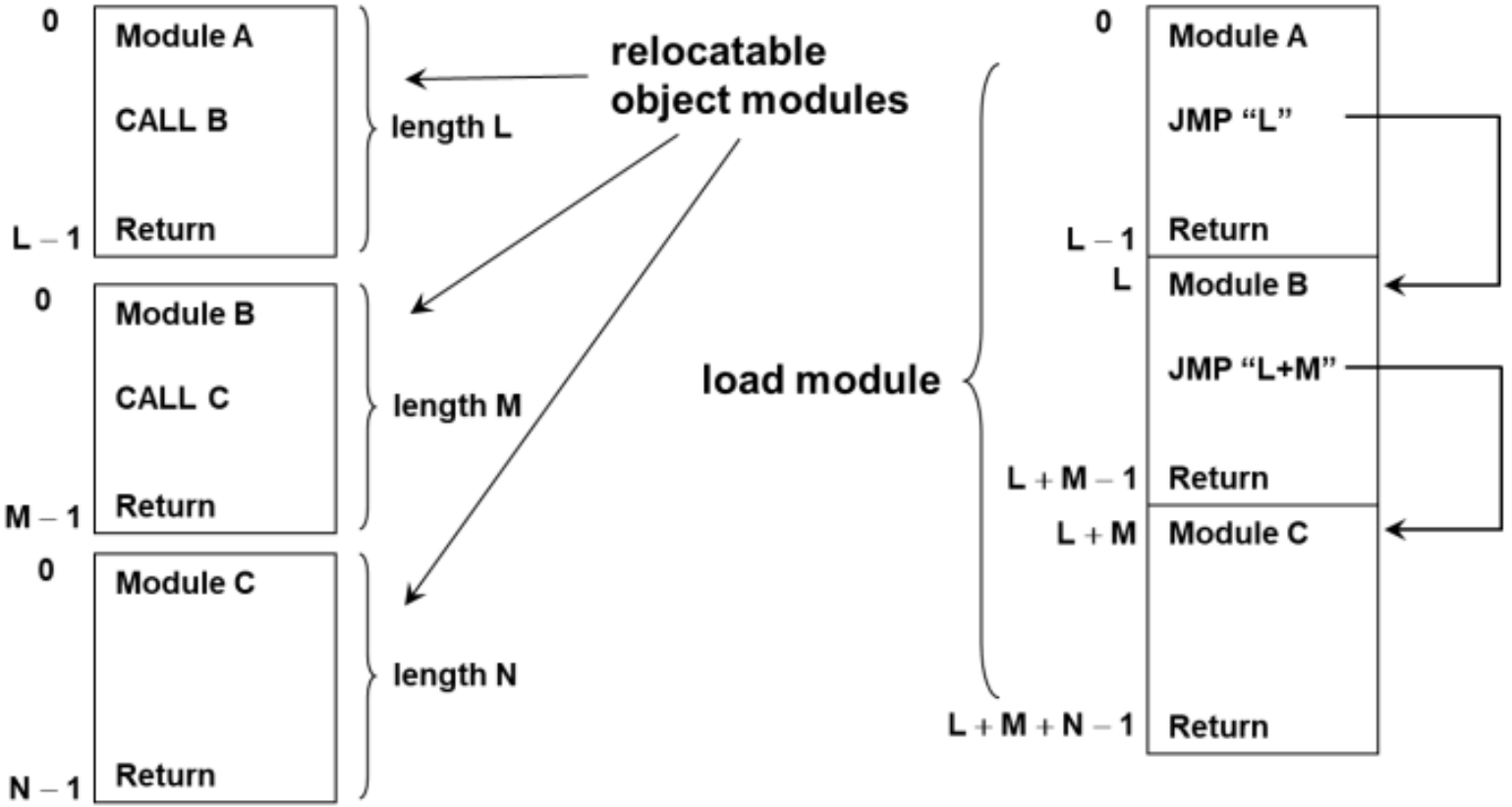
- Linker: kết hợp nhiều object module thành một file nhị phân có khả năng thực thi (file tạo thành gọi là load module)
- Loader: nạp module vào bộ nhớ chính



Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ Cơ chế thực hiện linking



Chương 7. QUẢN LÍ BỘ NHỚ

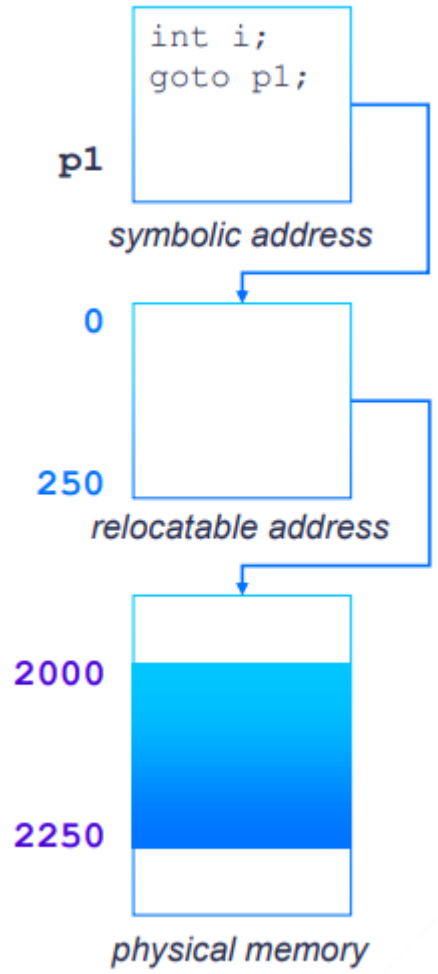
2. Địa chỉ bộ nhớ:

❖ Chuyển đổi địa chỉ

- Là quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác

❖ Biểu diễn địa chỉ nhớ

- Trong source code: symbolic (biến, hằng, pointer…)
- Trong thời điểm biên dịch: thường là địa chỉ khả tái định vị
- Trong thời điểm load/linking: có thể là địa chỉ thực



Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ Địa chỉ lệnh và dữ liệu được chuyển đổi thành địa chỉ thực có thể xảy ra tại ba thời điểm khác nhau.

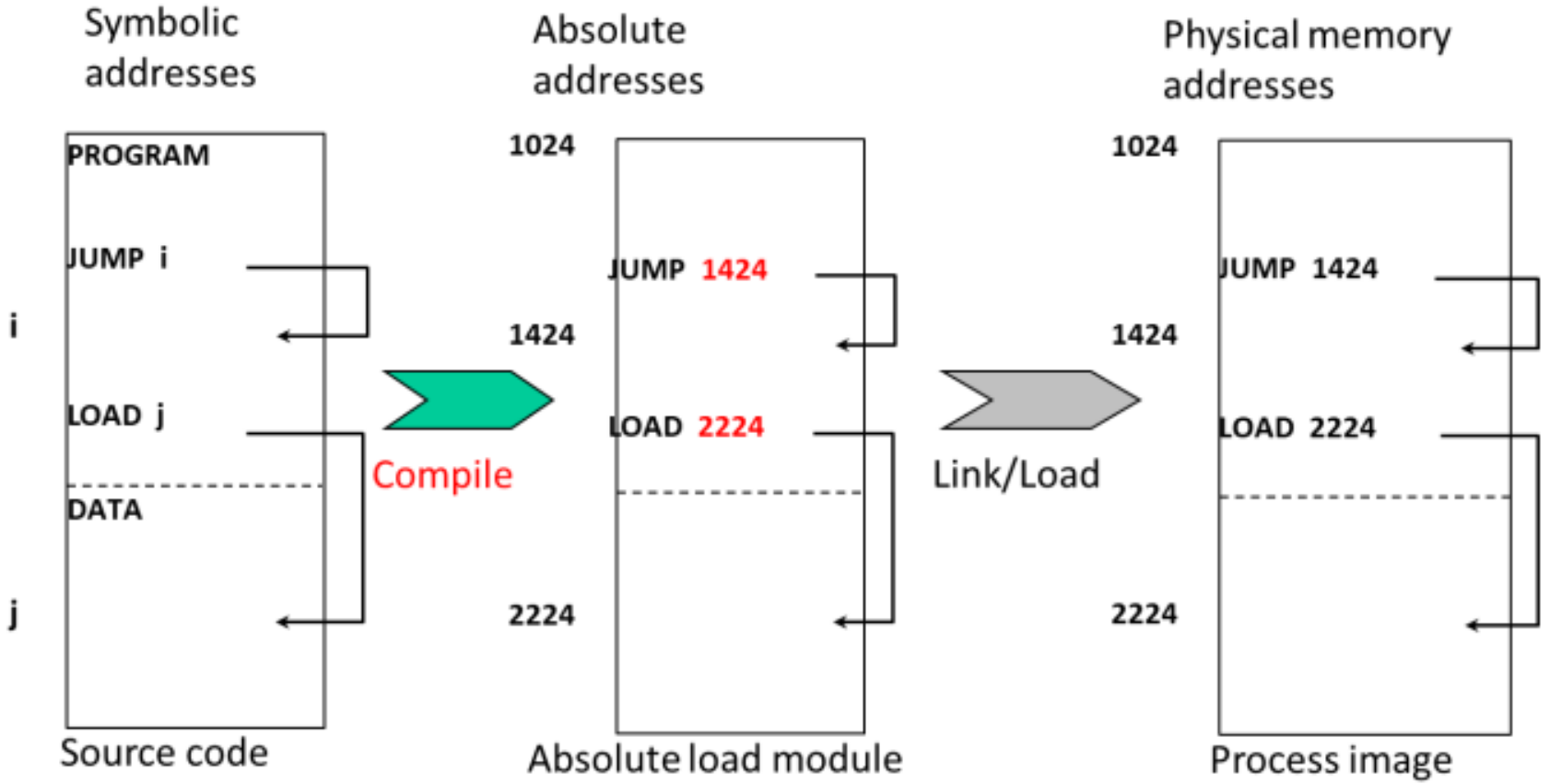
- Compile time: nếu biết trước địa chỉ bộ nhớ của chương trình thì có thể kết gán địa chỉ tuyệt đối lúc biên dịch.
  - ➢ Ví dụ: chương trình .COM của MS-DOS.
  - ➢ Khuyết điểm: phải biên dịch lại nếu thay đổi địa chỉ nạp chương trình.
- Load time: vào thời điểm loading, loader phải chuyển đổi địa chỉ khả tái định vị thành địa chỉ thực dựa trên một địa chỉ nền.
  - ➢ Địa chỉ thực được tính toán vào thời điểm nạp chương trình

➔ Phải tiến hành reload nếu địa chỉ nền thay đổi

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

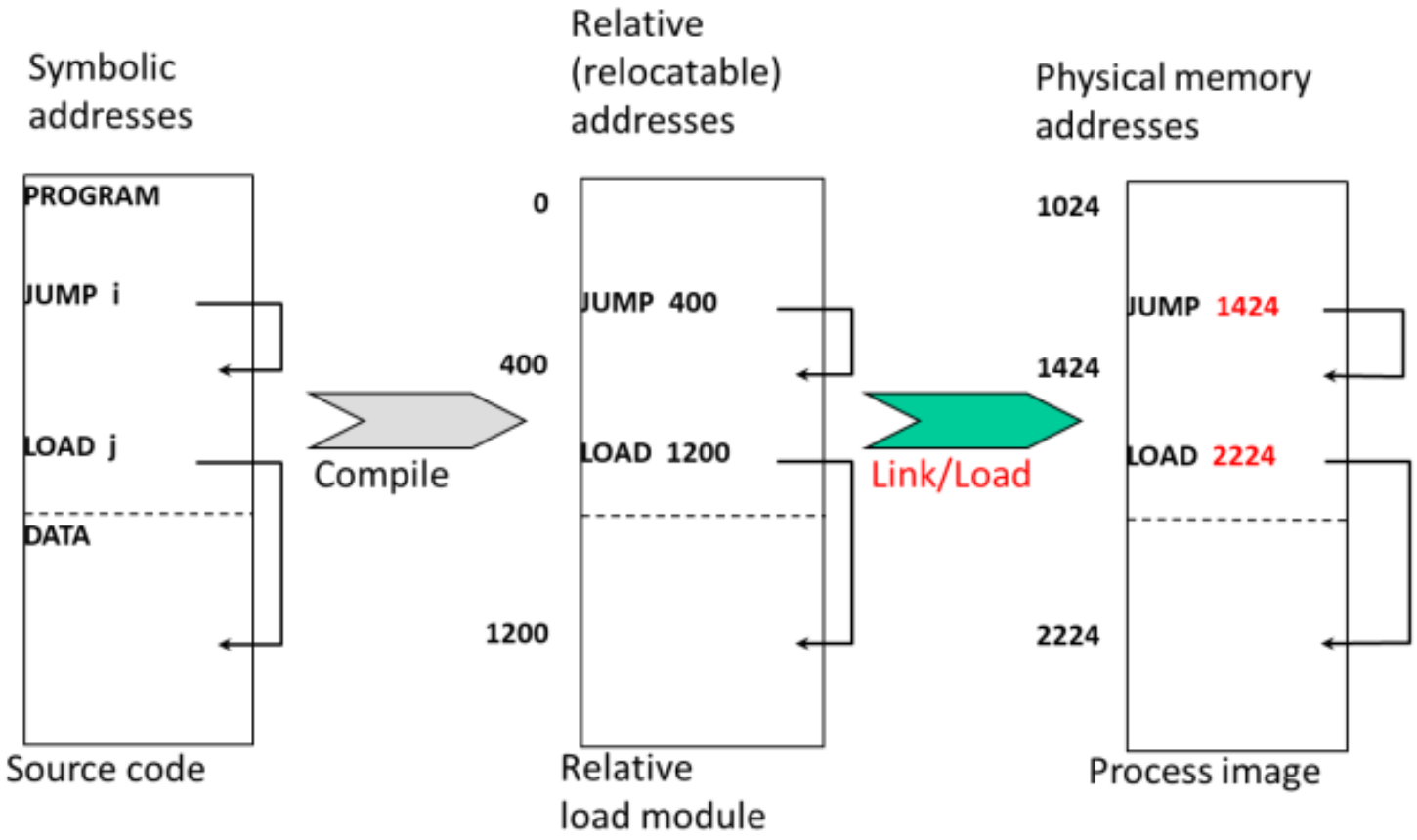
❖ Sinh địa chỉ vào thời điểm dịch



Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ **Sinh địa chỉ vào thời điểm nạp**

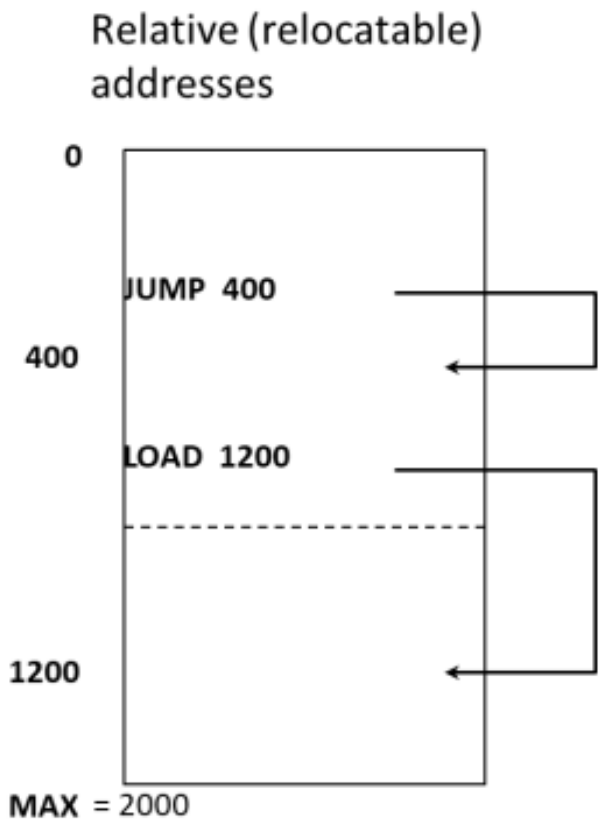


Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ **Excution time:** khi trong quá trình thực thi, tiến trình có thể được di chuyển từ segment này sang segment khác trong bộ nhớ thì quá trình chuyển đổi địa chỉ được trì hoãn đến thời điểm thực thi.

- Cần sự hỗ trợ của phần cứng cho việc ánh xạ địa chỉ
  - ➢ Ví dụ: Trường hợp địa chỉ luận lý là relocatable thì có thể dùng thanh ghi base và limit, …
  - ➢ Sử dụng trong đa số các OS đa dụng trong đó có các cơ chế swapping, paging, segmentation…



Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ **Dynamic linking**

- Quá trình link đến một module ngoài (external module) được thực hiện sau khi đã tạo xong load module (i.e. file có thể thực thi, executable).
  - ➢ Ví dụ trong Windows: module ngoài là các file .DLL còn trong Unix, các module ngoài là các file .so (shared library).
- Load module chứa các stub(đoạn mã nhỏ) tham chiếu (refer) đến routine của external module.
  - ➢ Lúc thực thi, khi stub được thực thi lần đầu (do process gọi routine lần đầu), stub nạp routine vào bộ nhớ, tự thay thế bằng địa chỉ của routine và routine được thực thi.
  - ➢ Các lần gọi routine sau sẽ xảy ra bình thường.
- Stub cần sự hỗ trợ của OS (như kiểm tra xem routine đã được nạp vào bộ nhớ chưa)

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ Dynamic linking

- **Ưu điểm**

➢ Thông thường, external module là một thư viện cung cấp các tiện ích của OS. Các chương trình thực thi có thể dùng các phiên bản khác nhau của external module mà không cần sửa đổi, biên dịch lại.

➢ Chia sẻ mã (code sharing): một external module chỉ cần nạp vào bộ nhớ một lần. Các tiến trình cần dùng external module này thì cùng chia sẻ đoạn mã của external module ⇒ tiết kiệm không gian nhớ và đĩa.

➢ Phương pháp dynamic linking cần sự hỗ trợ của OS trong việc kiểm tra xem một thủ tục nào đó có thể được chia sẻ giữa các tiến trình hay là phần mã của riêng một tiến trình (bởi vì chỉ có OS mới có quyền thực hiện việc kiểm tra này).

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ Dynamic loading: giúp chạy chương trình lớn hơn dung lượng bộ nhớ

- Cơ chế: chỉ khi nào cần được gọi đến thì một thủ tục mới được nạp vào bộ nhớ chính ⇒ tăng độ hiệu dụng của bộ nhớ bởi vì các thủ tục không được gọi đến sẽ không được gọi → không chiếm chỗ trong bộ nhớ.
- Rất hiệu quả trong trường hợp tồn tại khối lượng lớn mã chương trình có tần suất sử dụng thấp, không được sử dụng thường xuyên (ví dụ các thủ tục xử lý lỗi).
- Hỗ trợ từ hệ điều hành
  - ➢ Thông thường, user chịu trách nhiệm thiết kế và hiện thực các chương trình có dynamic loading.
  - ➢ Hệ điều hành chủ yếu cung cấp một số thủ tục thư viện hỗ trợ, tạo điều kiện dễ dàng hơn cho lập trình viên

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ **Overlay: giúp cho bộ nhớ linh động hơn(tính toán dung lượng min để chạy chương trình**

- Ý tưởng: chỉ giữ trong bộ nhớ những lệnh, dữ liệu cần tại mọi thời điểm (module tải…) ➔ giảm không gian nhớ liên tục cho chương trình
- Cơ chế:
  - ➢ Cho phép tổ chức chtr thành các đơn vị chtr(module)
  - ➢ Module luôn tồn tại trong quá trình thực hiện → module chtr chính
  - ➢ Quan hệ độc lập/phụ thuộc chỉ sự có mặt của 1 nhóm module trong bộ nhớ đòi hỏi/không đòi hỏi sự có mặt của 1 nhóm module khác

  ➔Các module độc lập, không cần thiết phải có đồng thời trong bộ nhớ
- Cần đến khi chtr có dung lượng lớn hơn bộ nhớ được cấp phát cho nó

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

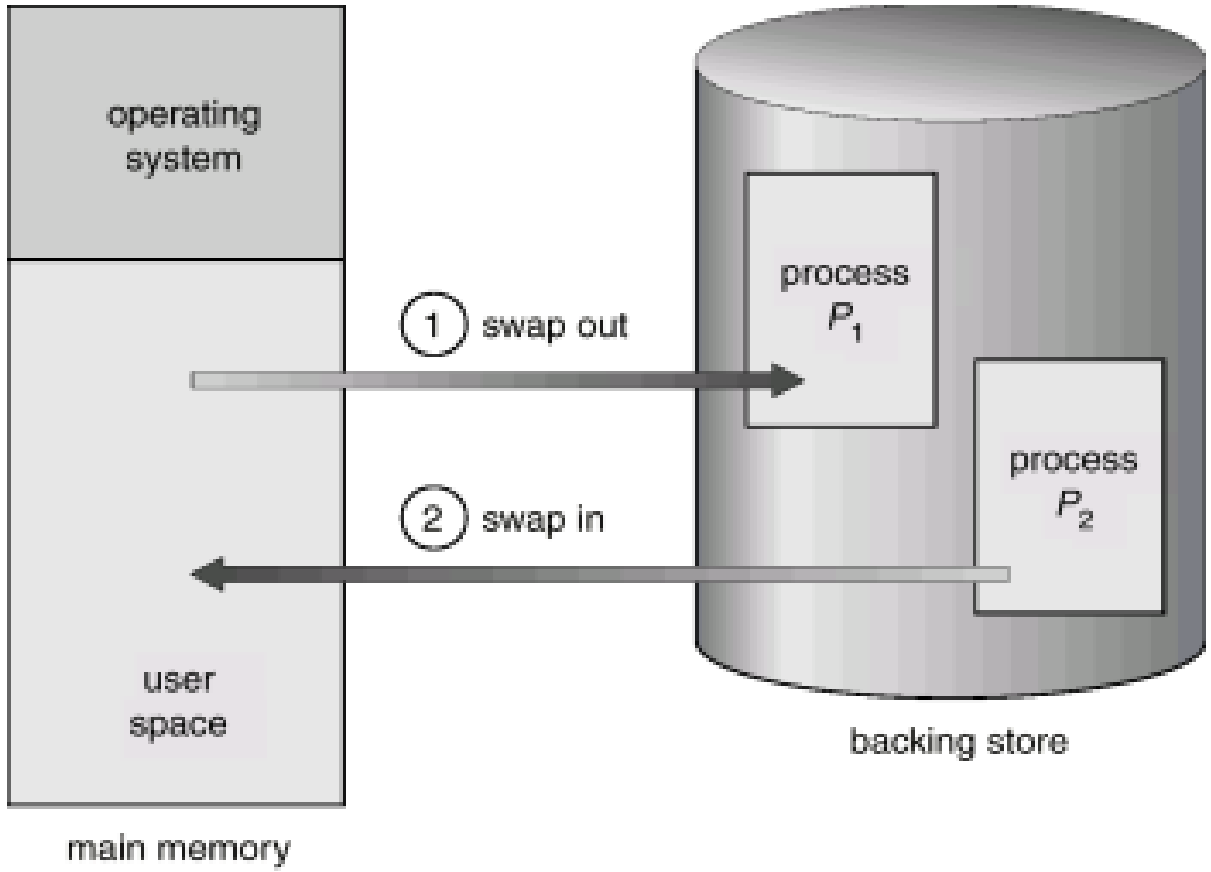
❖ Swapping:

- Một tiến trình có thể tạm thời bị swap ra khỏi bộ nhớ chính và lưu trên một hệ thống lưu trữ phụ (backing store). Sau đó, tiến trình có thể được nạp lại vào bộ nhớ để tiếp tục quá trình thực thi.
- Swapping policy: hai ví dụ
  - ➢ Round-robin: swap out P1 (vừa tiêu thụ hết quantum của nó), swap in P2 , thực thi P3 ,...
  - ➢ Roll out, roll in: dùng trong cơ chế định thời theo độ ưu tiên (priority-based scheduling)
  - ➢ Tiến trình có độ ưu tiên thấp hơn sẽ bị swap out nhường chỗ cho tiến trình có độ ưu tiên cao hơn mới đến được nạp vào bộ nhớ để thực thi.
- Phần lớn thời gian hoán đổi là thời gian chuyển dữ liệu; tổng thời gian chuyển tỷ lệ thuận với dung lượng bộ nhớ hoán đổi
- Hiện nay, ít hệ thống sử dụng cơ chế swapping trên.

Chương 7. QUẢN LÍ BỘ NHỚ

2. Địa chỉ bộ nhớ:

❖ **Swapping:**

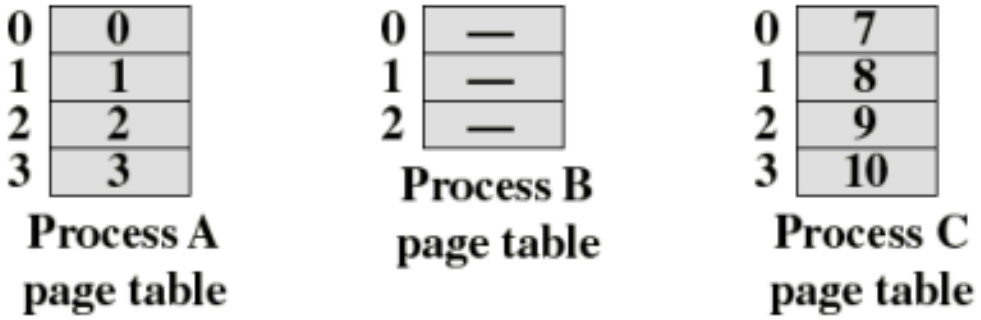


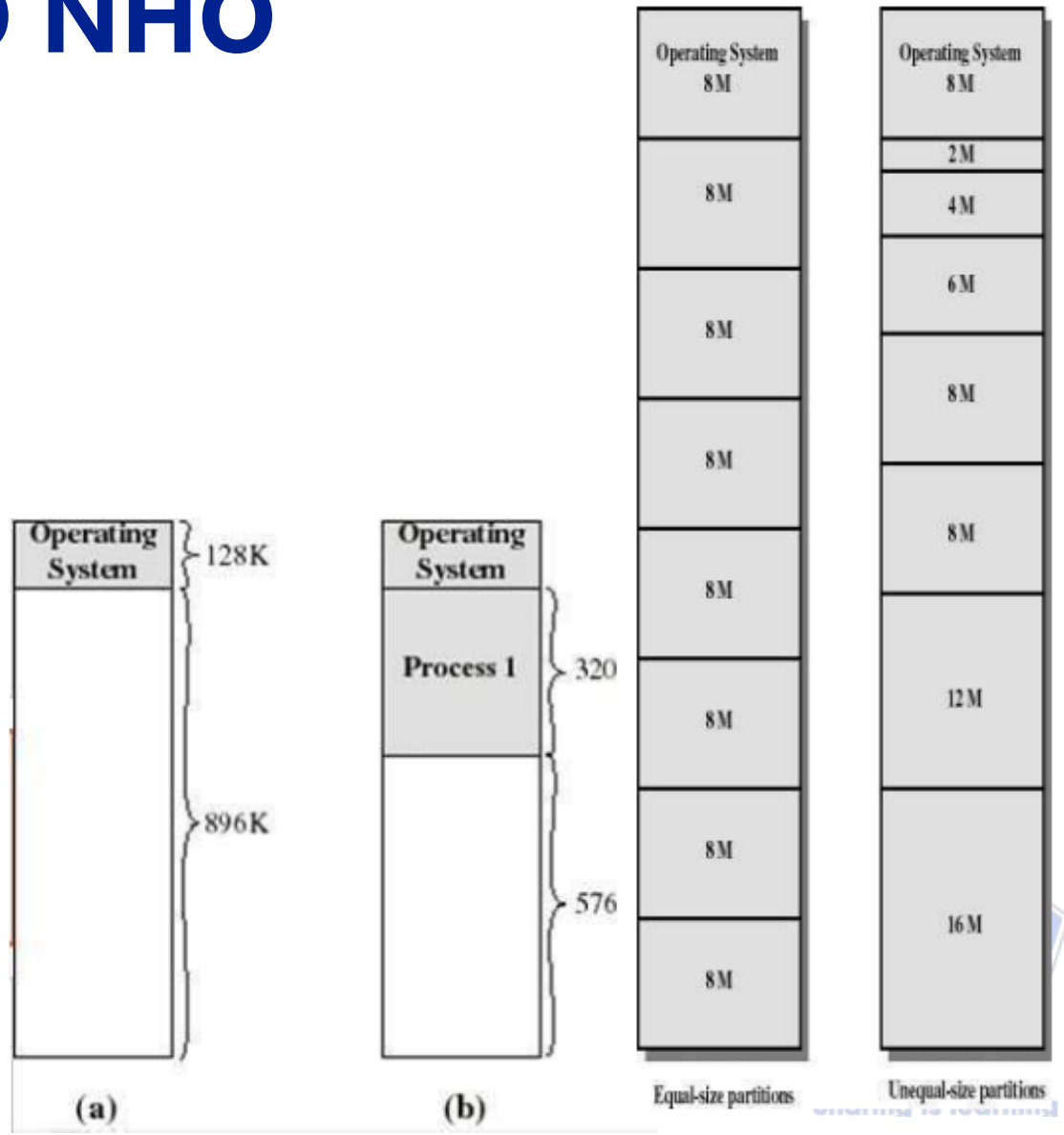
Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Một số cơ chế quản lý bộ nhớ

- Phân chia cố định (fixed partitioning)
- Phân chia động (dynamic partitioning)
- Phân trang đơn giản (simple paging)





Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Hiện tượng phân mảnh bộ nhớ (fragmentation):

- Phân mảnh ngoại (external fragmentation): tổng kích thước không gian bộ nhớ trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên không gian nhớ này không liên tục

➔ *Giải pháp*: Dùng cơ chế kết khối (compaction) để gom vùng nhớ trống lại thành vùng nhớ liên tục. Kết khối chỉ có thể thực hiện nếu sự tái định bị là động và thực hiện trong execution time

- Phân mảnh nội (internal fragmentation): Kích thước vùng nhớ cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu

➔ *Giải pháp:* Dùng chiến lược placement

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia cố định (Fixed partitioning)

- Bộ nhớ chính chia thành nhiều phần, có kích thước bằng hoặc khác nhau
- Process nào nhỏ hơn kích thước partition thì có thể nạp vào partition đó
- Nếu process có kích thước lớn hơn ➔overlay

➔*Nhận xét:* Kém hiệu quả do bị phân mảnh nội

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

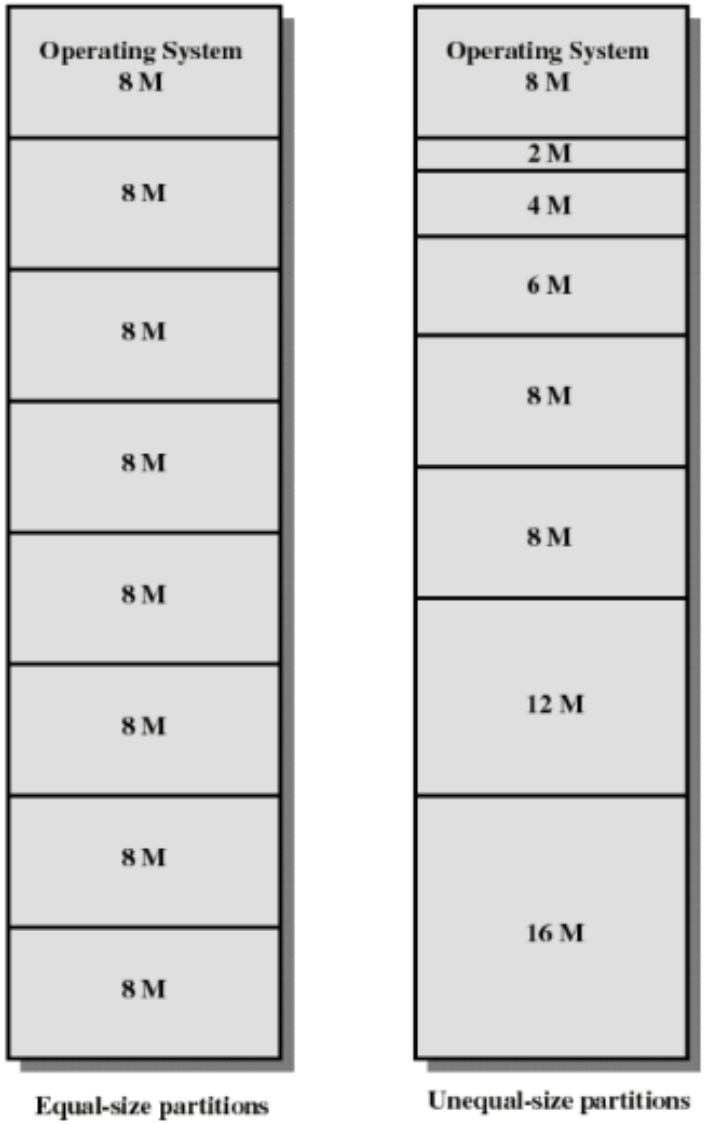
Phân chia cố định (Fixed partitioning)

Chiến lược placement

- Với partition có **cùng** kích thước:

➢ Còn partition trống ➔ nạp vào

➢ Không còn partition trống ➔ Swap process đang bị blocked ra bộ nhớ phụ, nhường chỗ cho process mới



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia cố định (Fixed partitioning)

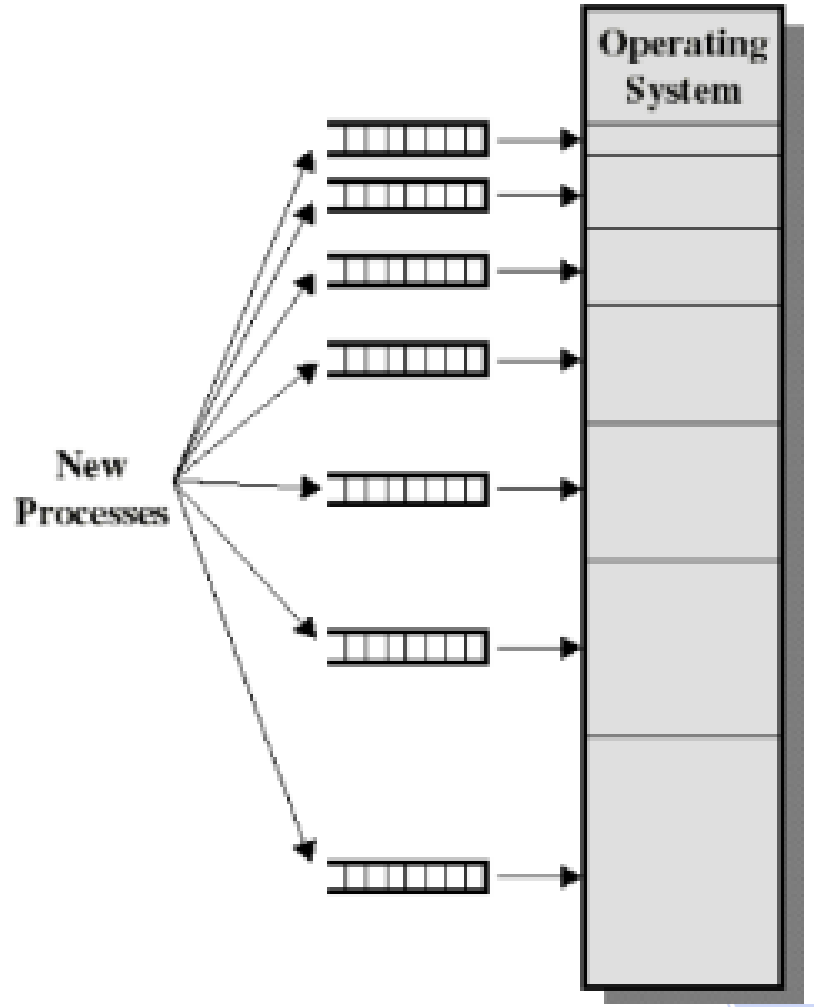
Chiến lược placement

- Với partition **khác** kích thước:
- **Giải pháp 1**: Sử dụng nhiều hàng đợi

➢ Mỗi process xếp vào partition nhỏ nhất phù hợp

➢ Ưu điểm: giảm thiểu phân mảnh nội

➢ Nhược điểm: có thể có hàng đợi trống



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia cố định (Fixed partitioning)

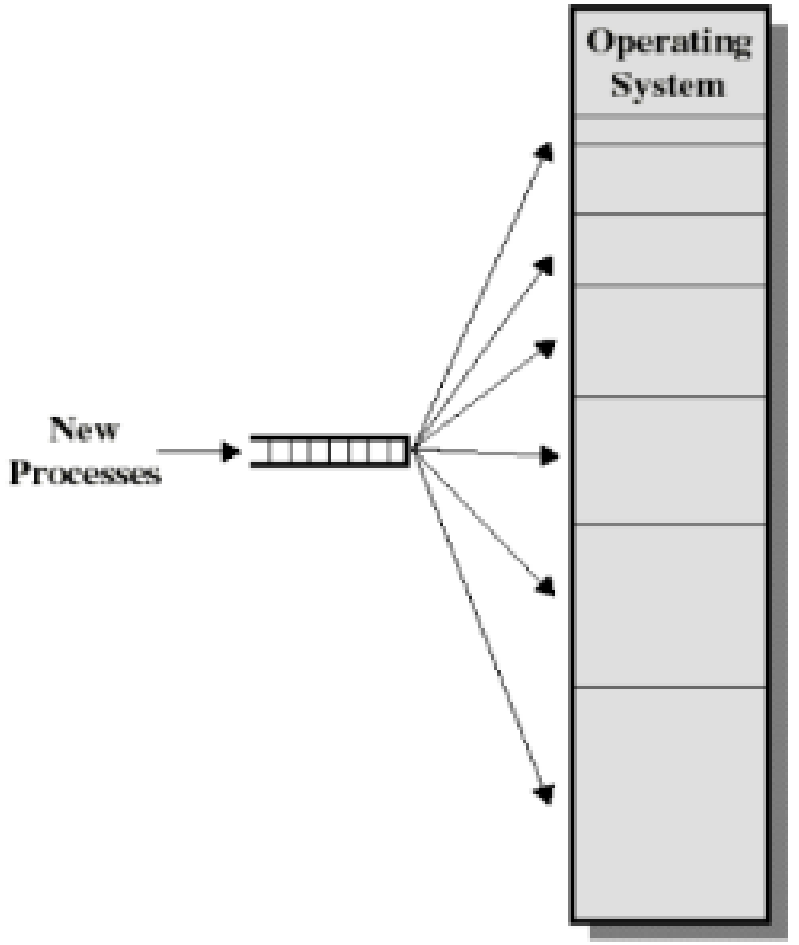
Chiến lược placement

- Với partition **khác** kích thước:
- **Giải pháp 2**: Sử dụng 1 hàng đợi

➢ Chỉ có một hàng đợi chung cho tất cả partition

➢ Khi cần nạp một process vào bộ nhớ chính

⇒ chọn partition nhỏ nhất còn trống



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia động (dynamic partitioning)

- Số lượng partition và kích thước không cố định, có thể khác nhau
- Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết

➔ *Nhận xét: Gây hiện tượng phân mảnh ngoại*

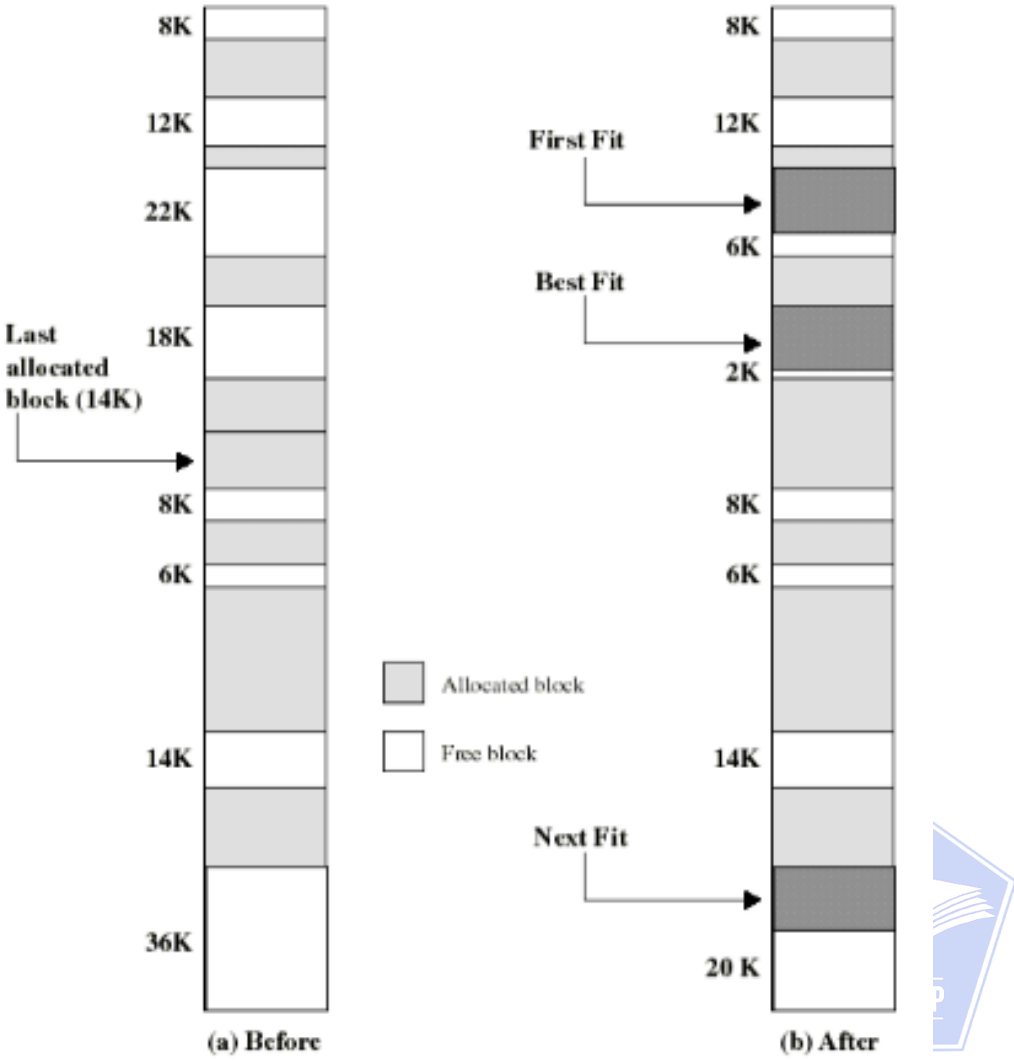
Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia động (dynamic partitioning)

Chiến lược placement (giúp giảm chi phí compaction):

- **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



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

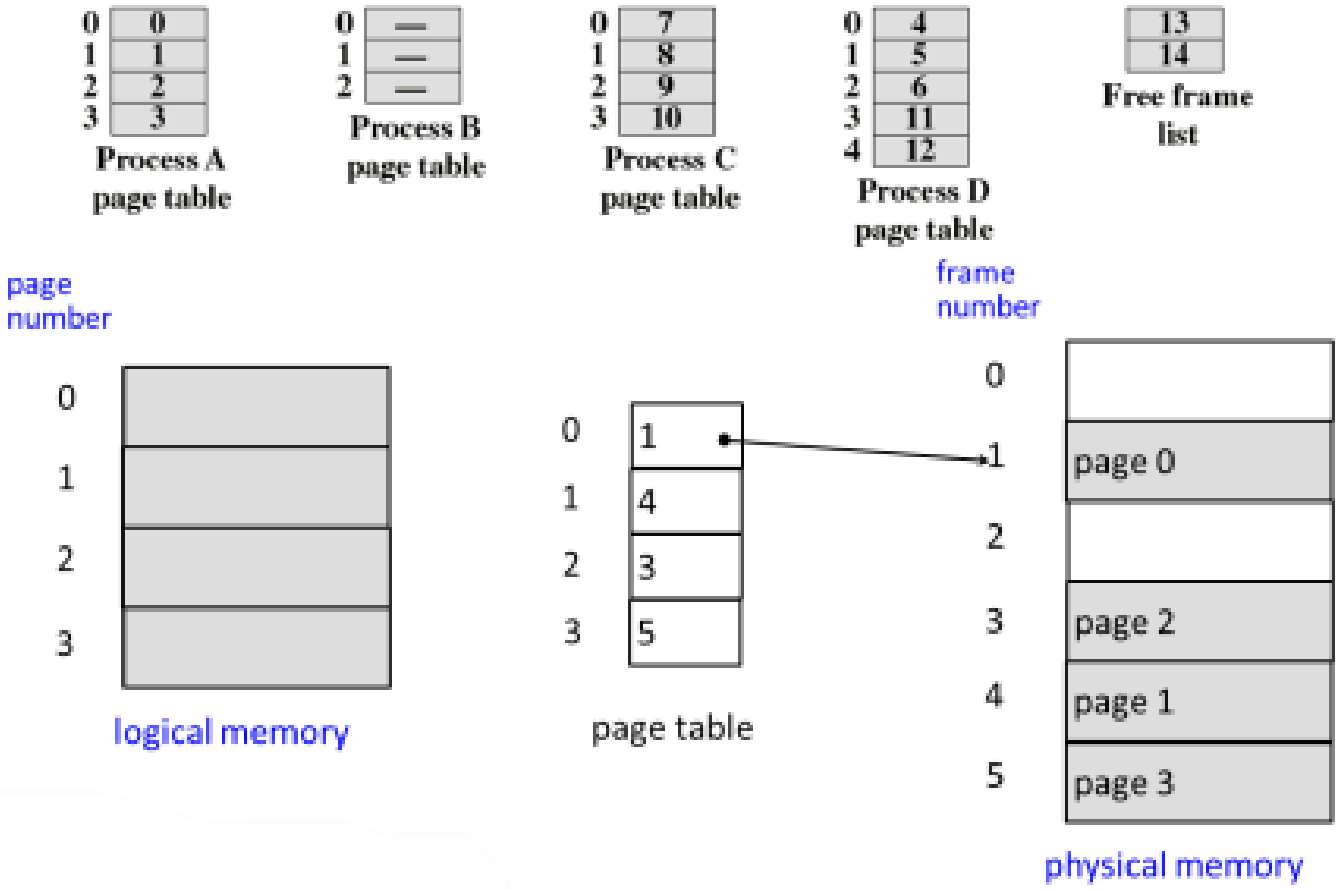
Phân chia trang (Paging)

- Chia nhỏ bộ nhớ vật lý thành các khối nhỏ có kích thước bằng nhau (frames). Kích thước của frame là luỹ thừa của 2 (từ 512 bytes đến 16MB)
- Chia bộ nhớ luận lý của tiến trình thành các khối nhỏ có kích thước bằng nhau (pages). Mỗi page có kích thước = 1 frame.
- Để chạy chtr kích thước n pages, cần tìm n frames còn trống nạp vào chtr
- Địa chỉ luận lý gồm có:
  - Số hiệu trang(page number) (p)
  - Địa chỉ tương đối trong trang/độ dời (page offset) (d)
- Bảng phân trang(page table) dùng để ánh xạ địa chỉ luận lý thành địa chỉ thực
- Nội dung mỗi phần tử trong page table cho biết chỉ số frame của bộ nhớ thực

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Phân chia trang (Paging)

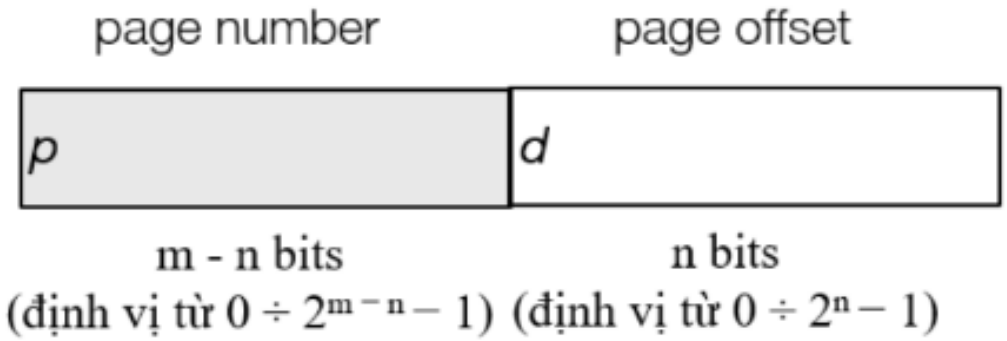


Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Chuyển đổi địa chỉ trong paging

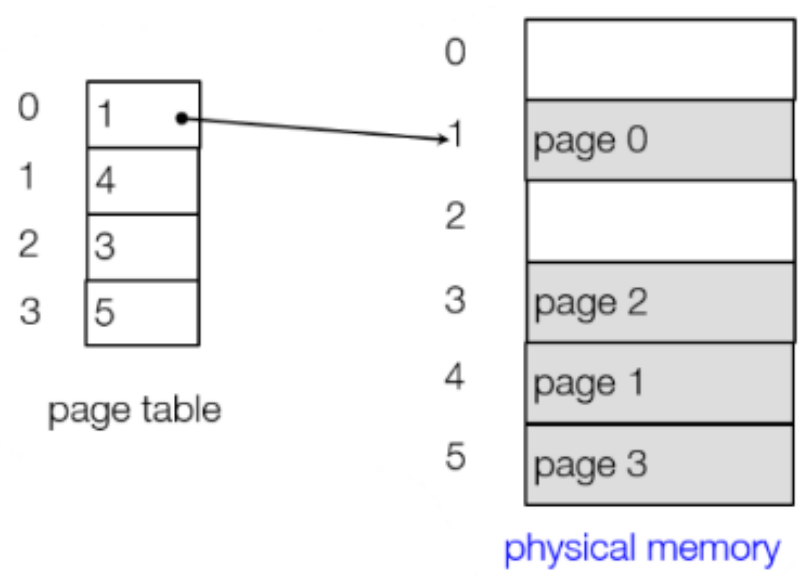
- Nếu kích thước của không gian địa chỉ ảo là 2^m, và kích thước của trang là 2^n (đơn vị là byte hay word tùy theo kiến trúc máy) thì
- Bảng trang sẽ có tổng cộng $2^m / 2^n = 2^{m-n}$ mục (entry)
- Mỗi page sẽ ứng với một frame và được tìm kiếm thông qua page table



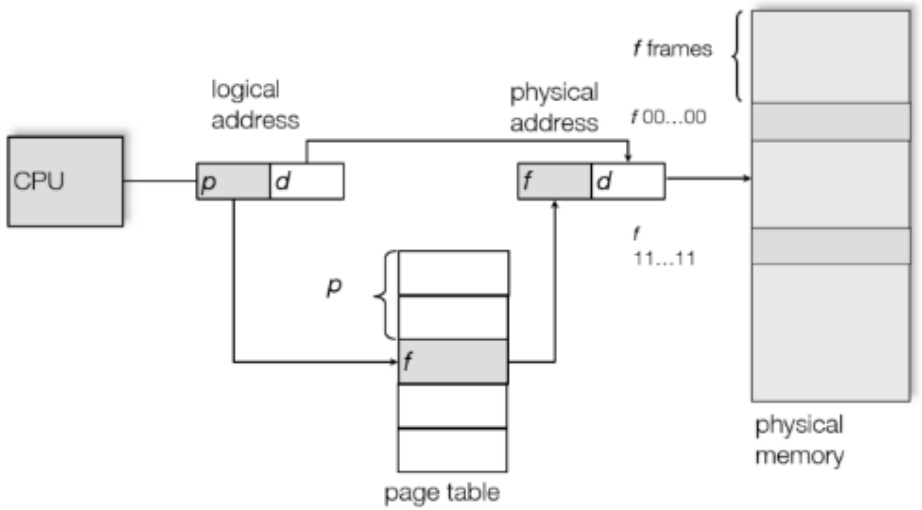
Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Chuyển đổi địa chỉ trong paging



Page table

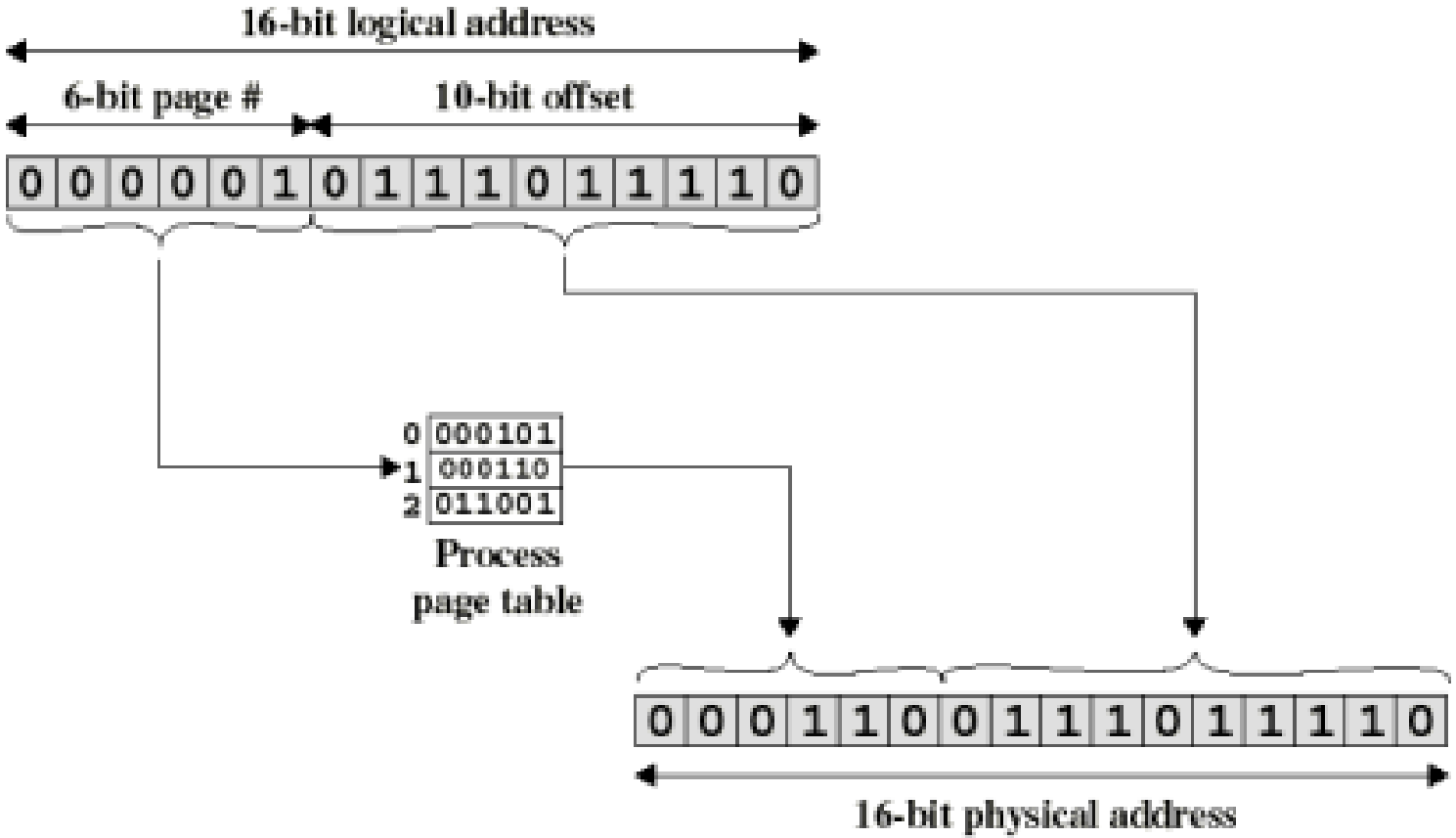


Chuyển đổi địa chỉ thông qua page table

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Chuyển đổi địa chỉ trong paging



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Hoạt động của Page Table

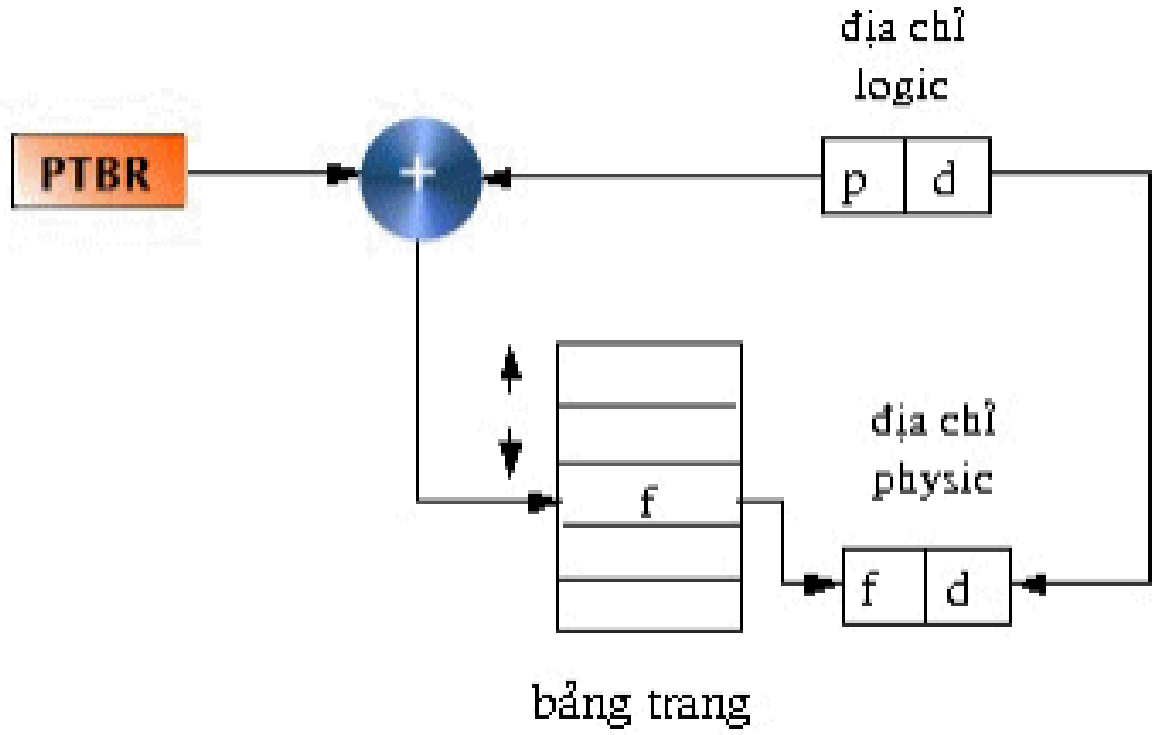
- Page table được lưu trong main memory
  - Mỗi process được OS cấp một bảng phân trang
  - Page – table base register (PTBR) chỉ tới page table
  - Page – table length register (PBLR) cho biết kích thước của page
- Theo cơ chế cài đặt này thì một thao tác truy cập lệnh hoặc dữ liệu cần đến **2 lần truy cập bộ nhớ chính**.
  - Lần 1 cho bảng trang.
  - Lần 2 cho lệnh hoặc dữ liệu.
- Thường dùng một bộ phận cache phần cứng có tốc độ truy xuất và tìm kiếm cao, gọi là thanh ghi kết hợp (associative register) hoặc translation look-aside buffers (TLBs).

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Hoạt động của Page Table

Dùng thanh ghi PTBR



Chương 7. QUẢN LÍ BỘ NHỚ

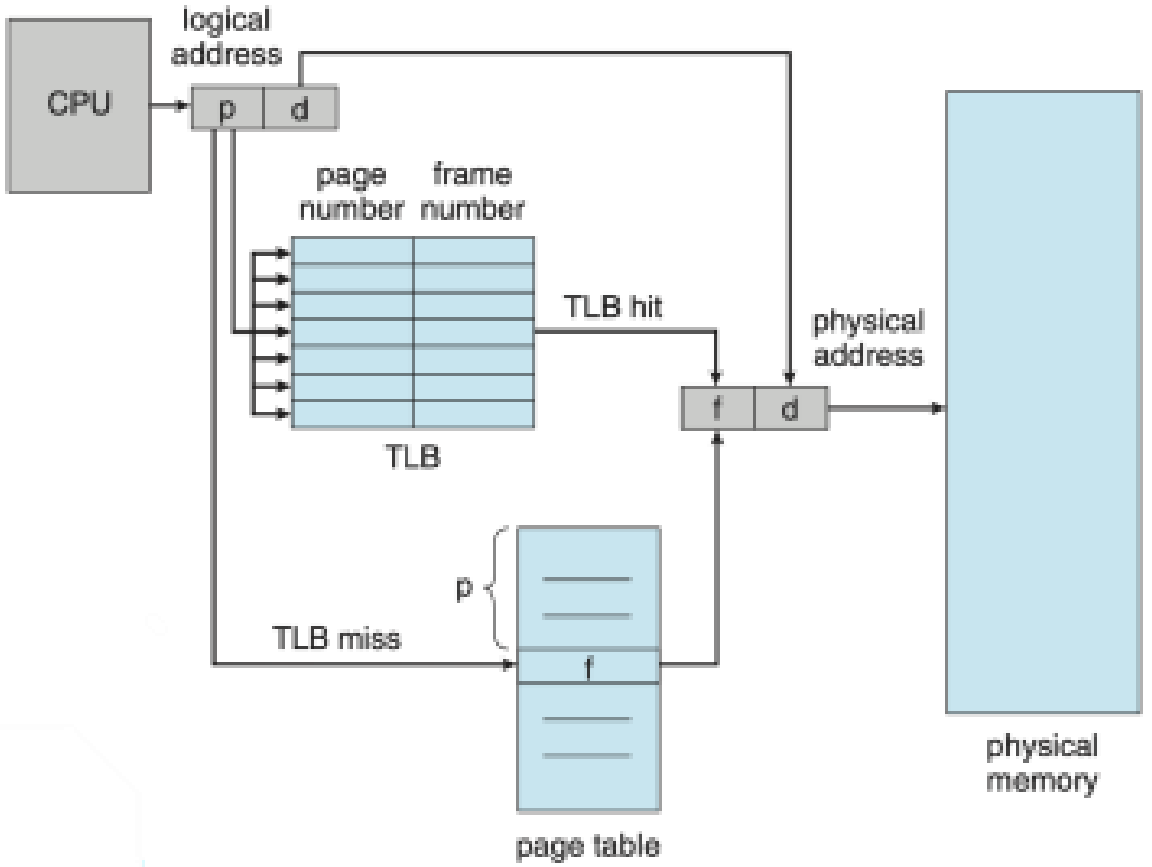
3. Các mô hình quản lý bộ nhớ:

Hoạt động của Page Table

Dùng TLB

TLB trong CACHE

PAGE trong RAM



Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Translation look-aside buffers (TLBs)

Thời gian truy xuất hiệu dụng (effective access time,EAT)

- Thời gian tìm kiếm trong TLB: **ε**
- Thời gian truy xuất trong bộ nhớ: **x**
- Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
- Kí hiệu hit ratio: **α**

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Translation look-aside buffers (TLBs)

Thời gian cần thiết để có được chỉ số frame

- Khi chỉ số trang có trong TLB (hit): **ε + x**
- Khi chỉ số trang không có trong TLB (miss): **ε+x+x**

Thời gian truy xuất hiệu dụng

- **EAT = (ε + x)α + (ε + 2x)(1 – α)**

  = (2– α)x + ε

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Translation look-aside buffers (TLBs)

Ví dụ 1: đơn vị thời gian nano giây

- Associative lookup = 20
- Memory access = 100
- Hit ratio = 0.8
- EAT = (100 + 20) × 0.8 + (200 + 20) × 0.2
  = 1.2 × 100 + 20 = 140 54

Chương 7. QUẢN LÍ BỘ NHỚ

3. Các mô hình quản lý bộ nhớ:

Translation look-aside buffers (TLBs)

Ví dụ 2: đơn vị thời gian nano giây

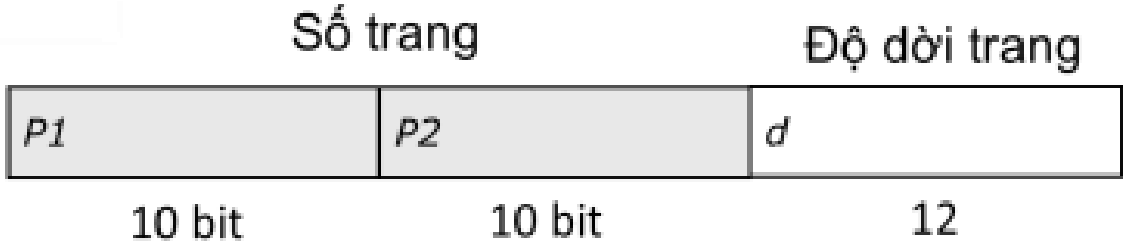
- Associative lookup = 20
- Memory access = 100
- Hit ratio = 0.98
- EAT = (100 + 20) × 0.98 + (200 + 20) × 0.02
  = 1.02 × 100 + 20 = 122

Chương 7. QUẢN LÍ BỘ NHỚ

4. Tổ chức bảng trang

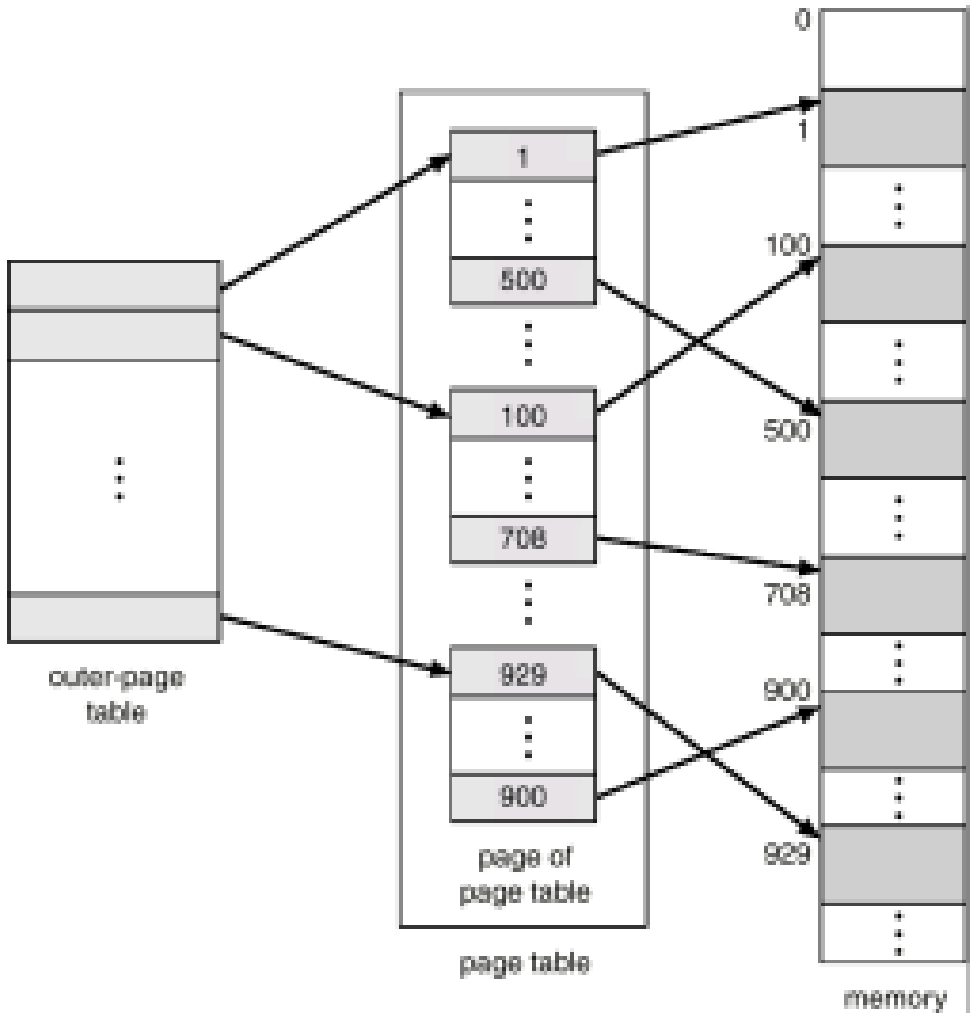
- Các hệ thống hiện đại đều hỗ trợ không gian địa chỉ ảo rất lớn (2^{32} đến 2^{64}), ở đây giả sử là 2^{32} .
  - Giả sử kích thước trang nhớ là 4KB (= 2^{12})

    ⇒ bảng phân trang sẽ có 2^{32} /2^{12} = 2^{20} = 1M mục.
  - Giả sử mỗi mục gồm 4 byte thì mỗi process cần 4MB cho bảng phân trang
  - Ví dụ: Phân trang 2 cấp



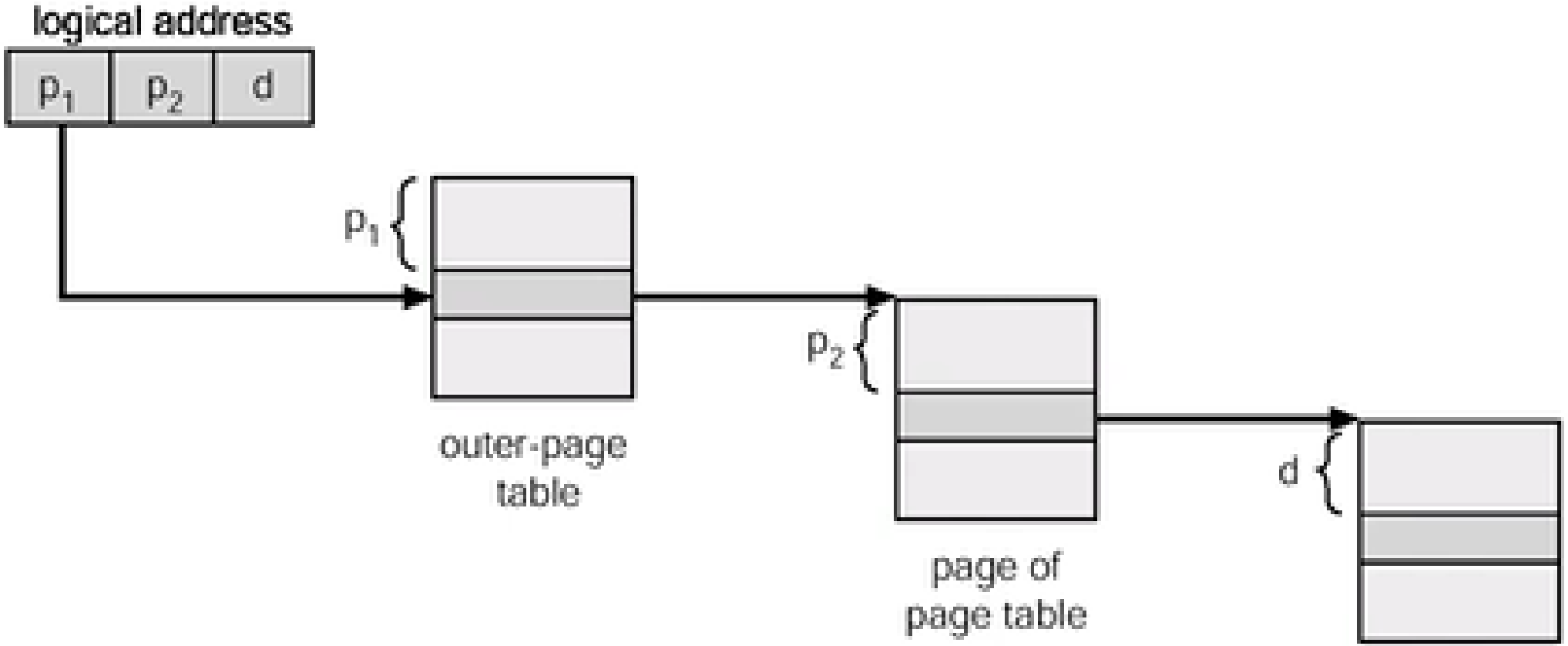
Chương 7. QUẢN LÍ BỘ NHỚ

4. Tổ chức bảng trang



Chương 7. QUẢN LÍ BỘ NHỚ

4. Tổ chức bảng trang



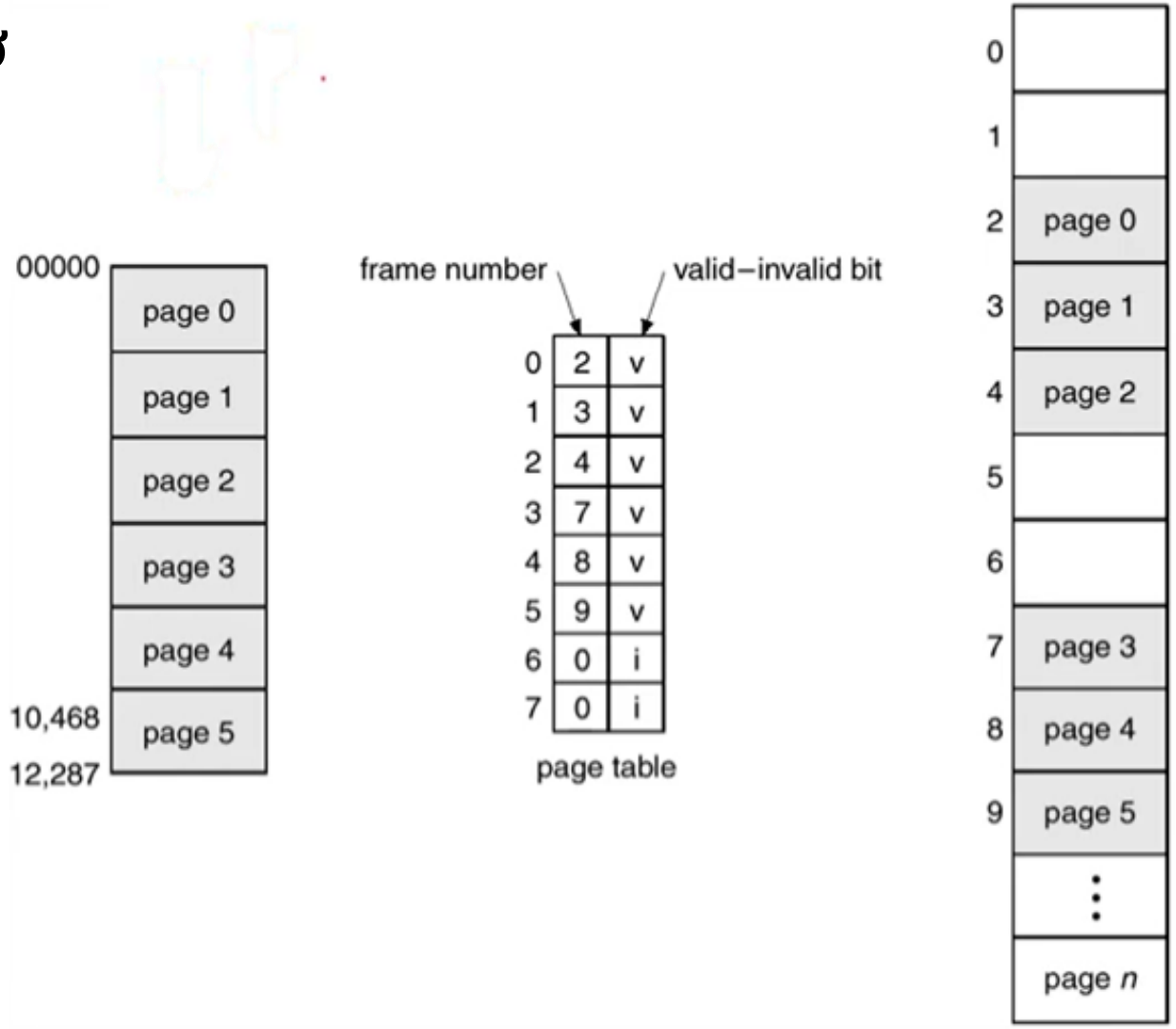
Chương 7. QUẢN LÍ BỘ NHỚ

5. Bảo vệ bộ nhớ

- Việc bảo vệ bộ nhớ được hiện thực bằng cách gắn với frame các bit bảo vệ (protection bits) được giữ trong bảng phân trang. Các bit này biểu thị các thuộc tính sau:
  - read-only, read-write, execute-only
- Ngoài ra, còn có một valid/invalid bit gắn với mỗi mục trong bảng phân trang:
  - "valid": cho biết là trang của tiến trình, do đó là một trang hợp lệ.
  - "invalid": cho biết là trang không của tiến trình, do đó là một trang bất hợp lệ

Chương 7. QUẢN LÍ BỘ NHỚ

5. Bảo vệ bộ nhớ



Chương 7. QUẢN LÍ BỘ NHỚ

Câu 1: Hiện tượng bộ nhớ có những vùng trống rời rạc, không liên tục, không chứa tiến trình nào được gọi là gì?

A. Bộ nhớ phân tán.

B. Bộ nhớ không liên tục.

C. Phân mảnh nội.

D. Phân mảnh ngoại

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 2: Chọn phát biểu đúng.

A. Hiện tượng phân mảnh ngoại thường xảy ra khi bộ nhớ được cấp phát theo chia thành các khối kích thước cố định và các process được cấp phát theo đơn vị khối.

B. Có thể dùng cơ chế kết khối (compaction) gom lại thành vùng nhớ liên tục để giải quyết hiện tượng phân mảnh ngoại

C. Hiện tượng phân mảnh ngoại là khi kích thước vùng nhớ được cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu.

D. Hiện tượng phân mảnh nội là kích thước không gian nhớ trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên vùng nhớ này không liên tục.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 3: Giả sử bộ nhớ chính được phân chia thành các vùng cố định theo thứ tự sau: 1(260 KB), 2(170 KB), 3(150 KB), 4(180 KB), 5(130 KB), 6 (190 KB). Biết vùng nhớ 2,5 đã được cấp phát, các vùng nhớ khác vẫn còn trống. Hỏi tiến trình P có kích thước 160 KB sẽ được cấp phát vào vùng nhớ nào, nếu dùng giải thuật Best-fit?

A. 1

B. 2

C. 4

D. 6

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

a. Xét trường hợp bộ nhớ được phân vùng cố định:

- First fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.
  - Tiến trình 426K sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

a. Xét trường hợp bộ nhớ được phân vùng cố định:

- Next fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.
  - Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

a. Xét trường hợp bộ nhớ được phân vùng cố định:

- Best fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 300K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.
  - Tiến trình 426K sẽ được cấp phát vùng nhớ 600K..

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

a. Xét trường hợp bộ nhớ được phân vùng cố định:

- Worst fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 300K.
  - Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

b. Xét trường hợp bộ nhớ được phân vùng động:

- First fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K => Vùng nhớ trống 388K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 388K.
  - Tiến trình 426K sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

b. Xét trường hợp bộ nhớ được phân vùng động:

- Best fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 300K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.
  - Tiến trình 426K sẽ được cấp phát vùng nhớ 600K..

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

b. Xét trường hợp bộ nhớ được phân vùng động:

- Next fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 200K.
  - Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 4: Giả sử bộ nhớ chính được cấp phát thành các phân vùng có kích thước là 600K, 500K, 200K, 300K (theo thứ tự), sau khi thực thi xong, các tiến trình có kích thước 212K, 417K, 112K, 426K (theo thứ tự) sẽ được cấp phát bộ nhớ như thế nào, nếu sư dụng: Thuật toán First fit, Best fit, Next fit, Worst fit? Thuật toán nào cho phép sử dụng bộ nhớ hiệu quả nhất trong trường hợp trên?

Giải:

b. Xét trường hợp bộ nhớ được phân vùng động:

- Worst fit:
  - Tiến trình 212K sẽ được cấp phát vùng nhớ 600K. => Vùng nhớ trống 388K.
  - Tiến trình 417K sẽ được cấp phát vùng nhớ 500K.
  - Tiến trình 112K sẽ được cấp phát vùng nhớ 388K.
  - Tiến trình 426K sẽ sẽ phải chờ vì không còn vùng nhớ trống thỏa yêu cầu.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 5: Xét một không gian địa chỉ có 12 trang, mỗi trang có kích thước 1024 từ, mỗi từ 2 bytes, ánh xạ vào bộ nhớ vật lý có 32 khung trang.

a. Địa chỉ logic gồm bao nhiêu bit?

b. Địa chỉ physic gồm bao nhiêu bit?

Giải:

a. Địa chỉ logic gồm 2 phần: chỉ số trang và độ dời (offset) trong trang. Cần 4 bit để biểu diễn chỉ số trang và 11 bit (2^{11} = 2K = 2048) để biểu diễn độ dời trong trang. Vậy địa chỉ logic gồm 15 bit.

b. 32 frame <= 2^5 ➔ cần 5 bit để biểu diễn chỉ số trang. ➔ Địa chỉ physic gồm 16 bit.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 6: Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a. Nếu thời gian cho một lần truy xuất bộ nhớ bình thường (memory reference) là 200ns thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ (paged memory reference) trong hệ thống này?

b. Nếu sử dụng TLBs với hit-ratio là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ (effective access time) trong hệ thống?

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 6: Xét một hệ thống sử dụng kỹ thuật phân trang, với bảng trang được lưu trữ trong bộ nhớ chính.

a. Nếu thời gian cho một lần truy xuất bộ nhớ bình thường (memory reference) là 200ns thì mất bao nhiêu thời gian cho một thao tác truy xuất bộ nhớ (paged memory reference) trong hệ thống này?

b. Nếu sử dụng TLBs với hit-ratio là 75%, thời gian để tìm trong TLBs xem như bằng 0, tính thời gian truy xuất bộ nhớ (effective access time) trong hệ thống?

Giải:

a. Mỗi thao tác truy xuất bộ nhớ trong hệ thống này sẽ cần thực hiện 2 lần truy xuất bộ nhớ thông thường: truy xuất bảng trang (để xác định vị trí khung trang – tìm bảng trang trong bộ nhớ chính) và truy xuất vị trí bộ nhớ (xác định dựa trên sự kết hợp giá trị khung trang tìm được ở lần trước với độ dời trong trang – tìm dữ liệu trong khung trang). Do đó thời gian của một thao tác truy xuất bộ nhớ sẽ là 200*2 = 400ns.

b. Thời gian truy xuất bộ nhớ (effective access time) trong hệ thống:
EAT = (200 + 0) x 0.75 + (200 + 200 + 0) x 0.25

= (2 – α)x + ε = 250 ns.

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 7: Một máy tính 32-bit địa chỉ, sử dụng một bảng trang 2 cấp. Địa chỉ ảo được phân bổ như sau: 9 bit dành cho bảng trang cấp 1, 11 bit cho bảng trang cấp 2 và còn lại cho offset. Cho biết kích thước một trang trong hệ thống và địa chỉ ảo có bao nhiêu trang?

Giải:

Số trang địa chỉ ảo: $2^9 * 2^{11} = 2^{20}$

Kích thước một trang : 2^{12}

P1	P2	d
9	11	12

Chương 7. QUẢN LÍ BỘ NHỚ

Câu 8: Cho bảng trang như hình bên.

a. Địa chỉ vật lý 6568 sẽ được chuyển thành địa chỉ ảo bao nhiêu? Biết rằng kích thước mỗi frame là 1K bytes.

b. Địa chỉ ảo 3254 sẽ được chuyển thành địa chỉ vật lý bao nhiêu? Biết rằng kích thước mỗi frame là 2K bytes.

0	**6**
1	**4**
2	**5**
3	**7**
4	**1**
5	**9**
6	**8**

Giải:

a. Địa chỉ vật lý 6568 nằm ở khung trang 6 với độ dời 424. Trang 0 được nạp vào khung trang 6 ➔ Địa chỉ ảo là 424.

b. Địa chỉ 3254 nằm ở trang 1 với độ dời 1206. Trang 1 được nạp vào khung trang 4 ➔ Địa chỉ vật lý là 9398.

Chương 8. BỘ NHỚ ẢO

Chương 8. BỘ NHỚ ẢO

1. Tổng quan
2. Cài đặt bộ nhớ ảo
3. Phân trang theo yêu cầu
4. Giải thuật thay trang
5. Vấn đề cấp phát Frames
6. Vấn đề Thrasing

Chương 8. BỘ NHỚ ẢO

1. Tổng quan

- **Bộ nhớ ảo (virtual memory):** là một kỹ thuật cho phép xử lý một tiến trình không được nạp toàn bộ vào bộ nhớ vật lý
- **Ưu điểm**:

➢ 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

➢ Giảm nhẹ công việc của lập trình viên

Chương 8. BỘ NHỚ ẢO

2. Cài đặt bộ nhớ ảo

- **Có hai kỹ thuật**
  - ➢ Phân trang theo yêu cầu (demand paging)
  - ➢ Phân đoạn theo yêu cầu (demand segmentation)
- Phần cứng memory management phải hỗ trợ paging và/hoặc segmentation
- OS phải quản lý sự di chuyển của trang/đoạn giữa bộ nhớ chính và bộ nhớ thứ cấp

Chương 8. BỘ NHỚ ẢO

3. Phân trang theo yêu cầu

- Demange Paging: Các trang của tiến trình chỉ được nạp vào bộ nhớ chính khi được yêu cầu
- **Page-fault:** khi tiến trình tham chiếu đến một trang không có trong bộ nhớ chính (valid bit) thì phần cứng sẽ gây ra một lỗi trang (page-fault)
- Khi có page-fault thì phần cứng sẽ gây ra một ngắt (page-fault-trap) kích khởi page-fault service routine (PFSR)

Chương 8. BỘ NHỚ ẢO

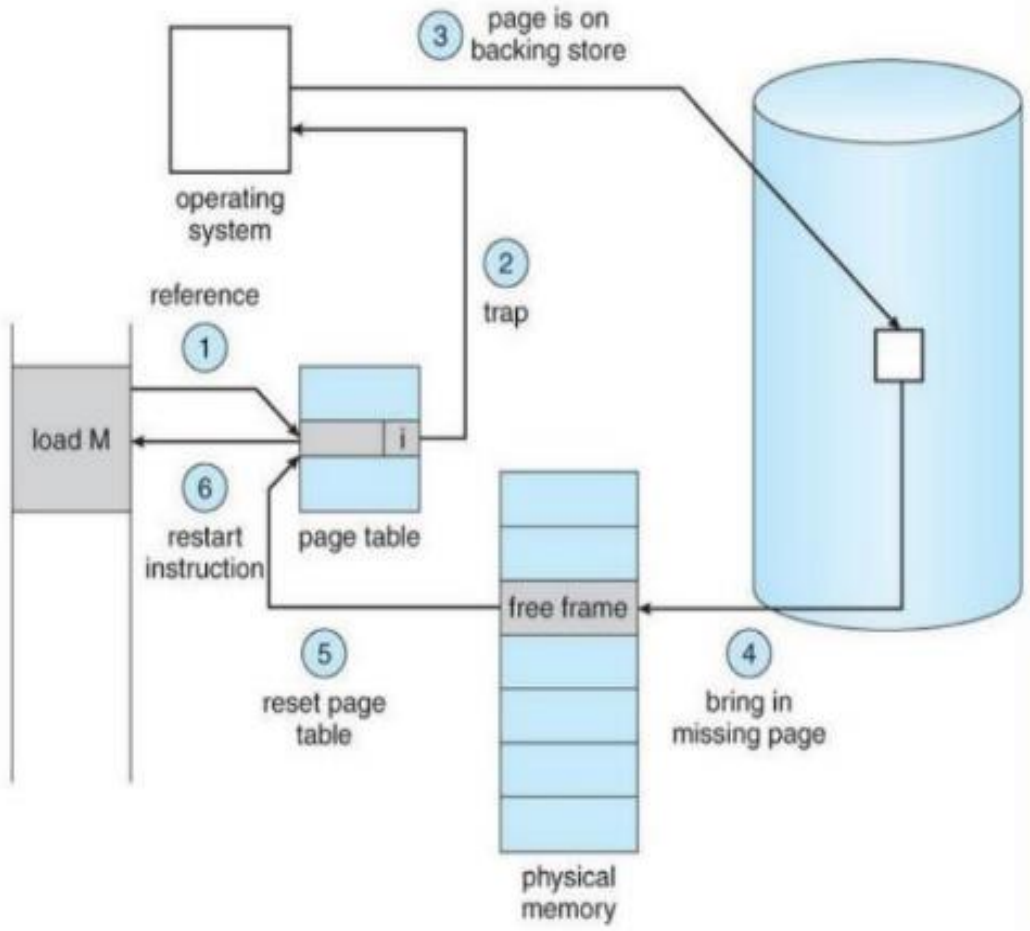
3. Phân trang theo yêu cầu

PFSR

- Chuyển process về trạng thái Blocked
- Phát ra một yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống; trong khi đợi I/O, một process khác được cấp CPU để thực thi
- Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành; PFSR cập nhật page table và chuyển process về trạng thái ready.

Nếu không tìm được frame trống:

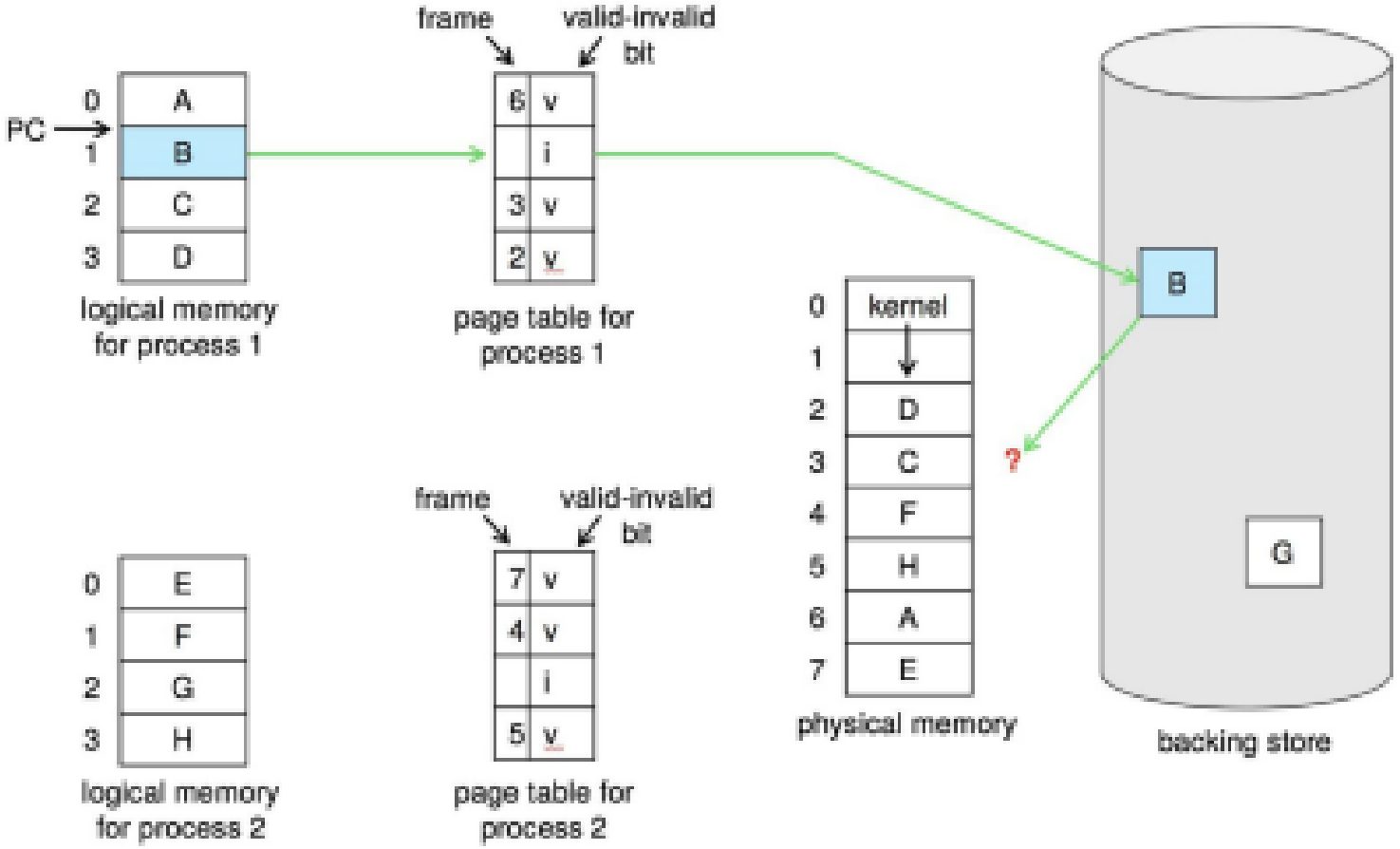
- Dùng giải thuật thay trang chọn một trang hi sinh (victim page)
- Ghi victim page lên đĩa



Chương 8. BỘ NHỚ ẢO

3. Phân trang theo yêu cầu

Khi cần thay thế trang



Chương 8. BỘ NHỚ ẢO

3. Phân trang theo yêu cầu

Thay thế trang nhớ

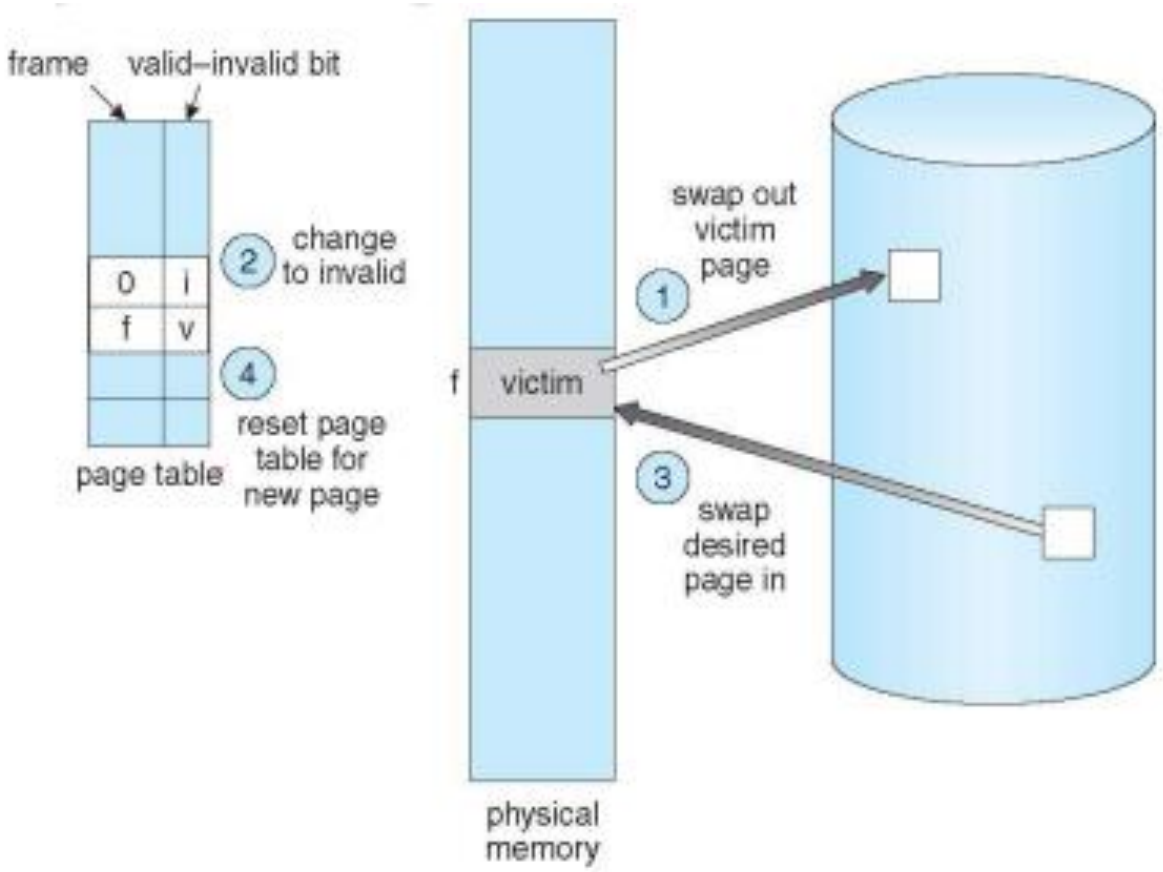
Bước 2 của PFSR giả sử phải thay trang vì không tìm được frame trống, PFSR được bổ sung như sau:

- Xác định vị trí trên đĩa của trang đang cần
- Tìm một frame trống:
  - Nếu có frame trống thì dùng nó
  - Nếu không có frame trống thì dùng một giải thuật thay trang để chọn một trang hy sinh (victim page)
  - Ghi victim page lên đĩa; cập nhật page table và frame table tương ứng
- Đọc trang đang cần vào frame trống (đã có được từ bước 2); cập nhật page table và frame table tương ứng.

Chương 8. BỘ NHỚ ẢO

3. Phân trang theo yêu cầu

Thay thế trang nhớ



Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

- Giải thuật thay trang FIFO
- Giải thuật thay trang OPT
- Giải thuật thay trang LRU

Dữ liệu cần biết

- Số khung trang
- Tình trạng ban đầu (thường là trống)
- Chuỗi tham chiếu

Xét ví dụ: Chuỗi truy xuất bộ nhớ:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

Xét ví dụ: Chuỗi truy xuất bộ nhớ: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

FIFO (frist in, frist out): vào trước ra trước

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
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

➔15 page-fault

Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

Giải thuật thay trang OPT thay thế trang nhớ sẽ được **tham chiếu trễ nhất trong tương lai** ⇒ cần phải biết trước các trang sẽ được tham chiếu trong tương lai

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
*	*	*	*		*		*			*			*				*		

➔ 9 lỗi trang

Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

- Mỗi trang được ghi nhận (trong bảng phân trang) thời điểm được tham chiếu ⇒ trang LRU là trang nhớ có thời điểm tham chiếu nhỏ nhất (OS tốn chi phí tìm kiếm trang nhớ LRU này mỗi khi có page fault).
- Do vậy, LRU cần sự hỗ trợ của phần cứng và chi phí cho việc tìm kiếm. Ít CPU cung cấp đủ sự hỗ trợ phần cứng cho giải thuật LRU.

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

➔ 12 lỗi trang

Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

Nghịch lý Belady

Sử dụng 3 khung trang

1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	4	4	4	5	5	5	5	5	5
	2	2	2	1	1	1	1	1	3	3	3
		3	3	3	2	2	2	2	2	4	4
*	*	*	*	*	*	*			*	*	

Sử dụng 4 khung trang

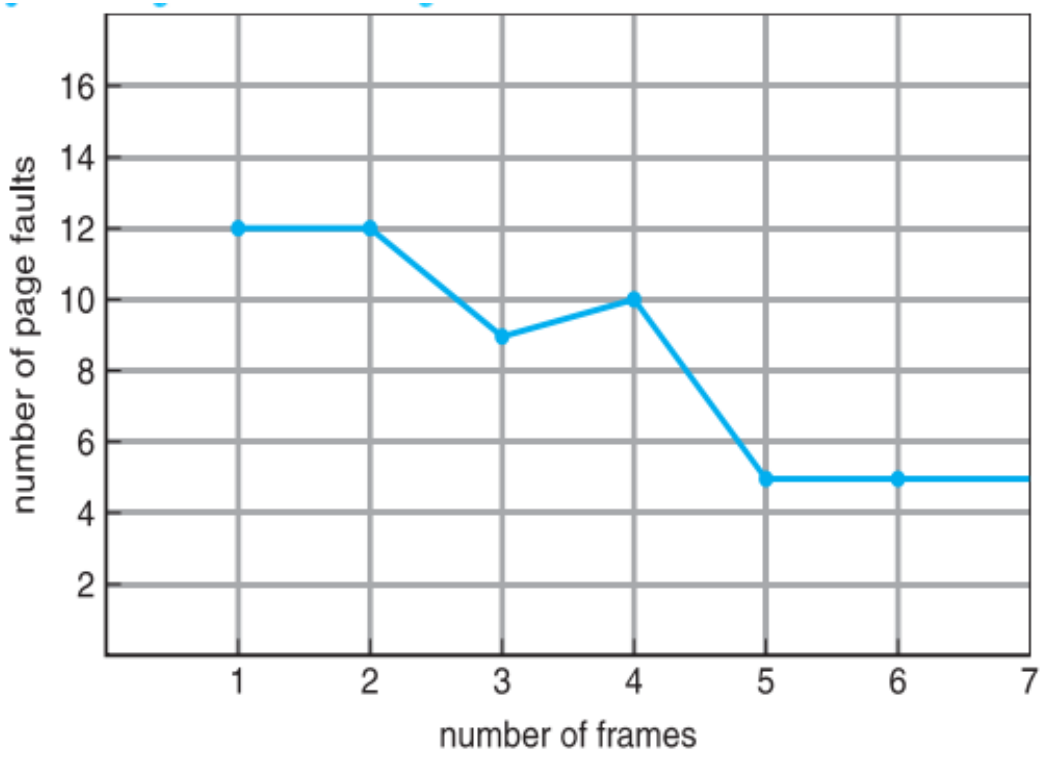
1	2	3	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	5	5	5	5	4	4
	2	2	2	2	2	2	1	1	1	1	5
		3	3	3	3	3	3	2	2	2	2
			4	4	4	4	4	4	3	3	3
*	*	*	*			*	*	*	*	*	*

Chương 8. BỘ NHỚ ẢO

4. Giải thuật thay trang

Nghịch lý Belady

- Số page-fault tăng mặc dù quá trình đã được cấp nhiều frame hơn



Chương 8. BỘ NHỚ ẢO

5. Vấn đề cấp phát Frames

- **OS phải quyết định cấp cho mỗi tiến trình bao nhiêu frame.**
  - Cấp ít frame ⇒ nhiều page fault
  - Cấp nhiều frame ⇒ giảm mức độ multiprogramming
- **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
  - *OS phải mất chi phí để ước định các process*

Chương 8. BỘ NHỚ ẢO

5. Vấn đề cấp phát Frames

Chiến lượt cấp phát tĩnh

- Cấp phát bằng nhau:
  - Ví dụ, có 100 frame và 5 tiến trình → mỗi tiến trình được 20 frame
- Cấp phát theo tỉ lệ: dựa vào kích thước tiến trình

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

Ví dụ:

$m = 64$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

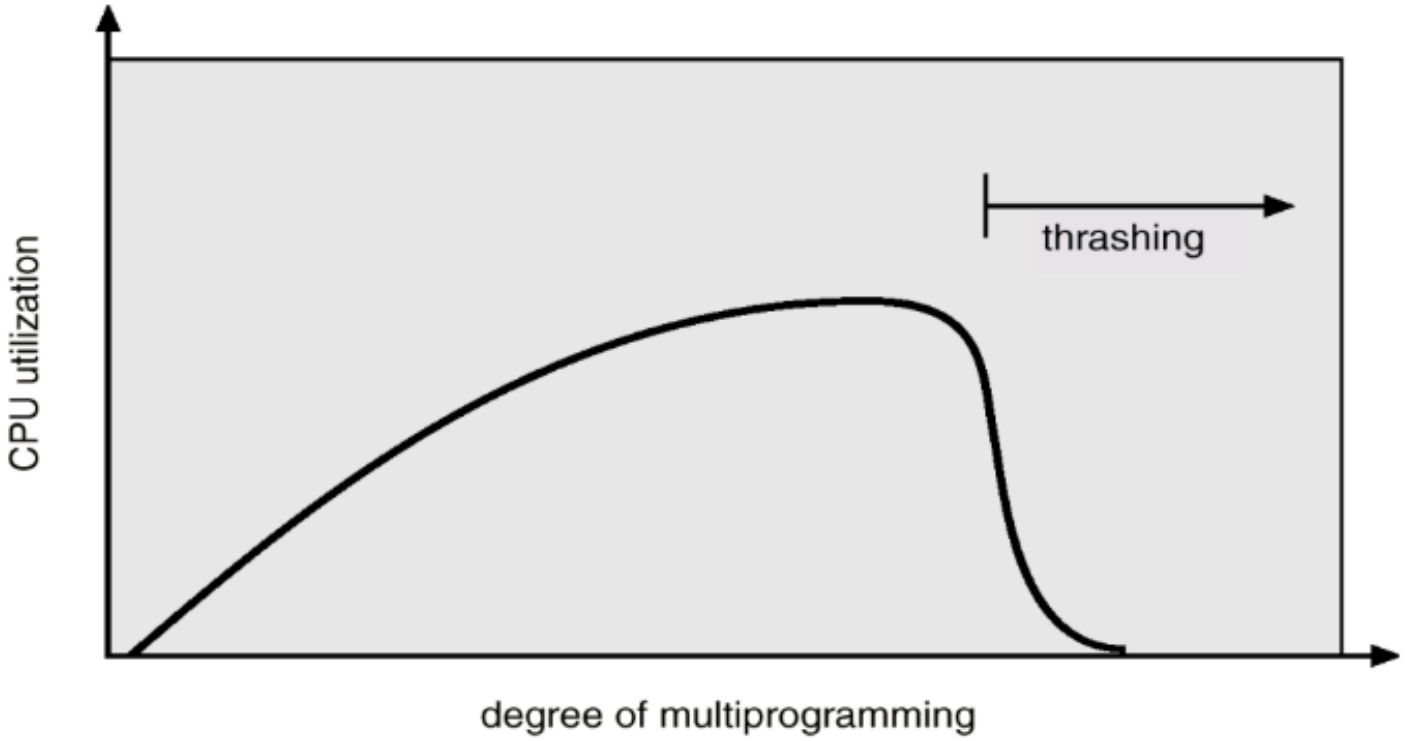
$a_2 = \frac{127}{137} \times 64 \approx 59$

- Cấp phát theo độ ưu tiên

Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

- Nếu một process không có đủ số frame cần thiết thì tỉ số page faults/sec rất cao
- Thrashing: hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục



Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

Mô hình cục bộ

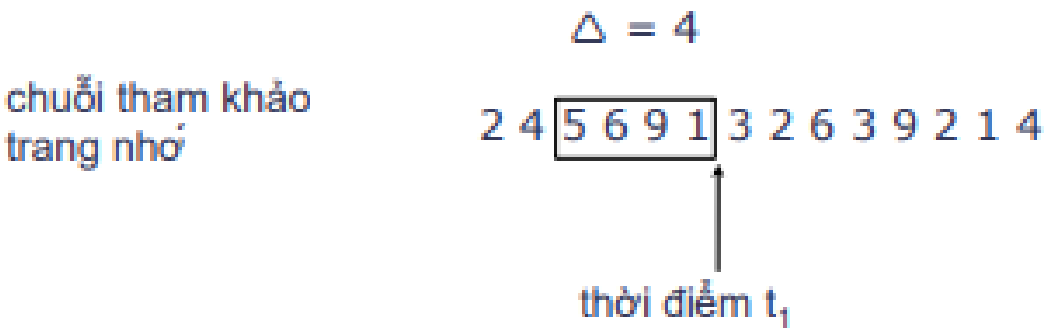
- Để hạn chế thrashing, hệ điều hành phải cung cấp cho tiến trình càng "đủ" frame càng tốt. Bao nhiêu frame thì đủ cho một tiến trình thực thi hiệu quả?
- Nguyên lý locality (locality principle)
  - Locality là tập các trang được tham chiếu gần nhau.
  - Một tiến trình gồm nhiều locality, và trong quá trình thực thi, tiến trình sẽ chuyển từ locality này sang locality khác.
- Vì sao hiện tượng thrashing xuất hiện? Khi Σ size of locality > memory size

Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

Giải pháp tập làm việc

- Được thiết kế dựa trên nguyên lý locality.
- Xác định xem tiến trình thực sự sử dụng bao nhiêu frame. • Định nghĩa:
  - WS(t) - các tham chiếu trang nhớ của tiến trình gần đây nhất cần được quan sát.
  - Δ - khoảng thời gian tham chiếu
- Ví dụ:

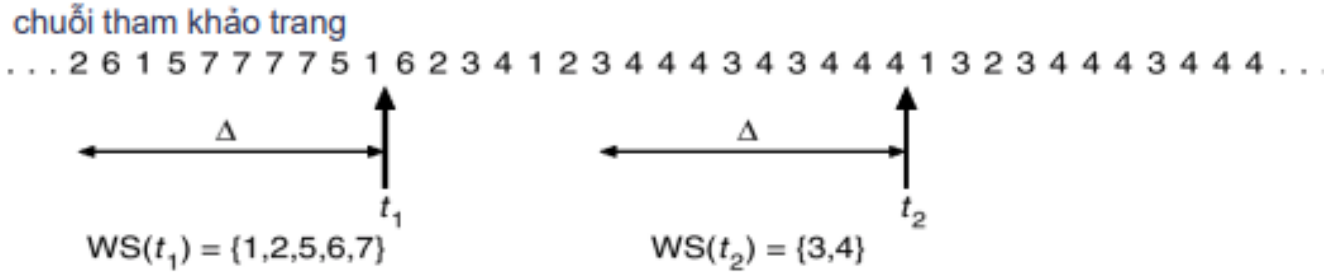


Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

Giải pháp tập làm việc

- Định nghĩa: Working set của tiến trình Pi , ký hiệu WSi , là tập gồm Δ các trang được sử dụng gần đây nhất.
  - Ví dụ Δ = 10 và chuỗi tham khảo trang



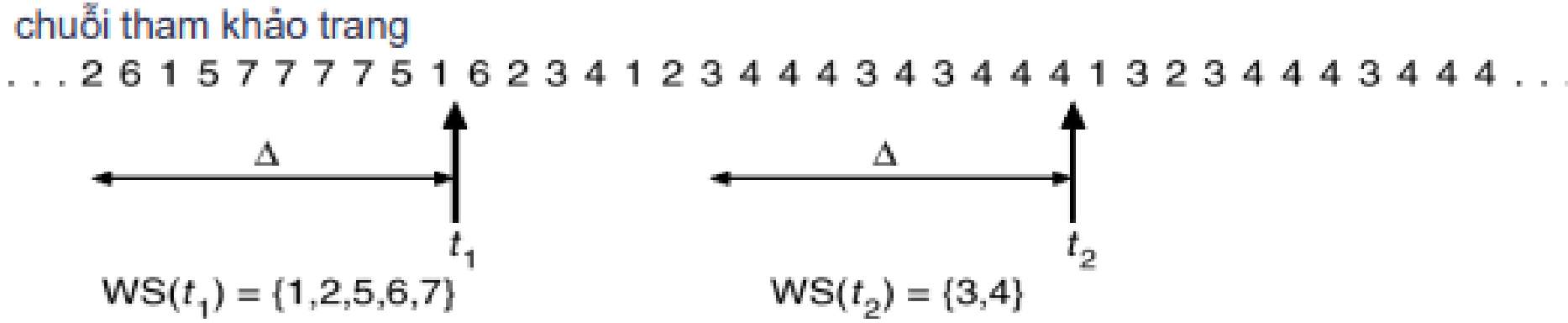
- Nhận xét:
  - ➢ Δ quá nhỏ ⇒ không đủ bao phủ toàn bộ locality.
  - ➢ Δ quá lớn ⇒ bao phủ nhiều locality khác nhau.
  - ➢ Δ = ∞ ⇒ bao gồm tất cả các trang được sử dụng.
- Dùng working set của một tiến trình để xấp xỉ locality của nó

Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

Giải pháp tập làm việc

- Định nghĩa: WSSi là kích thước của working set của Pi :
  - WSSi = số lượng các trang trong WSi 45
  - Ví dụ: Δ = 10 và chuỗi tham khảo trang



WSS(t1) = 5

WSS(t2) = 2

Chương 8. BỘ NHỚ ẢO

6. Vấn đề Thrasing

Giải pháp tập làm việc

- Đặt D = Σ WSSi = tổng các working-set size của mọi tiến trình trong hệ thống.
  - Nhận xét: Nếu D > m (số frame của hệ thống) ⇒ sẽ xảy ra thrashing.
- Giải pháp working set:
  - Khi khởi tạo một tiến trình: cung cấp cho quá trình số lượng frame thỏa mãn working-set size của nó.
  - Nếu D > m ⇒ tạm dừng một trong các tiến trình.
    - Các trang của tiến trình được chuyển ra đĩa cứng và các frame của nó được thu hồi
- WS loại trừ được tình trạng trì trệ mà vẫn đảm bảo mức độ đa chương

Chương 8. BỘ NHỚ ẢO

Câu 1: Có bao nhiêu phát biểu đúng về ưu điểm của bộ nhớ ảo?

(1) Giảm nhẹ công việc của lập trình viên.

(2) Tất cả tiến trình được thực thi nhanh hơn.

(3) Số lượng process trong bộ nhớ ít hơn.

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

(5) Tốc độ truy xuất bộ nhớ nhanh hơn.

A. 2 B. 3 C. 4 D. 5

Chương 8. BỘ NHỚ ẢO

Câu 2: Trong kỹ thuật cài đặt bộ nhớ ảo sử dụng phân trang theo yêu cầu, khi dùng chiến lược cấp phát động, nếu tỷ lệ page-fault thấp thì:

A. Giảm bớt frame

B. Cấp thêm frame.

C. Không thay đổi.

D. Tăng thêm gấp 2 lần.

Chương 8. BỘ NHỚ ẢO

Câu 3: Phát biểu nào sau đây **không** đúng?

A. Thrashing là hiện tượng các trang nhớ của một process bị hoán chuyển vào/ra liên tục.

B. Trong fixed-allocation khi cài đặt bộ nhớ ảo sử dụng phân trang theo yêu cầu thì số frame cấp cho mỗi process không đổi.

C. Trong variable-allocation khi cài đặt bộ nhớ ảo sử dụng phân trang the có thể thay đổi trong khi process chạy.

D. Khi hệ điều hành cấp ít frame có thể giảm page fault

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

1, 4, 2, 4 , 6, 7, 12, 9, 0, 4, 3, 5, 7, 8, 11, 19, 23, 5, 6, 7, 9, 8, 14, 10

A, F, G, W, B, C, A, N, M, Z, X, O, P, T, U, R, I, V, D, F, G, J, H, L, K, E, C

Vẽ bảng minh họa thuật toán và tính số lỗi trang (page fault) khi tiến trình truy xuất chuỗi bộ nhớ trên theo các giải thuật sau, giả sử có 4 frame:

a. FCFS

b. LRU

c. OPT

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

1, 4, 2, 4 , 6, 7, 12, 9, 0, 4, 3, 5, 7, 8, 11, 19, 23, 5, 6, 7, 9, 8, 14, 10

1	4	2	4	6	7	12	9	0	4	3	5	7	8	11	19	23	5	6	7	9	8	14	10

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

1, 4, 2, 4 , 6, 7, 12, 9, 0, 4, 3, 5, 7, 8, 11, 19, 23, 5, 6, 7, 9, 8, 14, 10

1	4	2	4	6	7	12	9	0	4	3	5	7	8	11	19	23	5	6	7	9	8	14	10

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

1, 4, 2, 4 , 6, 7, 12, 9, 0, 4, 3, 5, 7, 8, 11, 19, 23, 5, 6, 7, 9, 8, 14, 10

1	4	2	4	6	7	12	9	0	4	3	5	7	8	11	19	23	5	6	7	9	8	14	10

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

1, 4, 2, 4 , 6, 7, 12, 9, 0, 4, 3, 5, 7, 8, 11, 19, 23, 5, 6, 7, 9, 8, 14, 10

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

A, F, G, W, B, C, A, N, M, Z, X, O, P, T, U, R, I, V, D, F, G, J, H, L, K, E, C

Chương 8. BỘ NHỚ ẢO

Câu 4: Cho dãy sau:

A, F, G, W, B, C, A, N, M, Z, X, O, P, T, U, R, I, V, D, F, G, J, H, L, K, E, C

CHƯƠNG 9. HỆ ĐIỀU HÀNH LINUX VÀ WINDOWS

CHƯƠNG 9. HỆ ĐIỀU HÀNH LINUX VÀ WINDOWS

1. Nguyên tắc thiết kế của hệ điều hành Linux/Windows?
2. Các thành phần của hệ điều hành Linux/Windows?
3. Linux/Windows quản lý tiến trình như thế nào? Có điểm gì khác so với kiến thức đã học?
4. Các tiến trình trên Linux/Windows giao tiếp với nhau như thế nào? Có điểm gì khác so với kiến thức đã học?
5. Bộ nhớ ảo trên Linux/Windows được tổ chức và hoạt động như thế nào? Có điểm gì khác so với kiến thức đã học?

Hướng dẫn chữa đề thi thử

Câu 13: Xét một hệ thống sử dụng kỹ thuật phân trang với bảng trang được lưu trữ trong bộ nhớ chính. Nếu sử dụng TLSs với hit ratio = 0.7 thì thời gian truy xuất bộ nhớ trong hệ thống (effective access time) EAT = 300ns. Biết thời gian chu kỳ truy xuất bộ nhớ x = 200ns, hỏi thời gian để tìm trong TLBs là bao nhiêu?

A. 210ns.
B. 40ns.
C. 260ns.
D. 130ns.

- **EAT = (2−∝)x + ε**

EAT = 300ns, x = 200ns, hit ratio = 0.7

Thời gian truy xuất hiệu dụng (effective access time,EAT)

- Thời gian tìm kiếm trong TLB: **ε**
- Thời gian truy xuất trong bộ nhớ: **x**
- Hit ratio: tỉ số giữa số lần chỉ số trang được tìm thấy (hit) trong TLB và số lần truy xuất khởi nguồn từ CPU
- Kí hiệu hit ratio: **α**

Sử dụng các dữ liệu sau để trả lời câu hỏi 16, 17, 18:
Xét một không gian địa chỉ ảo có 64 trang, mỗi trang có kích thước 2048 bytes được ánh xạ vào bộ nhớ vật lý có 32 khung trang.

Câu 16: Địa chỉ luận lý gồm bao nhiêu bit?
A. 17.
B. 6.
C. 32.
D. 11.

Câu 17: Địa chỉ vật lý gồm bao nhiêu bit?
A. 5.
B. 16.
C. 7.
D. 11.

Câu 18: Bảng phân trang có tất cả bao nhiêu mục?
A. 32.
B. 11.
C. 2048.
D. 64.

- 64 = 2^6
- 2048 = 2^11
- 32 = 2^5
- Địa chỉ luận lý: page + offset = 6 + 11 = 17
- Địa chỉ vật lý: frame + offset = 5 + 11 = 16
- Bảng phân trang = số trang = 64

Câu 19: Xét một hệ thống có bộ nhớ được cấp phát theo cơ chế phân trang với kích thước trang và khung trang là 1024 byte. Biết các trang 1, 2, 3, 4 của bộ nhớ luận lý lần lượt được nạp vào khung trang 5, 2, 4, 1 của bộ nhớ vật lý. Hỏi địa chỉ vật lý 5524 nằm trong trang nào của bộ nhớ luận lý? (G1)

A. 1
B. 2
C. 3
D. 4

- Trang vật lý chứa địa chỉ 5524: 5524/1024 ~ 5
- Trang vật lý 5 tương ứng với trang luận lý 1

Câu 21: Xét một hệ thống có bộ nhớ được cấp phát theo cơ chế phân trang với kích thước trang và khung trang là 1024 byte. Biết trang 0 và trang 1 của bộ nhớ ảo lần lượt được nạp vào khung trang 4, 2 của bộ nhớ vật lý. Hỏi địa chỉ ảo 684 được ánh xạ thành địa chỉ vật lý bao nhiêu?

A. 684
B. 2732
C. 4780
D. 1708

- Địa chỉ vật lý = Khung trang * Trang + offset
- Trang của địa chỉ ảo 684 là: 684/1024 ~ 0
- Khung trang tương ứng với trang 0 là khung trang 4
- Địa chỉ vật lý = 4 * 1024 + 684 = 4780

Sử dụng các dữ liệu sau để trả lời câu hỏi 23, 24, 25:
Giả sử một tiến trình được cấp 4 khung trang trong bộ nhớ vật lý và 8 trang trong bộ nhớ ảo. Tại thời điểm nạp tiến trình vào, 4 khung trang trên bộ nhớ vật lý này đang trống. Tiến trình truy xuất 8 trang (1,2, 3, 4, 5, 6, 7, 8) trong bộ nhớ ảo theo thứ tự như sau:

6 2 4 7 5 8 3 4 1 2 3 7 5 6 1 2 8 3 5 1

Câu 23. Tại thời điểm tiến trình truy xuất trang nhớ số 8 lần đầu tiên, trang nhớ nào sẽ bị thay thế, nếu sử dụng giải thuật thay thế trang OPT?
A. 6
B. 2
C. 7
D. 5

Câu 24. Tại thời điểm tiến trình truy xuất trang nhớ số 1 lần đầu tiên, trang nhớ nào sẽ bị thay thế, nếu sử dụng giải thuật thay thế trang tối ưu LRU?
A. 4
B. 3
C. 5
D. 8

Câu 25. Tại thời điểm tiến trình truy xuất trang nhớ số 1 lần đầu tiên, có tất cả bao nhiêu lỗi trang đã xảy ra (không tính lỗi trang xảy ra khi nạp trang nhớ số 1 vào), nếu sử dụng giải thuật thay thế trang FIFO?
A. 6
B. 7
C. 8
D. 9

FIFO

6	2	4	7	5	8	3	4	1	2	3	7	5	6	1	2	8	5	3	1
6	6	6	6	5	5	5	5	1	1	1	1	1	6	6	6	6	5	5	1
	2	2	2	2	8	8	8	8	2	2	2	2	2	1	1	1	1	3	3
		4	4	4	4	3	3	3	3	3	7	7	7	7	2	2	2	2	2
			7	7	7	7	4	4	4	4	4	5	5	5	5	8	8	8	8
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x

LRU

6	2	4	7	5	8	3	4	1	2	3	7	5	6	1	2	8	5	3	1
6	6	6	6	5	5	5	5	1	1	1	1	5	5	5	5	8	8	8	8
	2	2	2	2	8	8	8	8	2	2	2	2	6	6	6	6	5	5	5
		4	4	4	4	3	3	3	3	3	3	3	3	1	1	1	1	3	3
			7	7	7	7	4	4	4	4	7	7	7	7	2	2	2	2	1
x	x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x

OPT

6	2	4	7	5	8	3	4	1	2	3	7	5	6	1	2	8	5	3	1
6	6	6	6	5	8	3	3	3	3	3	3	3	6	6	6	8	8	3	3
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		4	4	4	4	4	4	1	1	1	1	1	1	1	1	1	1	1	1
			7	7	7	7	7	7	7	7	7	5	5	5	5	5	5	5	5
x	x	x	x	x	x	x		x				x	x			x		x	

QR Điểm danh

BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ I NĂM HỌC 2023 – 2024

Sharing is learning

HẾT

CẢM ƠN CÁC BẠN ĐÃ THEO DÕI
CHÚC CÁC BẠN CÓ KẾT QUẢ THI THẬT TỐT!

BAN HỌC TẬP

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

CONTACT

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit