# BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

## TRAINING CUỐI KỲ HỌC KỲ I NĂM HỌC 2022 – 2023

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 2 ngày 06/02/2023

**Địa điểm:** Microsoft Teams

**Trainers:** Nguyễn Hà Mi – KTMP2021

Phạm Long Vũ – MTCL2021

# Chương 5: Đồng bộ

# Chương 5: Đồng bộ

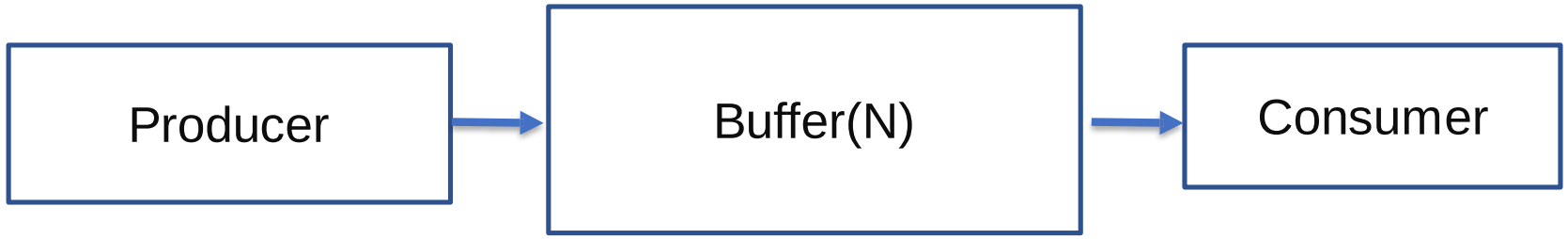
**Vấn đề cần đồng bộ:**

- 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

**Ví dụ: Producer - Consumer**

- 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

## Ví dụ: Producer - Consumer

- Quá trình Producer

```
item nextProduce;
 while(1){
        while(count == BUFFER_SIZE); /*ko lam gi*/
        buffer[in] = nextProducer;
        count++;
        in = (in+1)%BUFFER_SIZE; }
```

- Quá trình Consumer

```
item nextConsumer;
 while(1){
        while(count == 0); /*ko lam gi*/
        nextConsumer = buffer[out];
        count--;
        out = (out+1)%BUFFER_SIZE; }
```

# Chương 5: Đồng bộ

**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

**Yêu cầu của lời giải cho CS Problem:**

- Lời giải phải thỏa ba tính chất:

(1) 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.

(2) Progress: Một tiến trình tạm dừng bên ngoài miền giăng hông được ngăn cản các tiến trình khác vào miền giăng

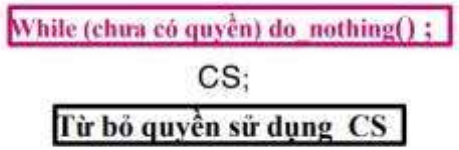
(3) Chờ đợi giới hạn (Bounded waiting): Mỗi process chỉ phải chờ để được vào vùng tranh chấp trong một khoảng thời gian có hạn định nào đó. Không xảy ra tình trạng đói tài nguyên (starvation).

# Chương 5: Đồng bộ

## Phân loại giải pháp:

### Busy-waiting

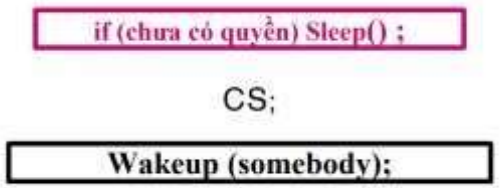
- Tiếp tục tiêu thụ CPU trong khi chờ đợi vào miền giăng
- Không đòi hỏi sự trợ giúp của Hệ điều hành




- Sử dụng các biến cờ hiệu
- Sử dụng việc kiểm tra luân phiên
- Giải pháp của Peterson
- Cấm ngắt
- Chỉ thị TSL

### Sleep & Wake up

- Từ bỏ CPU khi chưa được vào miền giăng
- Cần Hệ điều hành hỗ trợ




- Semaphore
- Monitor
- Message

# Chương 5: Đồng bộ

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

A. Peterson

B. Bakery

C. Semaphore

D. Cấm ngắt

# Chương 5: Đồng bộ

**Câu 16: 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 11: 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

## Busy – waiting: Giải thuật kiểm tra luân phiên

```
Process P0:
do
  while (turn != 0);
        critical section
  turn := 1;
        remainder section
while (1);
```

```
Process P1:
do
  while (turn != 1);
        critical section
  turn := 0;
        remainder section
while (1);
```

**Câu hỏi: Điều gì xảy ra nếu P0 có RS rất lớn còn RS của P1 rất nhỏ?**

A. P1 luôn vào vùng tranh chấp trước P0
B. P0 có thể cản P1 vào vùng tranh chấp mặc dù P0 đã nằm ngoài vùng tranh chấp.
C. Cả 2 tiến trình có thể vào vùng tranh chấp cùng 1 lúc.
D. P1 có thể cản P0 vào vùng tranh chấp mặc dù P1 đã nằm ngoài vùng tranh chấp.

=> Giải thuật trên **thỏa Mutal Exclusion** nhưng **không thỏa Progress và Bounded–Waiting**

# Chương 5: Đồng bộ

## Busy – waiting: Giải thuật sử dụng biến cờ hiệu:

- Biến chia sẻ :
  boolean flag[ 2 ]; /* khởi đầu flag[ 0 ] = flag[ 1 ] = false
- Nếu flag[ i ] = true thì Pi "sẵn sàng" vào critical section

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

**Câu hỏi: Giải thuật trên không thỏa mãn tính chất nào trong việc giải quyết CS Problem?**

A. Không thỏa mãn cả ba tính chất
B. Mutual exclusion
C. Progress
D. B và C đều đúng

# Chương 5: Đồng bộ

## Busy – waiting: Giải thuật Peterson: tiến trình P0, P1

Process $P_0$

```
do {
   /*  0 wants in  */
   flag[0] = true;
   /*  0 gives a chance to 1  */
   turn = 1;
   while (flag[1] &&turn == 1);
      critical section
   /*  0 no longer wants in  */
   flag[0] = false;
      remainder section
} while(1);
```

Process $P_1$

```
do {
   /*  1 wants in  */
   flag[1] = true;
   /*  1 gives a chance to 0  */
   turn = 0;
   while (flag[0] && turn == 0);
      critical section
   /*  1 no longer wants in  */
   flag[1] = false;
      remainder section
} while(1);
```

=> Thỏa mãn 3 điều kiện

# Chương 5: Đồng bộ

## Busy – waiting: Giải thuật Bakery: n tiến trình (Process)

- Trước khi vào CS, process Pi nhận một con số. Process nào giữ con số nhỏ nhất thì được vào CS
- Trường hợp Pi và Pj cùng nhận được một chỉ số:
  Nếu i < j thì Pi được vào trước. (Đối xứng)
- Khi ra khỏi CS, Pi đặt lại số của mình bằng 0
- Cơ chế cấp số cho các process thường tạo các số theo cơ chế tăng dần, ví dụ 1, 2, 3, 3, 3, 3, 4, 5,…
- Kí hiệu
  (a,b) < (c,d) nếu a < c hoặc nếu a = c và b < d
  max(a0,…,ak) là con số b sao cho b ≥ ai với mọi i = 0,…, k

```
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);
```

# Chương 5: Đồng bộ

## Busy – waiting: Giải thuật cấm ngắt:

- Trong hệ thống uniprocessor: mutual exclusion 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

- Trong hệ thống multiprocessor: mutual exclusion không được đảm bảo

  -Chỉ cấm ngắt tại CPU thực thi lệnh disable_interru

  -Các CPU khác vẫn có thể truy cập bộ nhớ chia sẻ

```
Process Pi:

do {
        disable_interrupts();
            critical section
        enable_interrupts();
            remainder section
} while (1);
```


Sharing is learning

# Chương 5: Đồng bộ

## Busy – waiting: Giải thuật dùng 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)
```

- 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

## Busy – waiting: Giải thuật dùng lệnh Test and set:

```
boolean TestAndSet( boolean
*target){
   boolean rv = *target;
   *target = true;
   return rv;
}
```

- Shared data:
  boolean lock = false;
- Process $P_i$ :

```
do {
    while (TestAndSet(&lock));
        critical section
    lock = false;
        remainder section
} while (1);
```

- 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 (bị bỏ đói)

# Chương 5: Đồng bộ

**Câu 17 : 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 mền

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 5: Đồng bộ

## Sleep & wake up : Semaphore:

- Là công cụ đồng bộ cung cấp bởi OS mà không đòi hỏi busy waiting
- Semaphore S là một biến số nguyên.
- Ngoài thao tác khởi động biến thì chỉ có thể được truy xuất qua hai tác vụ cố tính đơn nguyên (atomic) và loại trừ (mutual exclusive)
  + wait(S) hay còn gọi là P(S): giảm giá trị semaphore(S=S-1).
    Kế đó nếu giá trị này thì process thực hiện lệnh wait() bị blocked.
  + signal(S) hay còn gọi là V(S): tăng giá trị semaphore (S=S+1).
    Kế đó nếu giá trị này không dương, một process đang blocked bởi một lệnh wait()\ sẽ được hồi phục để thực thi.
- Tránh busy waiting: khi phải đợi thì process sẽ được đặt vào một blocked queue, trong đó chứa các process đang chờ đợi cùng một sự kiện.

# Chương 5: Đồng bộ

## Sleep & wake up : Hiện thực Semaphore:

```
void wait(semaphore S) {
        S.value--;
        if (S.value < 0) {
                add this process to S.L;
                block();
        }
}
void signal(semaphore S) {
        S.value++;
        if (S.value <= 0) {
                remove a process P from S.L;
                wakeup(P);
        }
}
```

# Chương 5: Đồng bộ

## Sleep & wake up : Ví dụ về Semaphore:

Cho 3 tiến trình P(P1,P2), Q(Q1,Q2), R(R1,R2). Hãy đồng bộ hóa hoạt động của 3 tiến trình sao cho:

a. P2 hoàn tất trước R2.

b. Q1 hoàn thành trước R1 và P2.

**Giải :**

| P | Q | R |
|---|---|---|
| P1<br>Wait(s2);<br>P2<br>signal(s1); | Q1<br>signal(s2);<br>Q2 | wait(s2);<br>R1<br>wait(s1);<br>R2 |

# Chương 5: Đồng bộ

**Các loại Semaphore:**

- Counting semaphore: một số nguyên có giá trị không hạn chế.
- Binary semaphore: có trị là 0 hay 1. Binary semaphore rất dễ hiện thực.
- Có thể hiện thực counting semaphore bằng binary semaphore

# Chương 5: Đồng bộ

**Các vấn đề đối với Semaphore:**

- Semaphore cung cấp một công cụ mạnh mẽ để bảo đảm mutual exclusion và phối hợp đồng bộ các process
- Tuy nhiên, nếu các tác vụ wait(S) và signal(S) nằm rải rác ở rất nhiều processes ⇒ khó nắm bắt được hiệu ứng của các tác vụ này. Nếu không sử dụng đúng ⇒ có thể xảy ra tình trạng deadlock hoặc starvation.
- Một process bị "die" có thể kéo theo các process khác cùng sử dụng biến semaphore.

# Chương 5: Đồng bộ

**Sleep & Wake up: Monitor**

- Cũng là một cấu trúc ngôn ngữ cấp cao tương tự CR, có chức năng như semaphore nhưng dễ điều khiển hơn
- Xuất hiện trong nhiều ngôn ngữ lập trình đồng thời như Concurrent Pascal, Modula-3, Java,…
- Có thể hiện thực bằng semaphore

# Chương 5: Đồng bộ

## Câu hỏi 25: Cho các tính chất sau:

(1)Khi một tiến trình P đang thực thi trong vùng tranh chấp của nó thì không có tiến trình Q nào khác đang thực thi trong vùng tranh chấp của Q
(2) Một tiến trình tạm dừng bên ngoài vùng tranh chấp không ngăn cản các tiến trình khác vào vùng tranh chấp
(3) Các tiến trình phải từ bỏ CPU khi chưa vào vùng tranh chấp
(4)Mỗi tiến trình phải chờ đợi để vào vùng tranh chấp trong một khoảng thời gian có hạn định nào đó. Không xảy ra tình trạng đói tài nguyên
(5)Khi một tiến trinh P đang thực thi trong vùng tranh chấp của nó thì không có tiến trình Q nào khác đang thực thi trong vùng tranh chấp của P

Lời giải dành cho vấn đề vùng tranh chấp cần phải thỏa mãn tính chất nào ?

A.(2), (4), (5)
B.(1), (3), (4)
C.(1), (2), (4)
D.(1), (2), (4), (5)


Sharing is learning

# Chương 5: Đồng bộ

Xét giải pháp đồng bộ sử dụng 3 semaphore full, empty, mutex để giải quyết bài toán bounded buffer như bên dưới. Biết giá trị khởi tạo của các semaphore trên lần lượt là 0, n và 1 với n là kích thước của buffer. Vai trò của semaphore mutex trong giải pháp này là gì?

**producer**

```
do {
    ...
    nextp = new_item();
    ...
    wait(empty);
    wait(mutex);
    ...
    insert_to_buffer(nextp);
    ...
    signal(mutex);
    signal(full);
} while (1);
```

**consumer**

```
do {
    wait(full)
    wait(mutex);
    ...
    nextc = get_buffer_item(out);
    ...
    signal(mutex);
    signal(empty);
    ...
    consume_item(nextc);
    ...
} while (1);
```

A. Đảm bảo producer và consumer không được thao tác trên buffer cùng lúc.
B. Đảm bảo producer không được ghi dữ liệu vào buffer đã đầy.
C. Đảm bảo consumer không được đọc dữ liệu từ buffer đang trống.
D. Đảm bảo không có deadlock hoặc starvation xảy ra.

# Chương 6: Deadlock


Sharing is learning

# Chương 6: 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

**Điều kiện xảy ra deadlock:**

- Loại trừ tương hỗ: ít nhất một tài nguyên được giữ theo nonsharable mode

  Ví dụ: printer <> read-only files (sharable)
- Giữ và chờ cấp thêm tài nguyên: Một tiến trình đang giữ ít nhất một tài nguyên và đợi thêm tài nguyên do quá trình khác giữ
- Không trưng dụng: 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
- Chu trình đợi:

# Chương 6: Deadlock

**Các phương pháp giải quyết deadlock:**

- Bảo đảm rằng hệ thống không rơi vào tình trạng deadlock bằng cách ngăn hoặc tránh deadlock
- Khác biệt:

Ngăn deadlock: không cho phép (ít nhất) một trong 4 điều kiện cần cho deadlock

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

Cho phép hệ thống vào trạng thái deadlock, nhưng sau đó phát hiện deadlock và phục hồi hệ thống

# Chương 6: Deadlock

- **Các phương pháp giải quyết deadlock:**

- Bỏ qua mọi vấn đề, xem như deadlock không bao giờ xảy ra trong hệ thống

Khá nhiều hệ điều hành sử dụng phương pháp này

Deadlock không được phát hiện, dẫn đến việc giảm hiệu suất của hệ thống. Cuối cùng, hệ thống có thể ngưng hoạt động và phải khởi động lại

# 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

**Ngăn deadlock: bằng cách ngăn một trong 4 điều kiện cần của deadlock**

- Ngăn mutual exclusion

Đối với tài nguyên không chia sẻ (printer): không làm được

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

- - - Hold and wait:

Cách 1: Mỗi tiến trình yêu cầu toàn bộ tài nguyên cần thiết một lần. Nếu có đủ tài nguyên thì hệ thống sẽ cấp phát, nếu không đủ tài nguyên thì tiến trình phải bị block

Cách 2: Khi yêu cầu tài nguyên, tiến trình không được giữ tài nguyên nào. Nếu đang có thì phải trả lại trước khi yêu cầu

# Chương 6: Deadlock

**Ngăn deadlock: bằng cách ngăn một trong 4 điều kiện cần của deadlock**

- Ngăn no preemption: nếu tiến trình A có giữ tài nguyên và đang yêu cầu tài nguyên khác nhưng tài nguyên này chưa được cấp phát ngay thì:
    - Cách 1: Hệ thống lấy lại mọi tài nguyên mà A đang giữ
    - Cách 2: Hệ thống sẽ xem tài nguyên mà A yêu cầu

# Chương 6: Deadlock

**Ngăn deadlock: bằng cách ngăn một trong 4 điều kiện cần của deadlock**

- Ngăn Circular wait: Mỗi tiến trình chỉ có thể yêu cầu thực thể của một loại tài nguyên theo thứ tự tăng dần (định nghĩa bởi hàm F) của loại tài nguyên.
- Ví dụ: Chuỗi yêu cầu thực thể hợp lệ: tape driver -> disk -> printer

* Khi một tiến trình yêu cầu một thực thể của loại tài nguyên Rj thì nó phải trả lại các tài nguyên Ri với F(Ri ) > F(Rj )

# Chương 6: Deadlock

**Tránh deadlock**

- Ngăn deadlock sử dụng tài nguyên không hiệu quả
- Tránh deadlock 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 deadlock sẽ kiểm tra trạng thái cấp phát tài nguyên để đảm bảo hệ thống không rơi vào deadlock
- Trạng thái cấp phát tài nguyên được định nghĩa dựa trên số tài nguyên còn lại, số tài nguyên đã được cấp phát và yêu cầu tối đa của các tiến trình

# 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 6: Deadlock

**Các giải thuật tránh deadlock**

- Mỗi tài nguyên chỉ có một thực thể -> Giải thuật đồ thị cấp phát tài nguyên
- Mỗi tài nguyên có nhiều thực thể -> Giải thuật Banker

# Chương 6: Deadlock

**Giải thuật tránh deadlock: Giải thuật Banker**

1. Gọi Work và Finish là hai vector độ dài là m và n. Khởi tạo Work = Available Finish[i] = false, i = 0, 1, …, n-1
2. Tìm i thỏa (a) Finish[i] = false (b) Needi ≤ Work (hàng thứ i của Need) Nếu không tồn tại i như vậy, đến bước 4.
3. Work = Work + Allocation
4. Finish[i] = true quay về bước 2

5. Nếu Finish[i] = true, i = 1,…, n, thì hệ thống đang ở trạng thái safe

# Chương 6: Deadlock

Cho sơ đồ sau:

| | Allocation | | | | Max | | | |
|---|---|---|---|---|---|---|---|---|
| Tiến trình | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 |
| P1 | 1 | 2 | 2 | 1 | 3 | 2 | 4 | 3 |
| P2 | 3 | 1 | 1 | 3 | 8 | 2 | 1 | 6 |
| P3 | 5 | 1 | 4 | 2 | 7 | 5 | 5 | 5 |
| P4 | 3 | 1 | 2 | 2 | 3 | 4 | 7 | 6 |
| P5 | 1 | 2 | 1 | 4 | 4 | 6 | 3 | 7 |

| Available | | | |
|---|---|---|---|
| R1 | R2 | R3 | R4 |
| 4 | 2 | 3 | 5 |

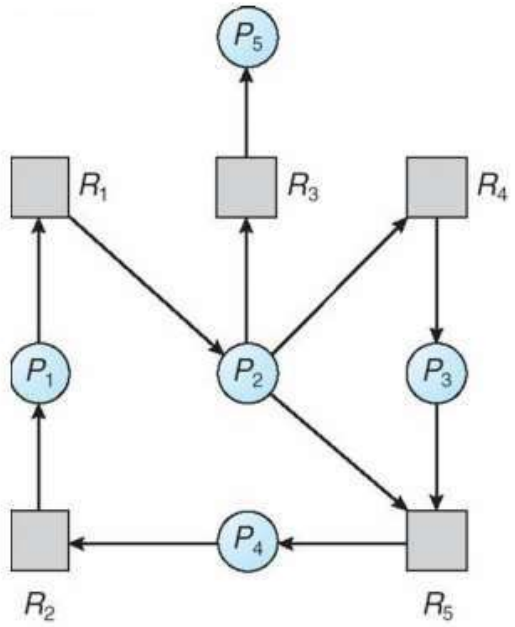
Câu hỏi 19: . Lựa chọn nào dưới đây là một chuỗi an toàn của hệ thống?

A. <P4, P1, P2, P5, P3>  B. <P2, P4, P3, P1, P5>

C. <P3, P1, P5, P4, P2>  D. <P1, P3, P2, P4, P5>

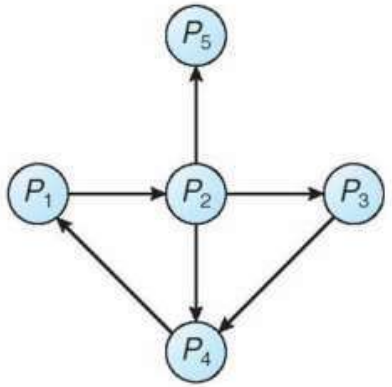
# Chương 6: Deadlock

## Giải thuật phát hiện deadlock: Tài nguyên chỉ có 1 thực thể




(a)

Resource-Allocation Graph




(b)

Corresponding wait-for graph

# Chương 6: Deadlock

**Giải thuật phát hiện deadlock: Tài nguyên có nhiều thực thể**

1. Gọi Work và Finish là vector kích thước m và n. Khởi tạo:
   a. Work = Available
b. For i = 1, 2,…, n, nếu Allocation(i)≠ 0 thì Finish[ i ] := false còn không thì Finish[ i ] := true
2. Tìm i thỏa mãn: a. Finish[ i ] = false
   b. Request(i) ≤ Work Nếu không tồn tại i như vậy, đến bước 4.
3. a. Work = Work + Allocation(i)
   b. Finish[ i ] = true, quay về bước 2.

Nếu Finish[ i ] = false, với một số i = 1,…, n, thì hệ thống đang ở trạng thái deadlock. Hơn thế nữa, Finish[ i ] = false thì Pi bị deadlocked.

# Chương 6: Deadlock

Cho sơ đồ sau:

| | Allocation | | | | Max | | | |
|---|---|---|---|---|---|---|---|---|
| Tiến trình | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 |
| P1 | 1 | 2 | 2 | 1 | 3 | 2 | 4 | 3 |
| P2 | 3 | 1 | 1 | 3 | 8 | 2 | 1 | 6 |
| P3 | 5 | 1 | 4 | 2 | 7 | 5 | 5 | 5 |
| P4 | 3 | 1 | 2 | 2 | 3 | 4 | 7 | 6 |
| P5 | 1 | 2 | 1 | 4 | 4 | 6 | 3 | 7 |

| Available | | | |
|---|---|---|---|
| R1 | R2 | R3 | R4 |
| 4 | 2 | 3 | 5 |

Câu hỏi 10: Yêu cầu cấp phát nào sau đây sẽ được đáp ứng? (G1)

A. P4 yêu cầu thêm tài nguyên (1, 2, 3, 4)

B. P3 yêu cầu thêm tài nguyên (2, 2, 1, 3)

C. P5 yêu cầu thêm tài nguyên (3, 2, 4, 3)

D. P2 yêu cầu thêm tài nguyên (2, 1, 0, 2)

# Chương 6: Deadlock

**Câu 1 (1đ)(TL):** Xét một hệ thống máy tính có 5 tiến trình: P1, P2, P3, P4, P5 và 4 loại tài nguyên: R1, R2, R3, R4. Tại thời điểm t0, trạng thái của hệ thống như sau:

| | Allocation | | | | Max | | | |
|---|---|---|---|---|---|---|---|---|
| Tiến trình | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 |
| P1 | 1 | 2 | 2 | 3 | 2 | 3 | 4 | 3 |
| P2 | 3 | 1 | 3 | 1 | 3 | 8 | 6 | 1 |
| P3 | 2 | 1 | 4 | 5 | 7 | 7 | 5 | 7 |
| P4 | 3 | 1 | 5 | 2 | 5 | 4 | 6 | 7 |
| P5 | 1 | 4 | 4 | 2 | 1 | 6 | 7 | 3 |

| Available | | | |
|---|---|---|---|
| R1 | R2 | R3 | R4 |
| 3 | 4 | 4 | 3 |

Tại thời điểm t1, nếu tiến trình P4 yêu cầu thêm tài nguyên (2, 3, 1, 3), hệ thống có đáp ứng không và giải thích tại sao? Biết hệ điều hành dùng giải thuật Banker để kiểm tra độ an toàn của hệ thống.

# Chương 6: Deadlock

**Giải:**

B1: Test: request 1 (2,3,1,3) < available (3,4,4,3) => **True**

=> Tại thời điểm T1, Nếu P4 yêu cầu thêm tài nguyên(2, 3, 1, 3) thì hệ thống được đáp ứng.

**New state:**

| | ALLOCATION | | | | MAX | | | | NEED | | | | AVAILABLE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Process | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 | R1 | R2 | R3 | R4 | |
| P1 | 1 | 2 | 2 | 3 | 2 | 3 | 4 | 3 | 1 | 1 | 2 | 0 | 1 | 1 | 3 | 0 | |
| P2 | 3 | 1 | 3 | 1 | 3 | 8 | 6 | 1 | 0 | 7 | 3 | 0 | 2 | 3 | 5 | 3 | P1 |
| P3 | 2 | 1 | 4 | 5 | 7 | 7 | 5 | 7 | 5 | 6 | 1 | 2 | 7 | 7 | 11 | 8 | P4 |
| P4 | 5 | 4 | 6 | 5 | 5 | 4 | 6 | 7 | 0 | 0 | 0 | 2 | 10 | 8 | 14 | 9 | P2 |
| P5 | 1 | 4 | 4 | 2 | 1 | 6 | 7 | 3 | 0 | 2 | 3 | 1 | 12 | 9 | 18 | 14 | P3 |
| | | | | | | | | | | | | | 13 | 13 | 22 | 16 | P5 |

Chuỗi an toàn: P1 - P4 - P2 - P3 - P5

# Chương 6: Deadlock

**Phục hồi deadlock:**

- Khi deadlock xảy ra, để 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
    - Lấy lại tài nguyên từ một hay nhiều tiến trình


Sharing is learning

# Chương 6: Deadlock

**Phục hồi deadlock:**

- Chấm dứt quá trình bị deadlock

+ Chấm dứt lần lượt từng tiến trình cho đến khi không còn deadlock

+ Sử dụng giải thuật phát hiện deadlock để xác định còn deadlock hay không

Những yếu tố để chấm dứt tiến trình:

- Độ ưu tiên của tiến trình
- Thời gian đã thực thi của tiến trình và thời gian còn lại
- Loại tài nguyên mà tiến trình đã sử dụng
- Tài nguyên mà tiến trình cần thêm để hoàn tất công việc Số lượng tiến trình cần được chấm dứt
- Tiến trình là interactive hay batch

# Chương 7:
# Quản lý bộ nhớ

# Chương 7: Quản lý bộ nhớ

**Khái niệm cơ sở:**

- Một chương trình phải được mang vào trong bộ nhớ chính và đặtnó trong một tiến trình để được xử lý

- 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 lý bộ nhớ

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

- 5 nhiệm vụ chính: cấp phát bộ nhớ cho process, tái định vị, bảo vệ, chia sẻ, kết gán địa chỉ nhớ luận lý

- Mục tiêu: Nạp càng nhiều process vào bộ nhớ càng tốt

# Chương 7: Quản lý bộ nhớ

**Địa chỉ bộ nhớ:**

Địa chỉ vật lý (physical address – địa chỉ thực): là một vị trí thực trong bộ nhớ chính

- 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 lý bộ nhớ

**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

- 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

# Chương 7: Quản lý bộ nhớ

**Câu 5: Khi nạp chương trình vào bộ nhớ, bộ linker có chức năng gì?**

A.Nạp load module vào bộ nhớ chính.

B.Kết hợp các object module thành một file nhị phân khả thực thi gọi là load module.

C.Nạp object module vào bộ nhớ chính.

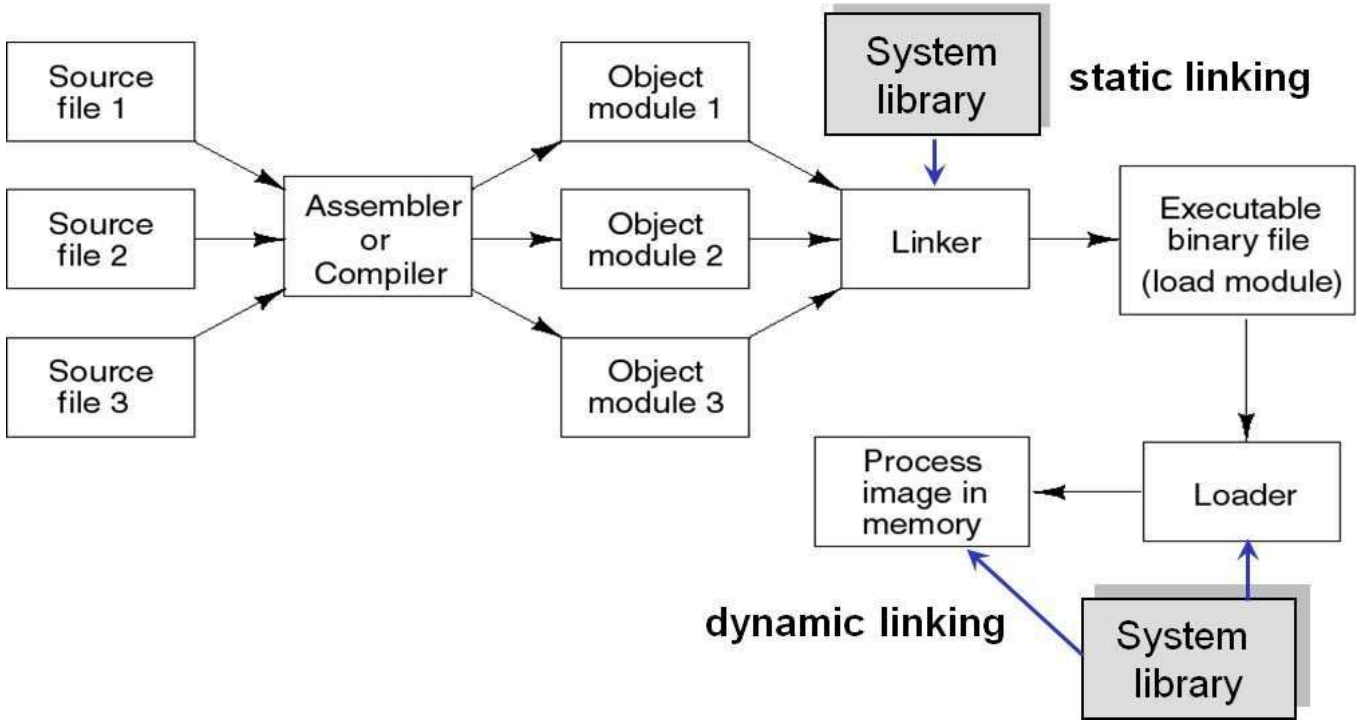
D.Kết hợp các load module.

# Chương 7: Quản lý bộ nhớ

**Nạp chương trình vào bộ nhớ**

**-Linker:** kết hợp nhiều object module thành một file nhị phân (file tạo thành gọi là load module).

- **Loader:** nạp module vào bộ nhớ chính.



# Chương 7: Quản lý bộ nhớ

**Câu 14: 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 lý bộ nhớ

**Hiện tượng phân mảnh bộ nhớ (fragmentation):**

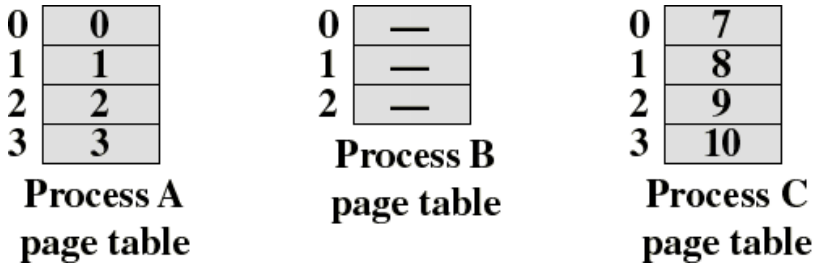
**Phân mảnh ngoại** (external fragmentation): Kích thước không gian bộ nhớ trống đủ để thỏa mãn một yêu cầu cấp phát, tuy nhiên không gian nhớ này không liên tục

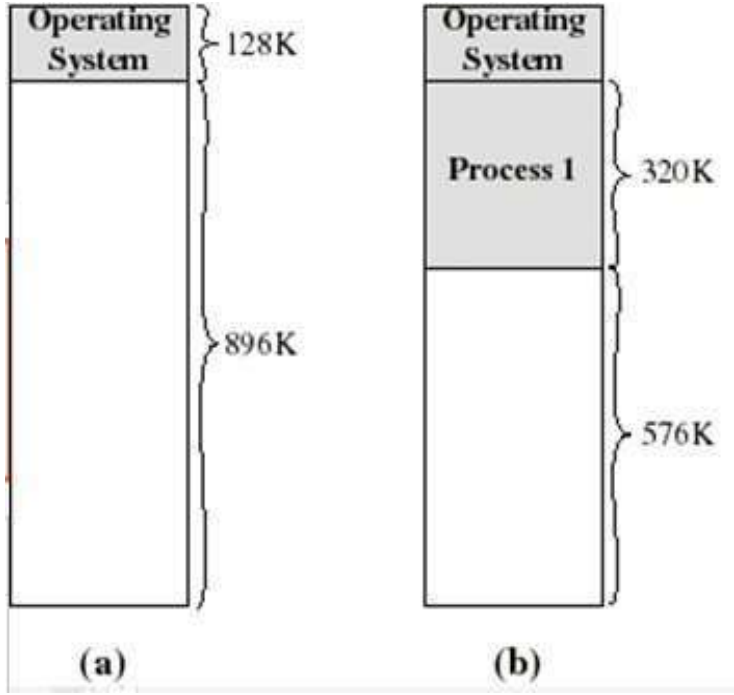
⇒ Giải pháp: Dùng cơ chế kết khối (compaction) để gom lại thành vùng nhớ liên tục

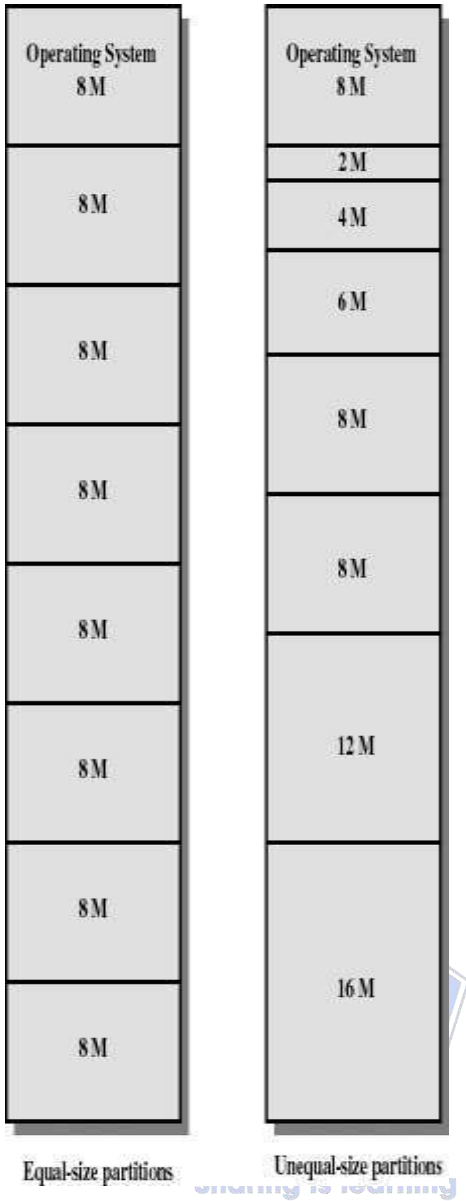
**Phân mảnh nội** (internal fragmentation): Kích thước vùng nhớ cấp phát có thể hơi lớn hơn vùng nhớ yêu cầu

⇒ Giải pháp: Dùng chiến lược placement

# Chương 7: Quản lý bộ nhớ




**Các mô hình quản lý bộ nhớ:**

Một số cơ chế quản lý bộ nhớ:

1. Phân chia cố định (fixed partitioning)
2. Phân chia động (dynamic partitioning)
3. Phân trang đơn giản (simple paging)

# Chương 7: Quản lý bộ nhớ

**Câu 9: 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 lý bộ nhớ

**Phân chia cố định (Fixed partitioning)**

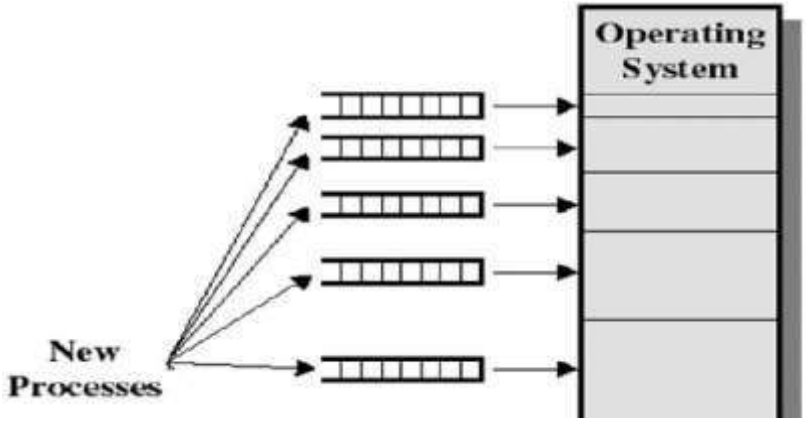
- Bộ nhớ chính chia thành nhiều phần, có kích thước bằng hoặc khác nhau
- Process nào nhỏ hơn kích thước partition thì có thể nạp vào

-Nếu process có kích thước lớn hơn => overlay

Nhận xét: Kém hiệu quả do bị phân mảnh nội

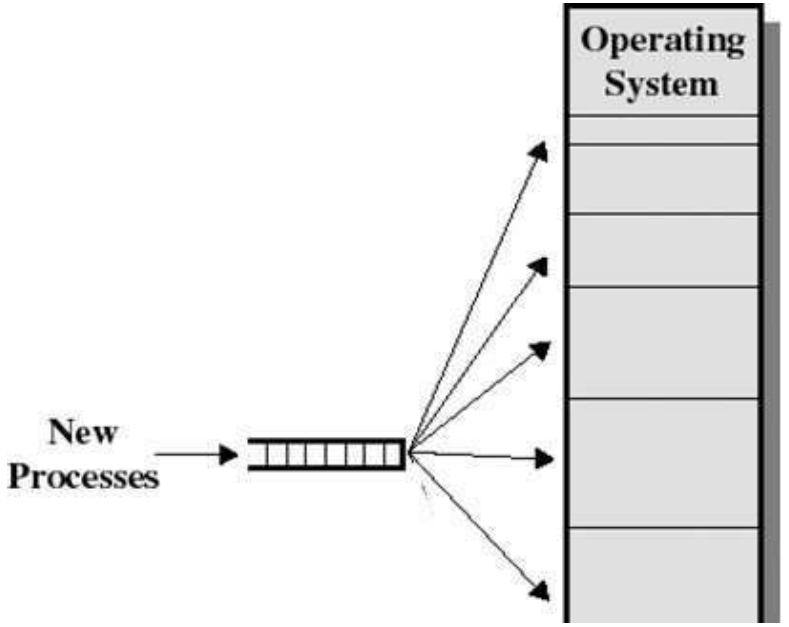
# Chương 7: Quản lý bộ nhớ

**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, nhương chỗ cho process mới
- Với partition khác kích thước:
- Giải pháp 1: Sử dụng nhiều hàng đợi
- Giải pháp 2: Sử dụng 1 hàng đợi



# Chương 7: Quản lý bộ nhớ

**Phân chia động (dynamic partitioning)**

- Số lượng partitionvà kích thước không cố định, có thể khác nhau

-Mỗi process được cấp phát chính xác dung lượng bộ nhớ cần thiết

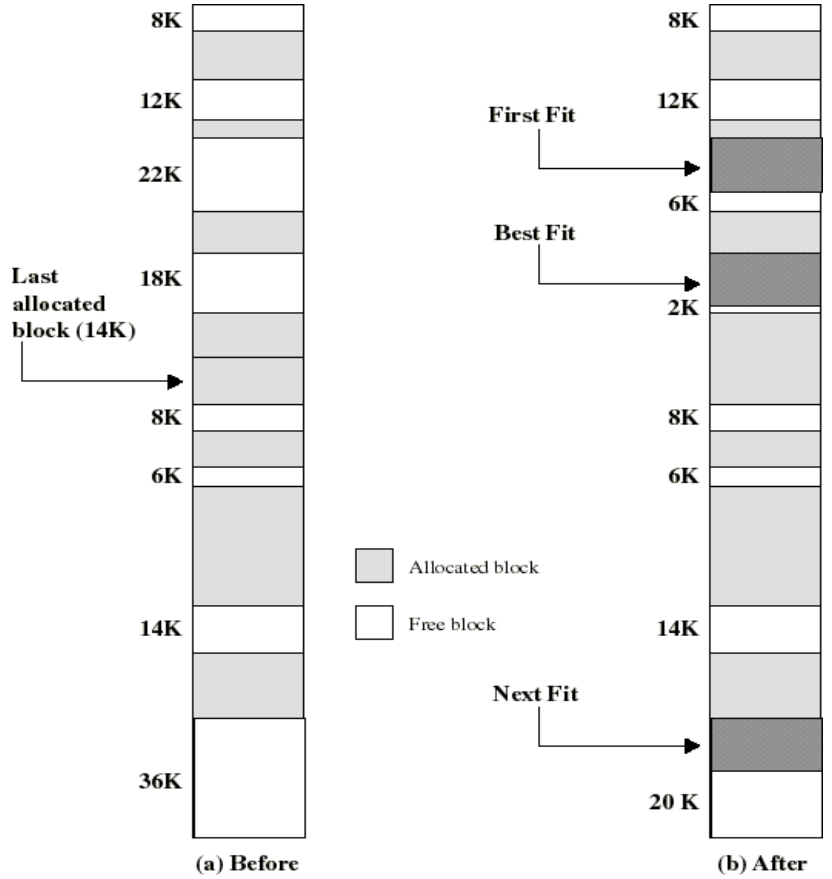
Nhận xét: Gây hiện tượng phân mảnh ngoại

# Chương 7: Quản 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




Example Memory Configuration Before and After Allocation of 16 Kby

# Chương 7: Quản lý bộ nhớ

**Câu 10: Có bao nhiêu phát biểu đúng trong các phát biểu sau?**

(1) Cơ chế quản lý bộ nhớ phân chia cố định có thể xảy ra hiện tượng phân mảnh ngoại.

(2) Cơ chế quản lý bộ nhớ phân chia động có thể xảy ra hiện tượng phân mảnh nội.

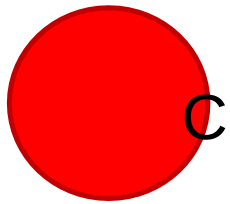
(3) Trong cơ chế quản lý bộ nhớ phân chia động, mỗi tiến trình được cấp phát chính xác dung lượng bộ nhớ cần thiết.

(4) Trong cơ chế quản lý bộ nhớ phân chia cố định, tại chiến lược Placement, khi partition có kích thước không bằng nhau, ta có 2 giải pháp để giải quyết.

(5) Cơ chế quản lý bộ nhớ phân chia cố định, nếu chương trình có kích thước lớn hơn partition thì phải dùng cơ chế overlay.

(6) Chuyển đổi địa chỉ là quá trình ánh xạ một địa chỉ từ không gian địa chỉ này sang không gian địa chỉ khác.

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




Sharing is learning

# Chương 7: Quản lý bộ nhớ

**Câu 21: 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 lý bộ nhớ

***Sử dụng các dữ liệu sau để trả lời câu hỏi 28, 29,30:***
***Xét một không gian địa chỉ ảo có 64 trang, mỗi trang có kích thước 2048 byte được ánh xạ vào bộ nhớ vật lý có 32 khung trang.***

**Câu 28: Địa chỉ luận lý gồm bao nhiêu bit?**

A.17 B. 6 C. 32 D. 11

**Câu 29: Địa chỉ vật lý gồm bao nhiêu bit?**

A.5 B. 16 C. 7 D. 11

**Câu 30: Bảng phân trang có tất cả bao nhiêu mục?**

A.32 B.11 C. 2048 D. 64

# Chương 7: Quản lý bộ nhớ

**Câu 26: 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

# Chương 7: 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)

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 = $(2-\propto)x + \varepsilon$

# Chương 7: Quản lý bộ nhớ

**- Hit ratio = 0.7**

**- Thời gian truy xuất bộ nhớ trong hệ thống EAT = 300ns**

**-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

Thời gian truy xuất hiệu dụng $EAT = (2-\propto)x + \varepsilon$

$=> 300 = (2-0,7).200 + \varepsilon$

$=> \varepsilon = 40$ ns

# Chương 8:
# Bộ nhớ ảo

# Chương 8: Bộ nhớ ảo

**Câu 27: 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

**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

**Câu 3: Sắp xếp các bước sau theo quy trình của PFSR (Page-fault service routine).**

(1) Chuyển process về trạng thái blocked.

(2) Sau khi I/O hoàn tất, đĩa gây ra một ngắt đến hệ điều hành; PFSR cập nhập page table và chuyển process về trạng thái ready.

(3) Phát ra yêu cầu đọc đĩa để nạp trang được tham chiếu vào một frame trống; trong khi đợi I/O, một process khác được cấp vào CPU để thực thi.

A. (1), (2), (3)

B. (1), (3), (2)

C. (2), (1), (3)

D. (2), (3), (1)

# Chương 8: Bộ nhớ ảo

**Cài đặt bộ nhớ ảo**

Có hai kỹ thuật

- Phân trang theo yêu cầu (demand paging)
- Phân đoạn theo yêu cầu (demand segmentation)

# Chương 8: Bộ nhớ ảo

**Phân trang theo yêu cầu**

- 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

**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

**Câu 24: 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

**Vấn đề cấp phát Frames**

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

**Câu 13: Phát biểu nào sau đây là đúng về nghịch lý Belady?**

A. Nghịch lý Belady là số page fault giảm mặc dù quá trình đã được cấp nhiều frame hơn.

B. Nghịch lý Belady là số page fault tăng khi quá trình được cấp ít frame hơn.

C. Nghịch lý Belady là số page fault giảm khi quá trình được cấp ít frame hơn

D. Nghịch lý Belady là số page fault tăng mặc dù quá trình đã được cấp nhiều frame hơn.

# Chương 8: Bộ nhớ ảo

**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

**Câu 15: 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 theo yêu cầu thì số frame cấp cho mỗi process 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 2: (1,5 điểm)*** Giả sử một tiến trình được phát 4 khung trang (frame) trong bộ nhớ vật lý và 6 trang (page) trong bộ nhớ ảo. Biết ban đầu, khi nạp tiến trình vào, 4 frame trên bộ nhớ vật lý này đang trống. Process truy xuất 6 trang (1, 2, 3, 4, 5, 6) trong bộ nhớ ảo theo thứ tự như sau:

**1 3 4 2 6 5 1 2 3 4 6 3 2 5 4 2 5 4 1 6**

Vẽ bảng minh họa thuật toán và tính số lỗi trang (page fault) khi:

A. Tiến trình truy xuất chuỗi bộ nhớ trên và hệ điều hành thay trang theo giải thuật FIFO.

B. Tiến trình truy xuất chuỗi bộ nhớ trên và hệ điều hành thay trang theo giải thuật LRU.

# Chương 8: Bộ nhớ ảo


## Theo giải thuật FIFO

| | 1 | 3 | 4 | 2 | 6 | 5 | 1 | 2 | 3 | 4 | 6 | 3 | 2 | 5 | 4 | 2 | 5 | 4 | 1 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 1 |
| 1 | | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| 2 | | | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | | | | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| | * | * | * | * | * | * | * | | * | * | * | | * | * | | | | | * | |

# Chương 8: Bộ nhớ ảo


## Theo giải thuật LRU

| | 1 | 3 | 4 | 2 | 6 | 5 | 1 | 2 | 3 | 4 | 6 | 3 | 2 | 5 | 4 | 2 | 5 | 4 | 1 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 |
| 1 | | 3 | 3 | 3 | 3 | 5 | 5 | 5 | 5 | 4 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 2 | | | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 6 | 6 | 6 | 6 | 4 | 4 | 4 | 4 | 4 | 4 |
| 3 | | | | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6 |
| | * | * | * | * | * | * | * | | * | * | * | | | * | * | | | | * | * |

# BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

## TRAINING GIỮA KỲ HỌC KỲ I NĂM HỌC 2022 – 2023


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*