

# Hệ Điều Hành

(*Nguyên lý các hệ điều hành*)

Bộ môn Khoa Học Máy Tính  
Viện Công Nghệ Thông Tin và Truyền  
Thông

# Chương 2 Quản lý tiến trình

## ① Định nghĩa tiến trình

# Chương 2 Quản lý tiến trình

Tiến trình (TT)

(nhắc lại)

## ● Khi chương trình đang thực hiện

- Được cung cấp tài nguyên (CPU, bộ nhớ, thiết bị vào/ra. . .) để hoàn thành công việc
- Tài nguyên được cấp khi:
  - Bắt đầu c/trình
  - Trong khi c/trình đang thực hiện
- Gọi là tiến trình (process)

## ● Hệ thống bao gồm tập các TT thực hiện đồng thời

- TT hệ điều hành: T/hiện mã lệnh hệ thống
- TT người dùng: T/hiện mã lệnh người dùng

## ● Có thể chứa 1 hoặc nhiều tiến trình

# Chương 2 Quản lý tiến trình

## Tiến trình (nhắc lại)

### ● Trách nhiệm của HĐH:

- Đảm bảo hoạt động của TT và tiểu trình (luồng)
- Tạo/xóa TT (người dùng, hệ thống)
- Điều phối TT
- Cung cấp cơ chế đồng bộ, truyền thông và ngăn ngừa tình trạng bế tắc giữa các TT

## Chương 2 Quản lí tiến trình

- ① Tiến trình
- ② Luồng (Thread)
- ③ Điều phối CPU
- ④ Tài nguyên găng và điều độ tiến trình
- ⑤ Bẽ tắc và xử lý bẽ tắc

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

- Khái niệm tiến trình
- Điều phối tiến trình (Process Scheduling)
- Thao tác trên tiến trình
- Hợp tác tiến trình
- Truyền thông liên tiến trình

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

# Tiến trình và trạng thái hệ thống

## • Trạng thái hệ thống

- **Vi xử lý:** Giá trị các thanh ghi
- **Bộ nhớ:** Nội dung các ô nhớ
- **Thiết bị ngoại vi:** Trạng thái thiết bị

## 🟡 Thực hiện chương trình $\Rightarrow$ Trạng thái hệ thống thay đổi

- Thay đổi rời rạc, theo từng câu lệnh được thực hiện



## 🟡 TT là một dãy thay đổi trạng thái của hệ thống

- Xuất phát từ 1 trạng thái ban đầu
- Chuyển từ trạng thái này sang trạng thái khác được thực hiện theo yêu cầu nằm trong chương trình của người sử dụng

*Tiến trình là sự thực hiện chương trình*

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Tiến trình >< chương trình

🟡 **Chương trình:** thực thể thụ động (nội dung file trên đĩa)

- **Mã chương trình:** Lệnh máy (CD2190EA...)
- **Dữ liệu:**

- **Biến được lưu trữ và sử dụng trong bộ nhớ**
- **Biến toàn cục**
- **Biến được cung cấp động (malloc, new,...)**
- **Biến stack (tham số hàm, biến cục bộ)**

- **Thư viện liên kết động (DLL)**
  - Không được dịch & liên kết cùng với chương trình

Khi chương trình đang thực hiện, tài nguyên tối thiểu cần có

- **Bộ nhớ cho mã chương trình và dữ liệu**
- **Các thanh ghi của VXL phục vụ cho quá trình thực hiện**

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

Tiến trình >< chương trình

● **Tiến trình: thực thể chủ động (bộ đếm lệnh, tập tài nguyên)**

Một chương trình có thể

● Chỉ là 1 phần của trạng thái tiến trình

● 1 chương trình, nhiều TT ( bộ dữ liệu khác nhau)

VD: `gcc hello.c || gcc baitap.c`

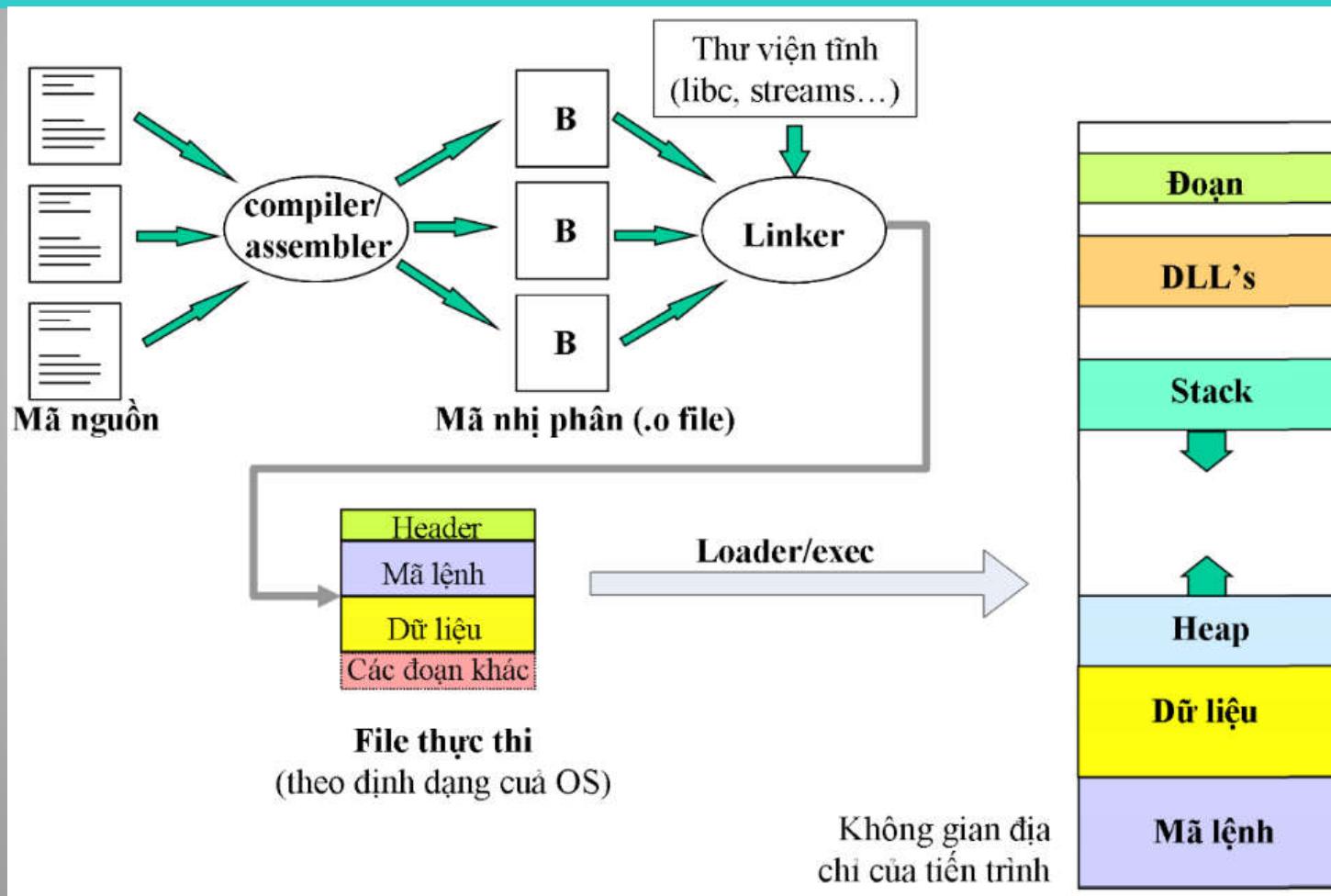
● Gọi tới nhiều TT

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Dịch và thực hiện một chương trình



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

### Dịch và thực hiện một chương trình

- *HĐH tạo 1 TT và phân phối vùng nhớ cho nó*
- *Bộ thực hiện (loader/exec)*
  - *Đọc và dịch (interprets) file thực thi (header file)*
  - *Thiết lập không gian địa chỉ cho TT để chứa mã lệnh và dữ liệu từ file thực thi*
  - *Đặt các tham số dòng lệnh, biến môi trường (argc, argv, envp) vào stack*
  - *Thiết lập các thanh ghi của VXL tới các giá trị thích hợp và gọi hàm "\_start()" (hàm của HĐH)*

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

### Dịch và thực hiện một chương trình

● *Chương trình bắt đầu thực hiện tại "`_start()`".*

*Hàm này gọi tới hàm `main()` (hàm của chương*

*trình)*

*⇒ "Tiến trình" đang thực hiện, không còn đê cập  
đến "chương trình" nữa*

● *Khi hàm `main()` kết thúc, OS gọi tới hàm*

*"`exit()`" để hủy bỏ TT và thu hồi tài nguyên*

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

### Trạng thái tiến trình

Khi thực hiện, TT thay đổi trạng thái

- **Khởi tạo (New)** TT đang được khởi tạo
- **Sẵn sàng (Ready)** TT đang đợi sử dụng processor vật lý
- **Thực hiện (Running)** Các câu lệnh của TT đang được thực hiện
- **Chờ đợi (Waiting)** TT đang chờ đợi 1 sự kiện nào đó xuất hiện (sự hoàn thành thao tác vào/ra)
- **Kết thúc (Terminated)** TT thực hiện xong

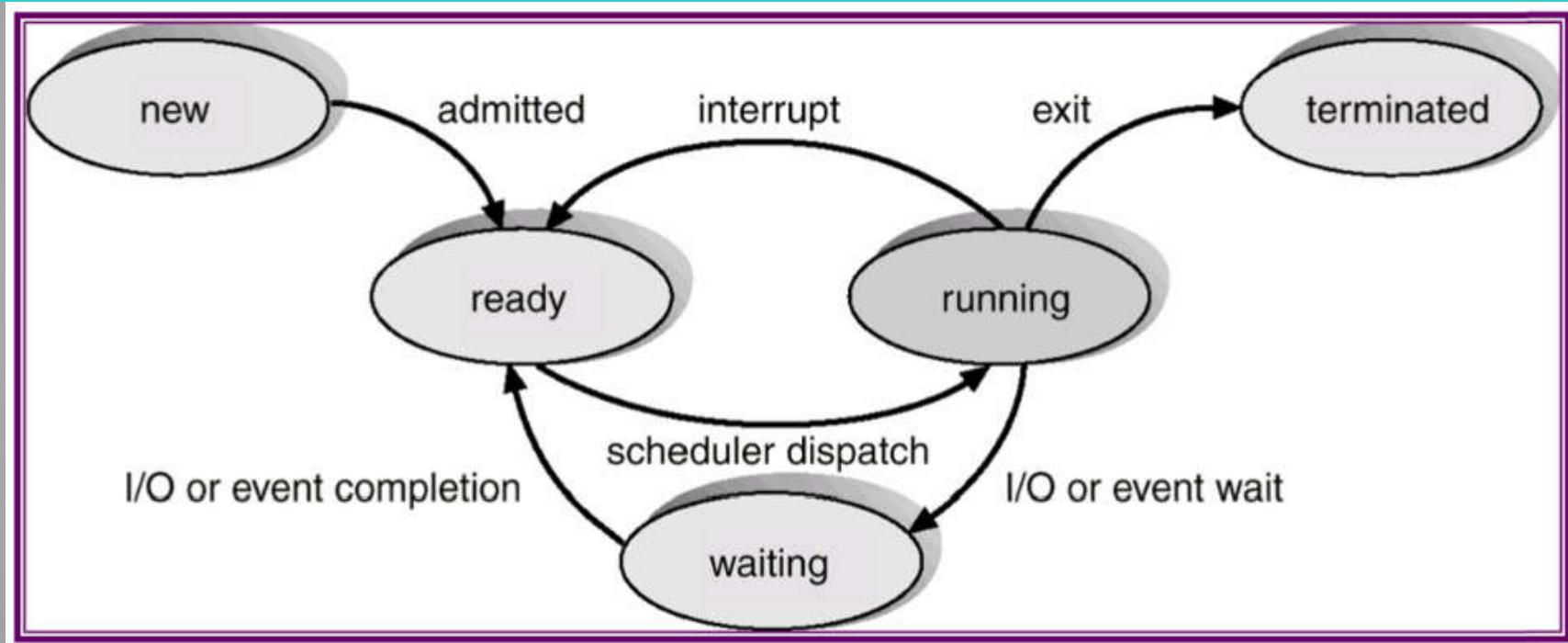
*Trạng thái của TT là một phần trong hoạt động hiện tại của TT*

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

### Lưu đồ thay đổi trạng thái tiến trình (Silberschatz 2002)



Hệ thống có một processor

- Có duy nhất 1 TT ở trạng thái thực hiện
- Có thể có nhiều TT ở trạng thái chờ đợi hoặc sẵn sàng

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Trạng thái tiến trình



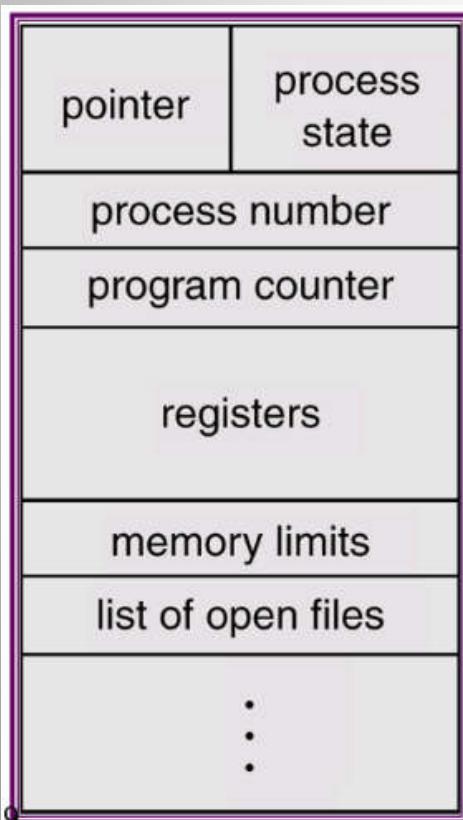
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Khối điều khiển tiến trình (PCB: Process Control Block)

- Mỗi TT được thể hiện trong hệ thống bởi 1 khối điều khiển TT
- PCB: cấu trúc thông tin cho phép xác định duy nhất 1 TT



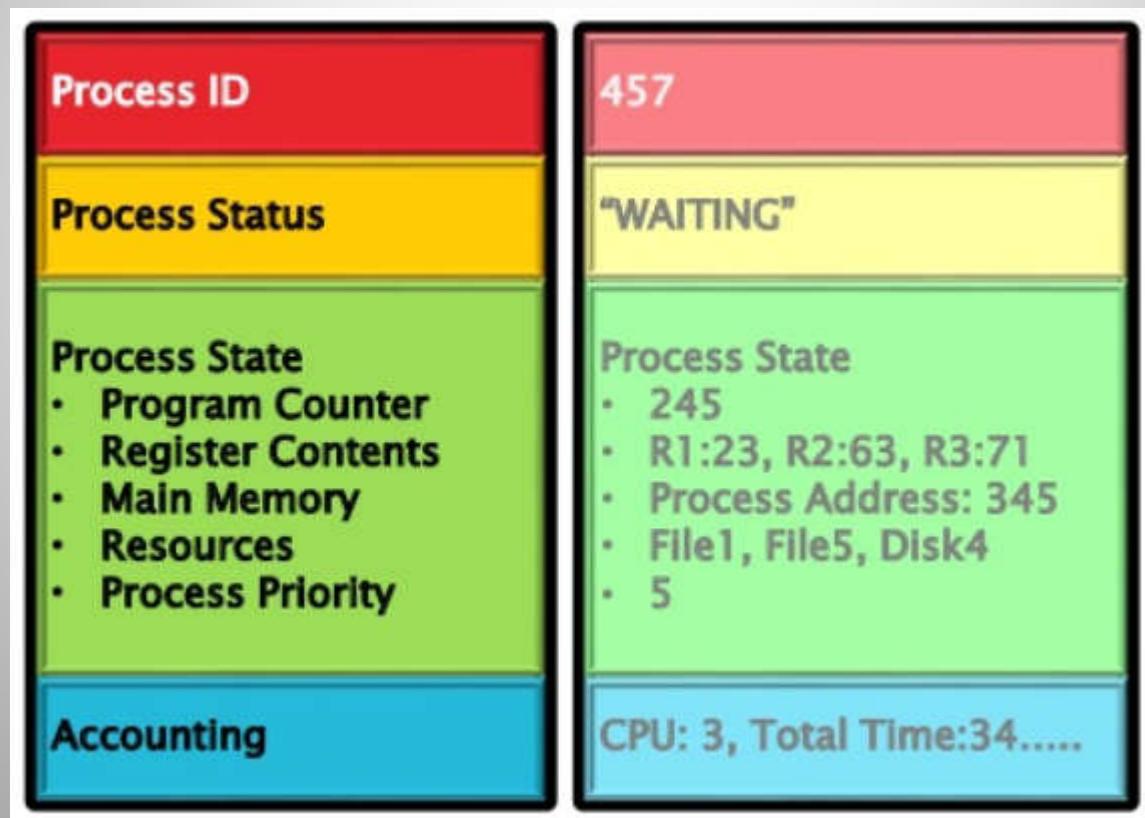
- Trạng thái TT
- Bộ đếm lệnh
- Các thanh ghi của CPU
- Thông tin dùng để điều phối TT
- Thông tin quản lý bộ nhớ
- Thông tin tài nguyên có thể sử dụng
- Thông tin thống kê
- Con trỏ tới 1 PCB khác
- ...

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Khối điều khiển tiến trình (PCB: Process Control Block)

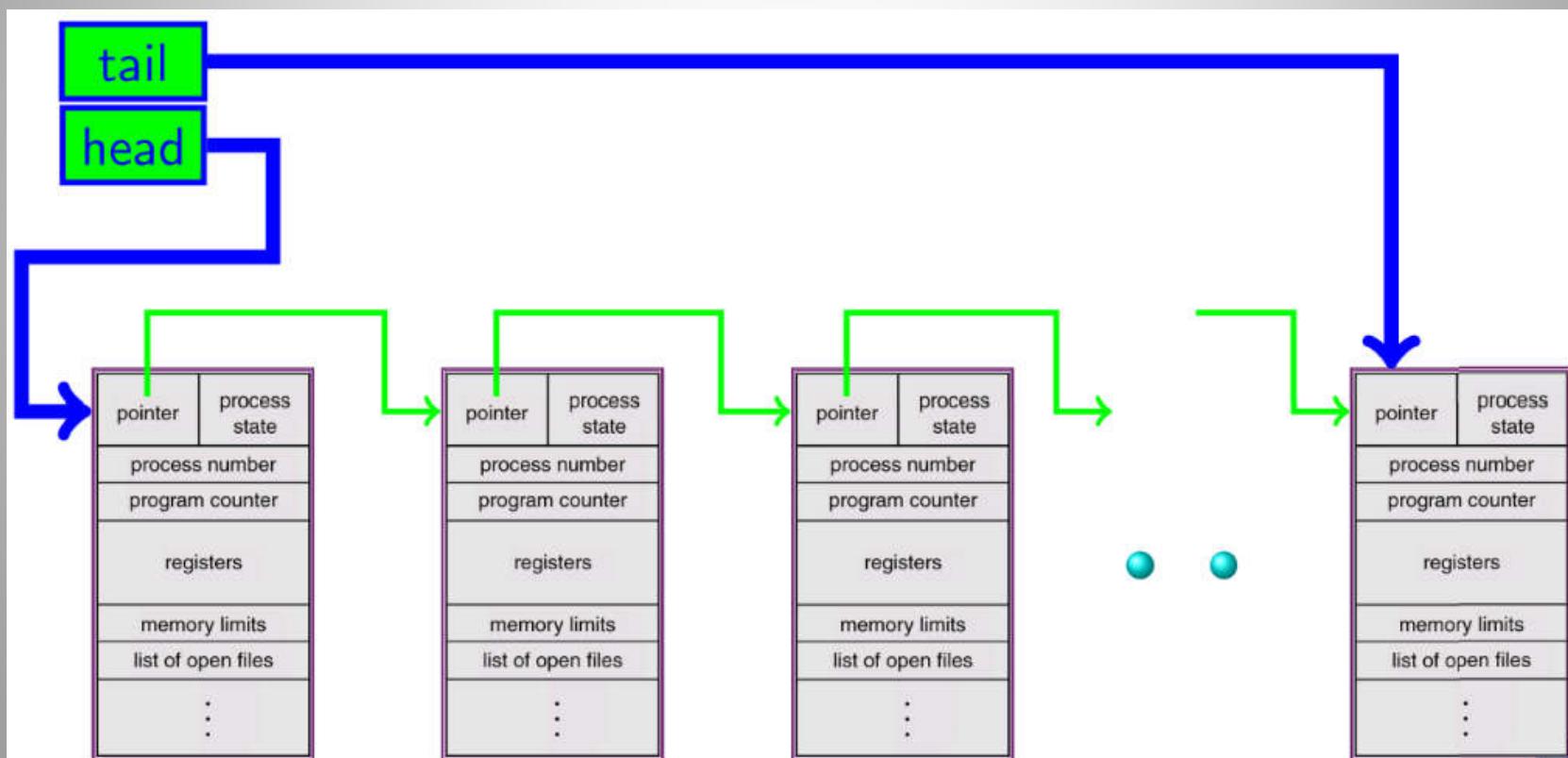


## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Danh sách tiến trình



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Khái niệm tiến trình

## Tiến trình đơn luồng và tiến trình đa luồng

- TT đơn luồng : Là TT thực hiện *chỉ 1 luồng thực thi*  
⇒ Cho phép thực hiện chỉ 1 nhiệm vụ tại 1 thời điểm
- TT đa luồng : Là TT có *nhiều luồng thực thi*  
⇒ Cho phép thực hiện **nhiều** hơn 1 nhiệm vụ tại 1 thời điểm

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.1. Điều phối tiến trình

- Khái niệm tiến trình
- Điều phối tiến trình (Process Scheduling)
- Thảo tác trên tiến trình
- Hợp tác tiến trình
- Truyền thông liên tiến trình

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Giới thiệu

### Trong hệ thống đa nhiệm

Mục đích Sử dụng tối đa thời gian của CPU

⇒ Cần có nhiều TT trong hệ thống

Vấn đề Luân chuyển CPU giữa các TT

⇒ Phải có hàng đợi cho các TT

### Hệ thống 1 processor

⇒ 1 TT thực hiện

⇒ Các TT khác phải đợi tới khi CPU tự do

## Chương 2 Quản lý tiến trình

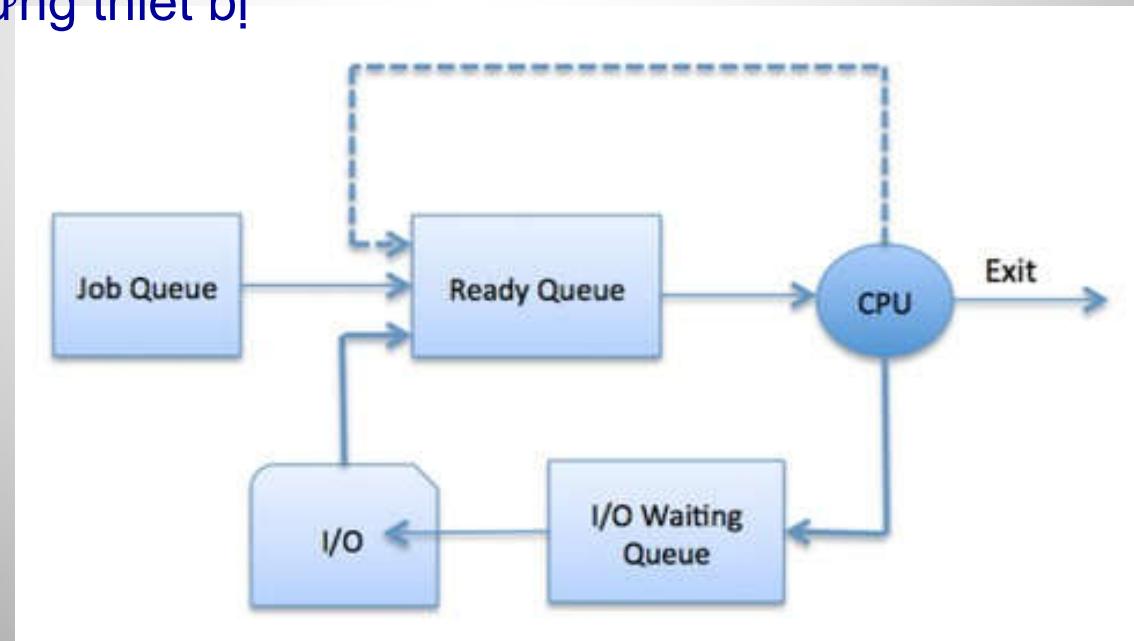
### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Các hàng đợi tiến trình I

Hệ thống có nhiều hàng đợi dành cho TT

- **Job-queue** Tập các TT trong hệ thống
- **Ready-Queue** Tập các TT tồn tại trong bộ nhớ, đang sẵn sàng và chờ đợi để được thực hiện
- **Device queues** Tập các TT đang chờ đợi 1 thiết bị vào ra. Phân biệt hàng đợi cho từng thiết bị

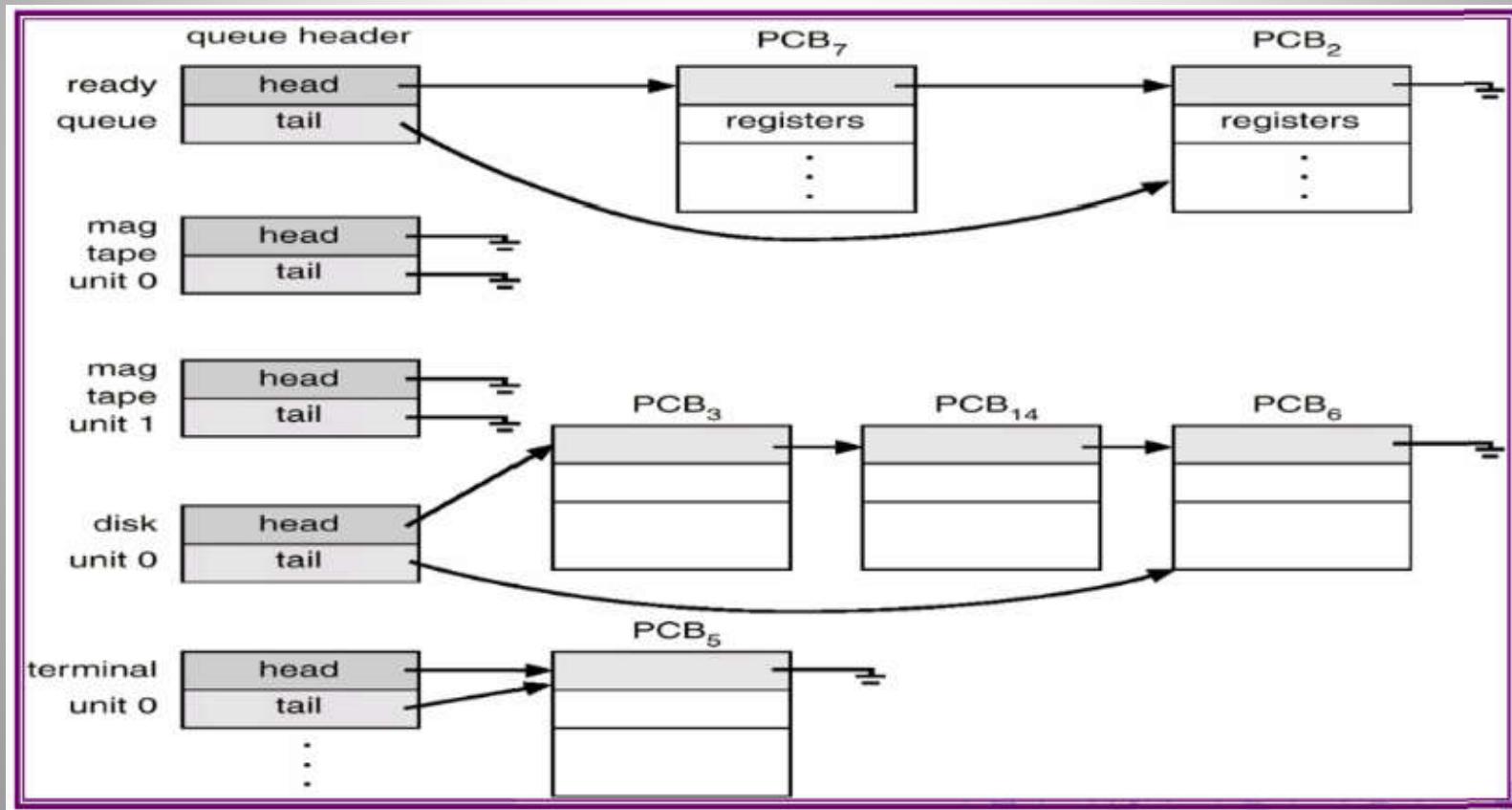


## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Các hàng đợi tiến trình II



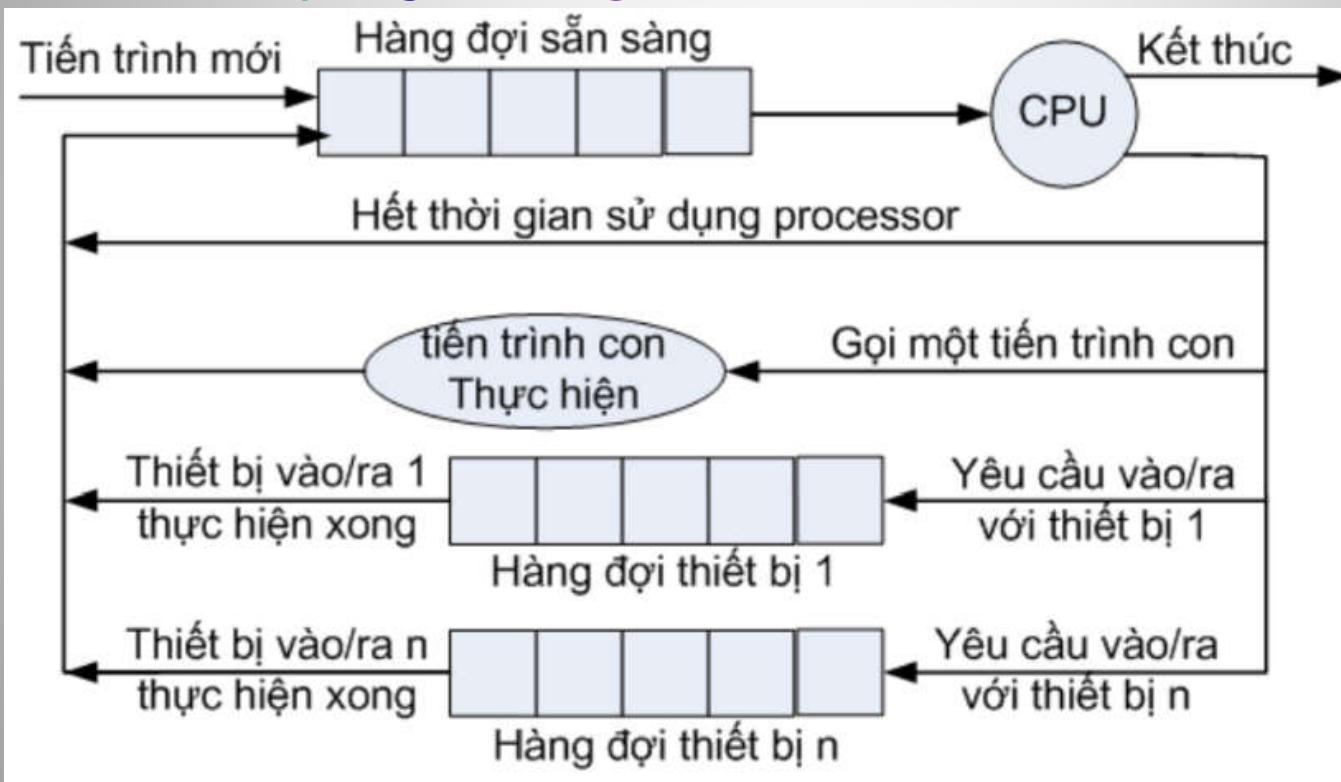
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Các hàng đợi tiến trình III

- Các TT di chuyển giữa hàng đợi khác nhau



- TT mới tạo, được đặt trong hàng đợi sẵn sàng, và đợi cho tới khi được lựa chọn để thực hiện

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

### Các hàng đợi tiến trình IV

- TT đã được chọn và đang thực hiện
  - ① Đưa ra 1 yêu cầu vào ra -> đợi trong 1 hàng đợi thiết bị
  - ② Tạo 1 TT con và đợi TT con kết thúc
  - ③ Hết thời gian sử dụng CPU, phải quay lại hàng đợi sẵn sàng
- Trường hợp (1&2) sau khi sự kiện chờ đợi hoàn thành,
  - TT sẽ chuyển từ t/thái đợi -> t/thái sẵn sàng
  - TT quay lại hàng đợi sẵn sàng
- TT tiếp tục chu kỳ (*sẵn sàng, thực hiện, chờ đợi*) cho tới khi kết thúc
  - Xóa khỏi tất cả các hàng đợi
  - PCB và tài nguyên đã cấp được giải phóng

## Chương 2 Quản lý tiến trình

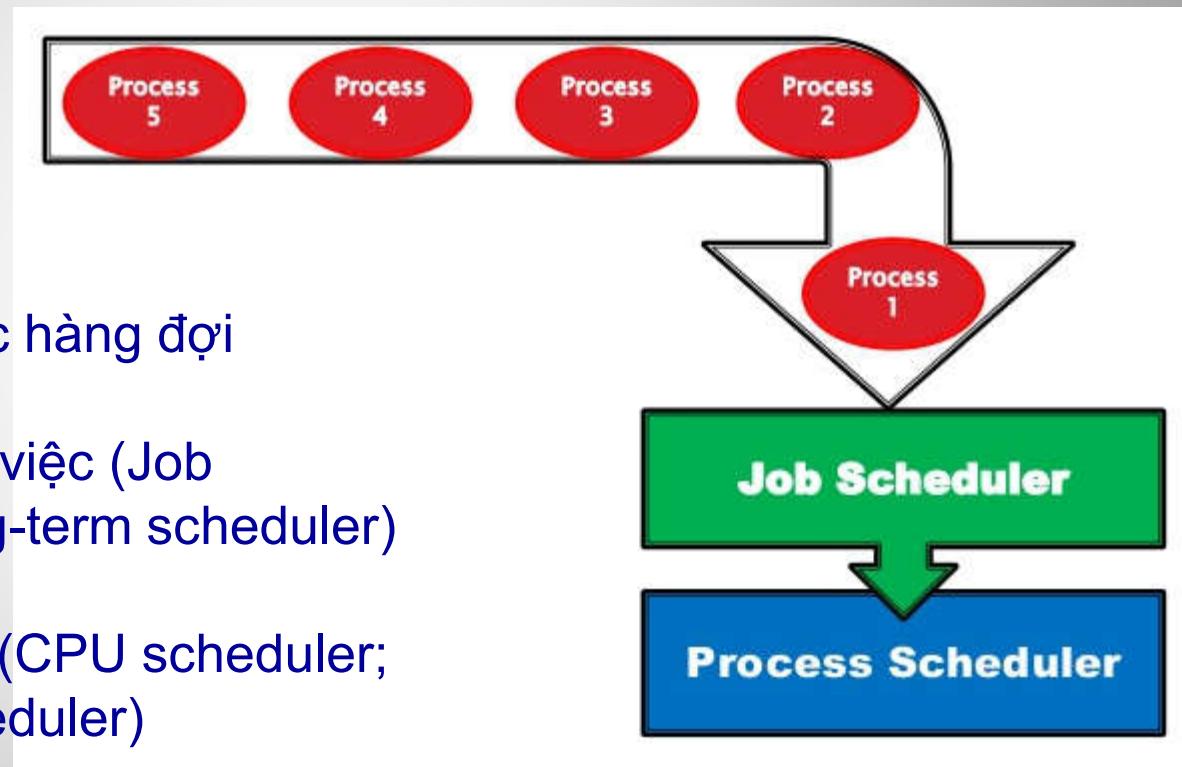
### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Bộ điều phối (Scheduler)

Lựa chọn TT trong các hàng đợi

- Điều phối công việc (Job scheduler; Long-term scheduler)
- Điều phối CPU (CPU scheduler; Short-term scheduler)



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

### Điều phối công việc

- Chọn các TT từ hàng đợi vào được lưu trong các vùng đệm (đĩa tù) và đưa vào bộ nhớ để thực hiện
- Thực hiện không thường xuyên (đơn vị giây/phút)
- Điều khiển mức độ đa chương trình (số TT trong bộ nhớ)
- Khi mức độ đa chương trình ổn định, điều phối công việc được gọi chỉ khi có TT rời khỏi hệ thống

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối công việc

- Vấn đề lựa chọn công việc (điều phối)
  - Xét chương trình sau

```
PROGRAM PrintValue:
```

```
BEGIN
```

```
    Input A;
```

IO

```
    Input B;
```

```
    C = A + B;
```

CPU

```
    D = A - B;
```

```
    Print "The sum of inputs is: ", C;
```

IO

```
    Print "The Difference of inputs is: ", D;
```

```
END.
```

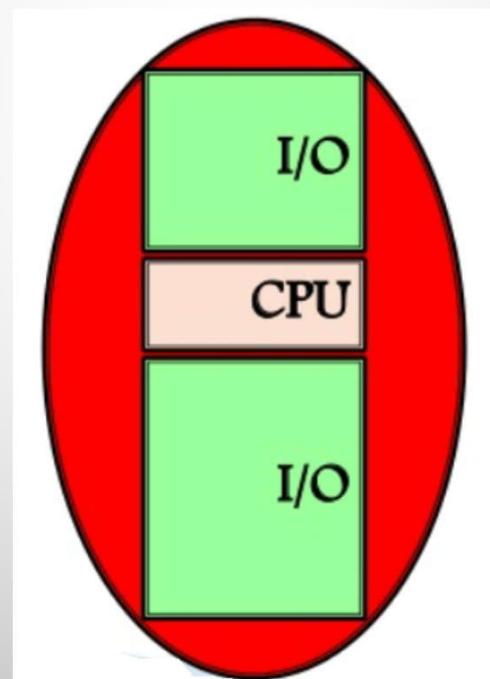
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối công việc

- Vấn đề lựa chọn công việc (điều phối)
  - TT **thiên về vào/ra**: ít thời gian CPU
  - VD: Các chương trình đồ họa



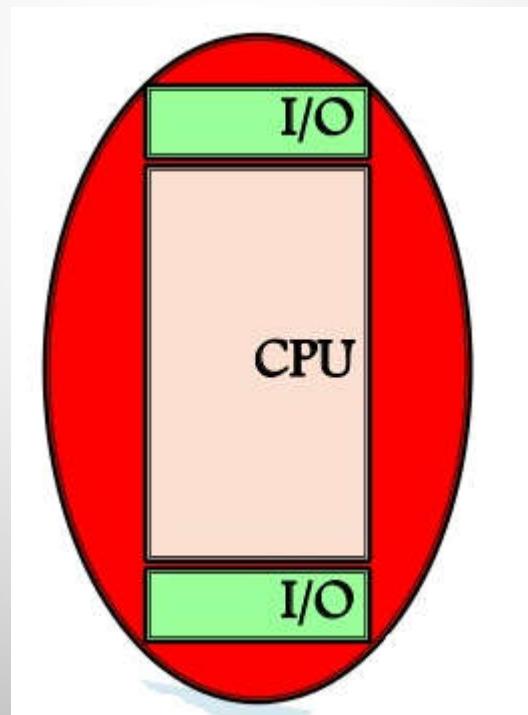
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối công việc

- Vấn đề lựa chọn công việc (điều phối)
  - TT **thiên về tính toán**: nhiều thời gian CPU
  - VD: Các chương trình toán học...



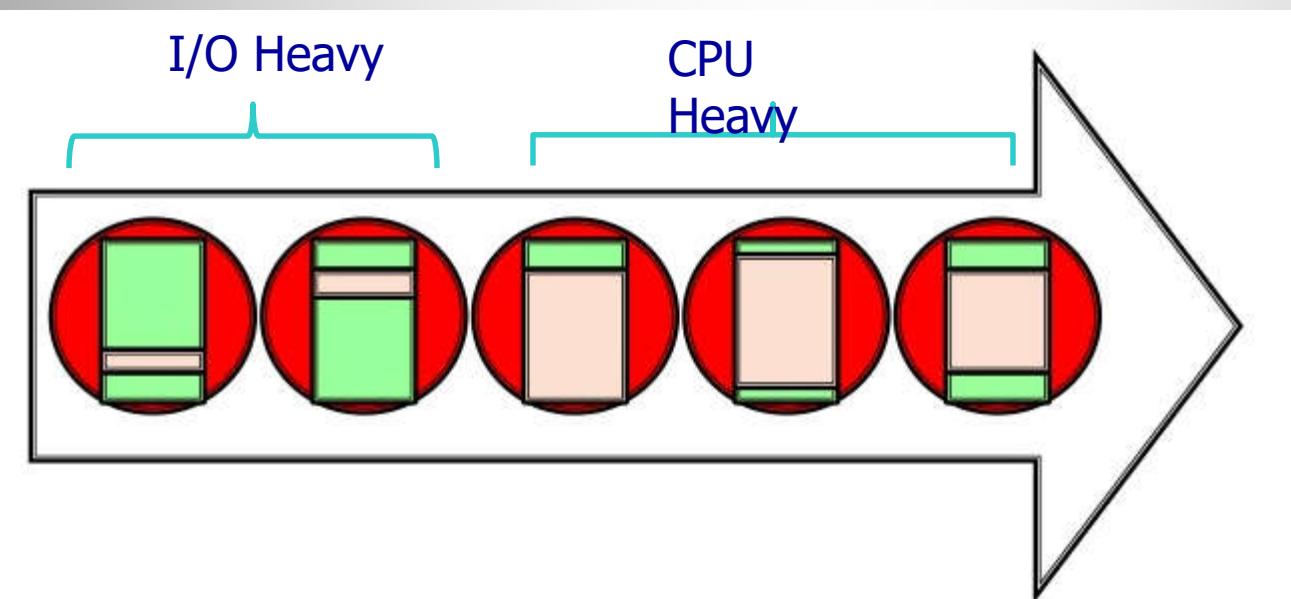
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối công việc

- Vấn đề lựa chọn công việc (điều phối)
  - Cần lựa chọn lنى cả 2 loại TT
    - ⇒ tt vào ra: hàng đợi sẵn sàng rỗng, lãng phí CPU
    - ⇒ tt tính toán: hàng đợi thiết bị rỗng, lãng phí thiết bị
- Nếu đầu vào là các Job như sau



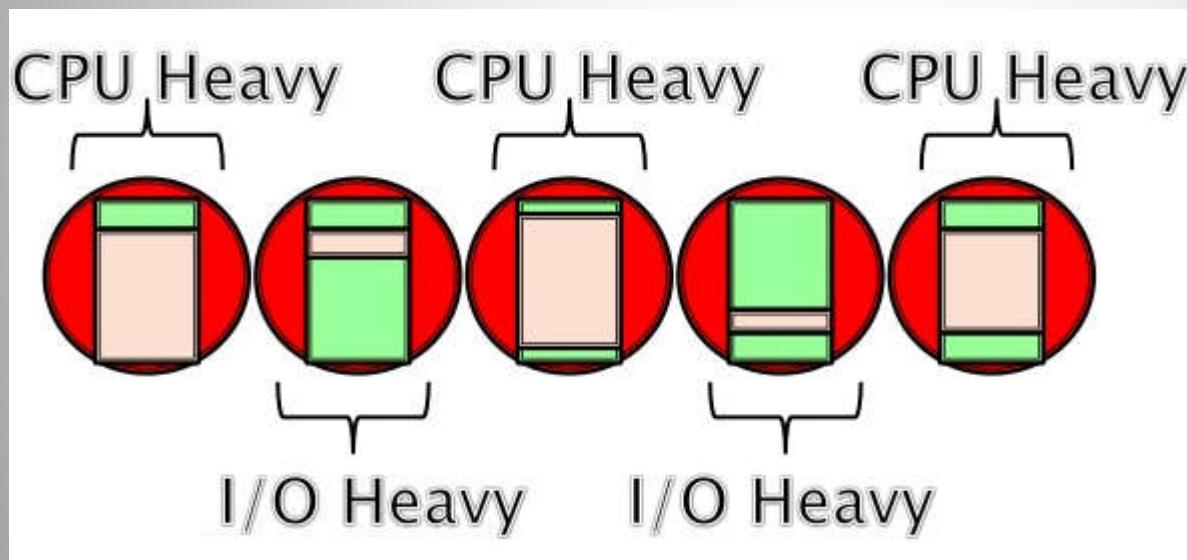
## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối công việc

- Job scheduler sẽ sắp xếp lại vị trí các Job rồi gửi sang Process Scheduler



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Điều phối CPU

- Lựa chọn 1 TT từ hàng đợi các TT đang sẵn sàng thực hiện và phân phối CPU cho nó
- Được thực hiện thường xuyên (VD: 100ms/lần)
  - TT thực hiện vài ms rồi thực hiện vào/ra
  - Lựa chọn TT mới, đang sẵn sàng
- Phải thực hiện nhanh
  - 10ms để quyết định  $\Rightarrow 10/(110) = 9\%$  thời gian CPU lãng phí
- Vấn đề luân chuyển CPU từ TT này -> TT khác
  - Phải lưu t/thái của TT cũ (PCB) và khôi phục t/thái cho TT mới
  - Thời gian luân chuyển là lãng phí
  - Có thể được hỗ trợ bởi phần cứng
- Vấn đề lựa chọn TT (điều phối CPU)

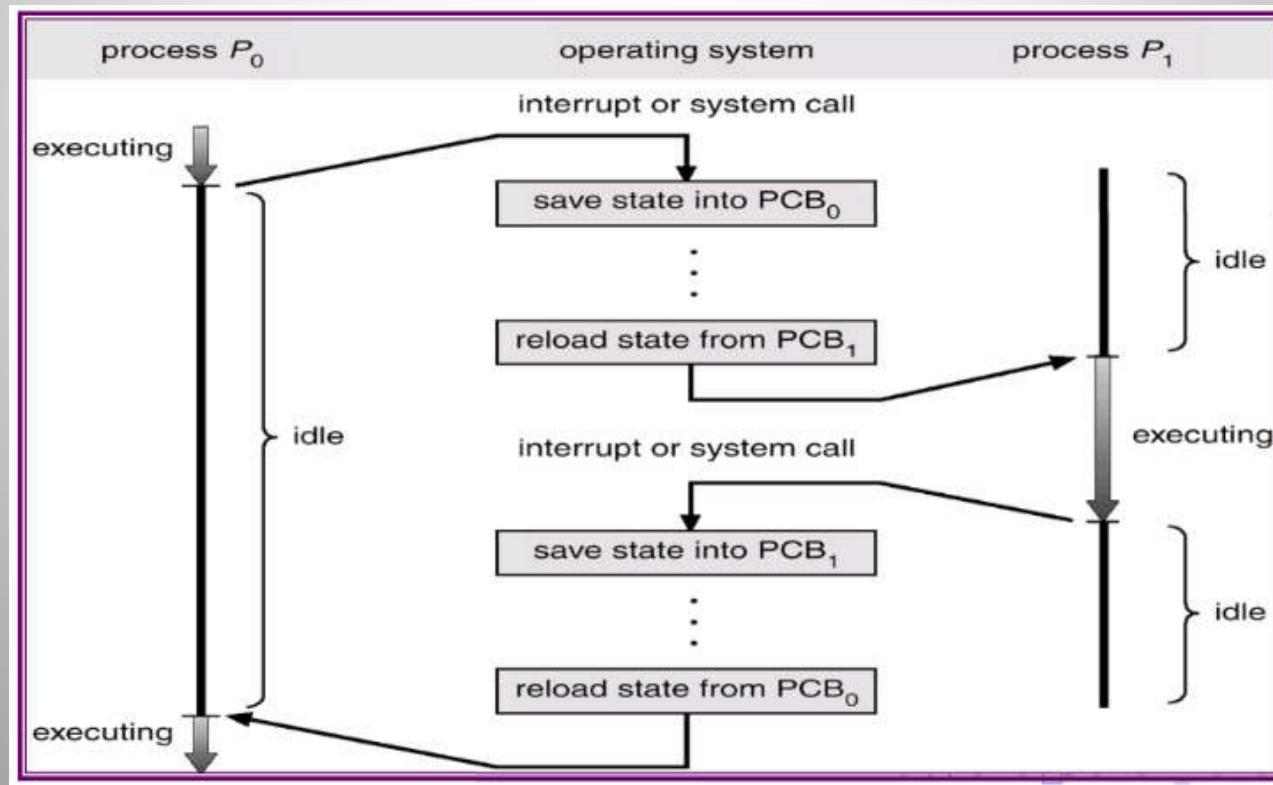
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Chuyển ngữ cảnh (context switch)

- Chuyển CPU từ TT này -> TT khác (hoán đổi TT thực hiện)
- Thực hiện khi xuất hiện tín hiệu ngắt (ngắt thời gian) hoặc TT đưa ra lời gọi hệ thống (thực hiện vào/ra)
- Lưu đồ của chuyển CPU giữa các TT(Silberschatz 2002)

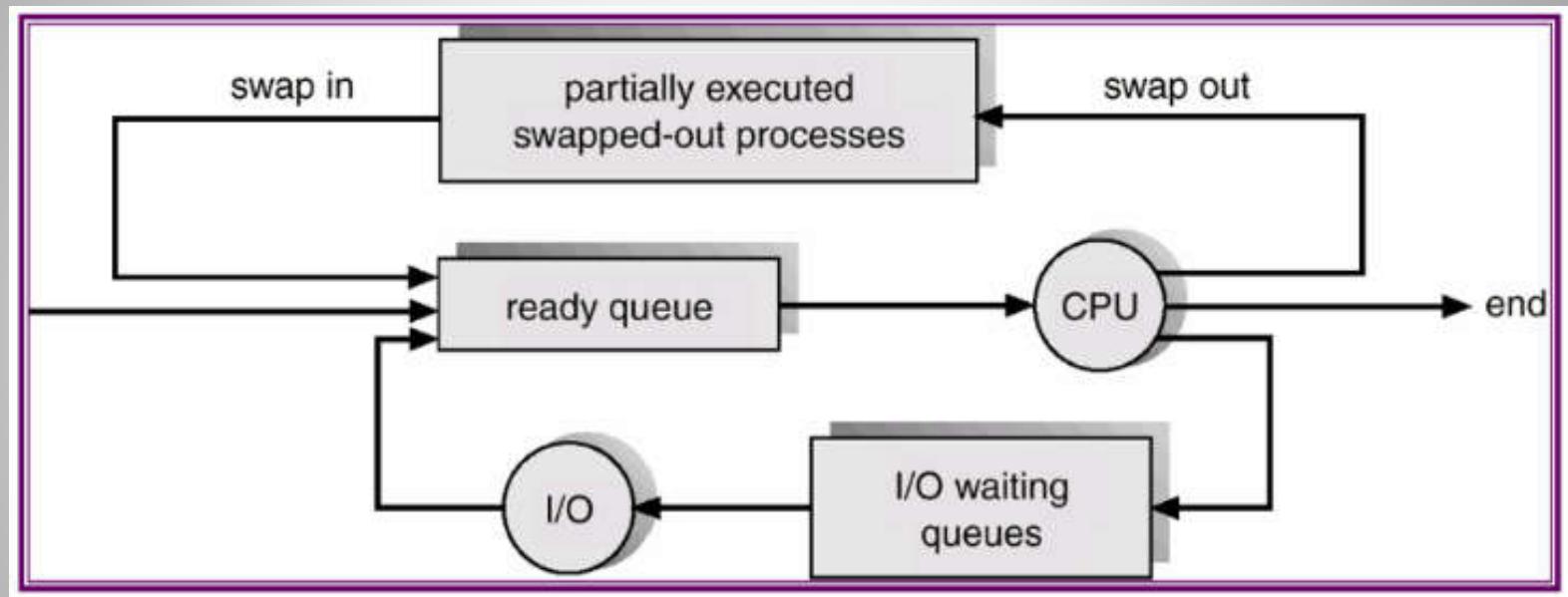


## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.2. Điều phối tiến trình

## Hoán chuyển tiến trình (Medium-term scheduler)



- Nhiệm vụ
  - Đưa TT ra khỏi bộ nhớ (làm giảm mức độ đa chương trình)
  - Sau đó đưa TT quay trở lại (có thể ở vị trí khác) và tiếp tục thực hiện
- Mục đích: Giải phóng vùng nhớ, tạo vùng nhớ tự do rộng hơn

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

- Khái niệm tiến trình
- Điều phối tiến trình (Process Scheduling)
- **Thao tác trên tiến trình**
- Hợp tác tiến trình
- Truyền thông liên tiến trình

Chương 2 Quản lý tiến trình

1. Tiến trình

1.3. Thao tác trên tiến trình

Thao tác trên tiến trình

● Tạo tiến trình

● Kết thúc tiến trình

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

### Tạo tiến trình

- TT có thể tạo nhiều TT mới cùng hoạt động  
(CreateProcess(), fork())
  - TT tạo: TT cha
  - TT được tạo: TT con
- TT con có thể tạo TT con khác ⇒ Cây tiến trình

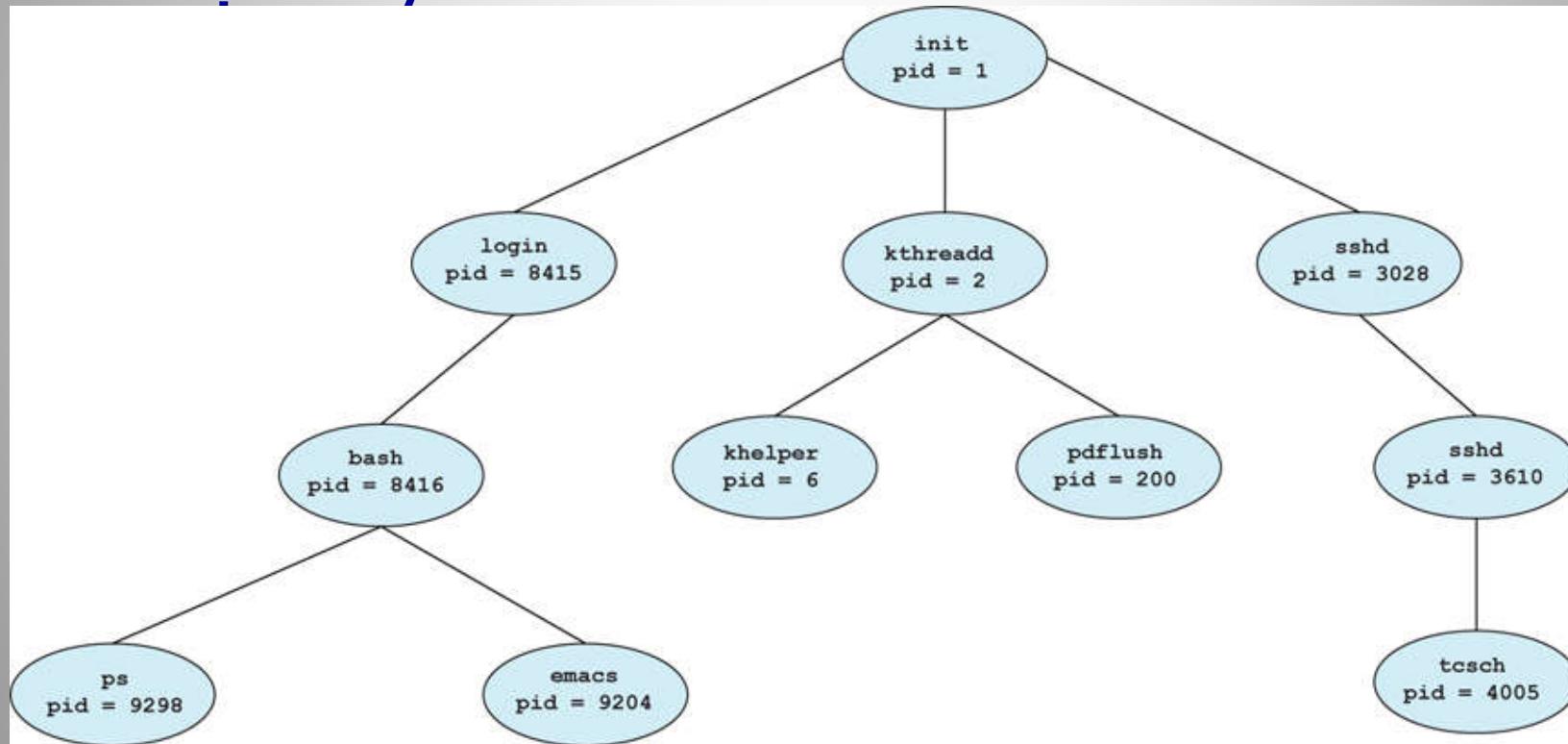
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

## Cây tiến trình

### ● Ví dụ: Cây tiến trình trên HĐH Solaris



## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

### Tạo tiến trình

#### ● Vấn đề phân phối tài nguyên

- TT con lấy từ HĐH
- TT con lấy từ TT cha
  - Tất cả
  - Một phần của TT cha (ngăn ngừa việc tạo quá nhiều TT con)

#### ● Vấn đề thực hiện

- TT cha tiếp tục thực hiện đồng thời với TT con
- TT cha đợi TT con kết thúc

## Kết thúc tiến trình

### ● Hoàn thành câu lệnh cuối và yêu cầu HĐH

xóa nó (exit)

- Gửi trả dữ liệu tới TT cha
- Các tài nguyên đã cung cấp được trả lại hệ thống

### ● TT cha có thể kết thúc sự thực hiện của TT con

- TT cha phải biết định danh TT con  $\Rightarrow$  TT con phải gửi định danh cho TT cha khi được khởi tạo

## Kết thúc tiến trình (tiếp.)

### ● TT cha kết thúc TT con khi

- TT con sử dụng **vượt quá mức tài nguyên** được cấp
- Nhiệm vụ cung cấp cho TT con **không** còn **cần** thiết nữa
- TT **cha** kết thúc và **HĐH** không cho phép TT con tồn tại khi

TT cha kết thúc

⇒**Cascading termination**. VD: *kết thúc hệ thống*

## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

# Một số hàm với tiến trình trong WIN32 API

Library  
DLL

Kernel32.lib  
Kernel32.dll

## ● CreateProcess( . . . )

- LPCTSTR Tên của chương trình được thực hiện
- LPTSTR Tham số dòng lệnh
- LPSECURITY\_ATTRIBUTES Thuộc tính an ninh t/trình
- LPSECURITY\_ATTRIBUTES Thuộc tính an ninh luồng
- BOOL Cho phép kế thừa các thẻ thiết bị (TRUE/FALSE)
- DWORD Cờ tạo tiến trình (VD CREATE\_NEW\_CONSOLE)
- LPVOID Trỏ tới khối môi trường
- LPCTSTR Đường dẫn đầy đủ đến chương trình
- LPSTARTUPINFO Cấu trúc thông tin cho tiến trình mới
- LPPROCESS\_INFORMATION Thông tin về tiến trình mới

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.3. Thao tác trên tiến trình

### Một số hàm với tiến trình trong WIN32 API (tiếp)

#### ● *TerminateProcess(HANDLE hProcess, UINT uExitCode)*

- *hProcess* Thủ tiến trình bị kết thúc đóng
- *uExitCode* Mã kết thúc tiến trình

#### ● *WaitForSingleObject(HANDLE hHandle, DWORD dwMs)*

- *hHandle* Thủ đối tượng
- *dwMs* Thời gian chờ đợi (INFINITE)

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

- Khái niệm tiến trình
- Điều phối tiến trình (Process Scheduling)
- Thảo tác trên tiến trình
- **Hợp tác tiến trình**
- Truyền thông liên tiến trình

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

### Phân loại tiến trình

#### ● Các TT tuần tự

- Điểm bắt đầu của TT này nằm sau điểm kết thúc của TT kia

#### ● Các TT song song

- Điểm bắt đầu của TT này nằm giữa điểm bắt đầu và kết thúc của TT kia
- **Độc lập:** Không ảnh hưởng tới hoặc bị ảnh hưởng bởi TT khác đang thực hiện trong hệ thống
- **Có hợp tác:**Ảnh hưởng tới hoặc chịu ảnh hưởng bởi TT khác đang thực hiện trong hệ thống

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

## Hợp tác tiến trình

### ● Mục đích

- Chia sẻ thông tin
- Tăng tốc độ tính toán
- Module hóa
- Tiện dụng

### ● Đòi hỏi cơ chế cho phép

- Truyền thông giữa các TT
- Đồng bộ hóa hoạt động của các TT

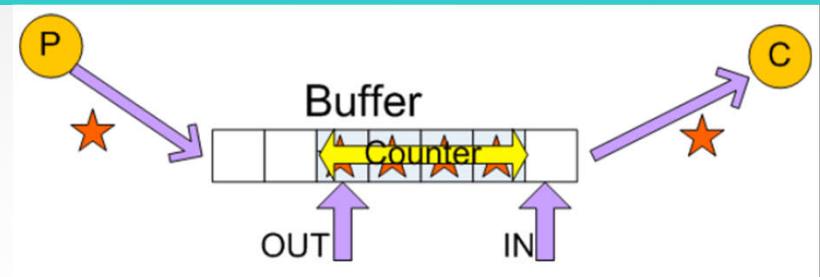
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) I

- Hệ thống gồm 2 tiến trình
  - Producer sản xuất ra các sản phẩm
  - Consumer tiêu thụ các sản phẩm được sản xuất ra
- Ứng dụng
  - Chương trình in (producer) sản xuất ra các ký tự được tiêu thụ bởi bộ điều khiển máy in (consumer)
  - Trình dịch (producer) sản xuất ra mã hợp ngữ, trình hợp ngữ (consumer/producer) tiêu thụ mã hợp ngữ rồi sản xuất ra module đối tượng được bộ thực hiện (consumer) tiêu thụ



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

- *Producer* và *Consumer* hoạt động đồng thời  
Sử dụng vùng đệm dùng chung (**Buffer**) chứa sản phẩm được điền vào bởi *Producer* và được lấy ra bởi *Consumer*
  - IN Vị trí trống kế tiếp trong vùng đệm;
  - OUT Vị trí đầy đầu tiên trong vùng đệm.
  - Counter Số sản phẩm trong vùng đệm
- Producer và Consumer phải đồng bộ
  - Consumer không cố gắng tiêu thụ sản phẩm chưa được sản xuất

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) III

- Vùng đệm dung lượng vô hạn
  - Khi Buffer rỗng, *Consumer* phải đợi
  - *Producer* không phải đợi khi đặt sản phẩm vào buffer
- Vùng đệm dung lượng hữu hạn
  - Khi Buffer rỗng, *Consumer* phải đợi
  - *Producer* phải đợi nếu vùng đệm đầy

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.4. Hợp tác tiến trình

## Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

### *Producer*

```
while(1) {  
    /*produce an item in nextProduced*/  
    while (Counter == BUFFER_SIZE) ; /*do nothing*/  
    Buffer[IN] = nextProduced;  
    IN = (IN + 1) % BUFFER_SIZE;  
    Counter++;  
}
```

### *Consumer*

```
while(1){  
    while(Counter == 0) ; /*do nothing*/  
    nextConsumed = Buffer[OUT];  
    OUT =(OUT + 1) % BUFFER_SIZE;  
    Counter--; /*consume the item in nextConsumed*/  
}
```

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

- Khái niệm tiến trình
- Điều phối tiến trình (Process Scheduling)
- Thảo tác trên tiến trình
- Hợp tác tiến trình
- **Truyền thông liên tiến trình**

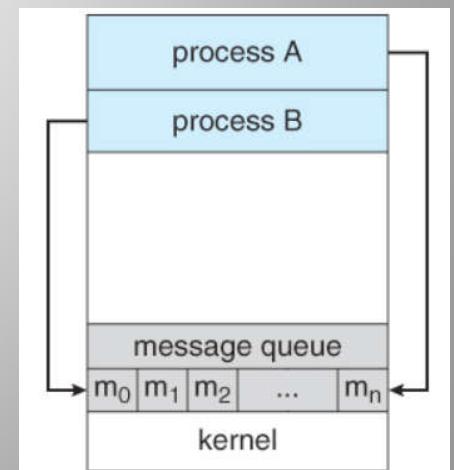
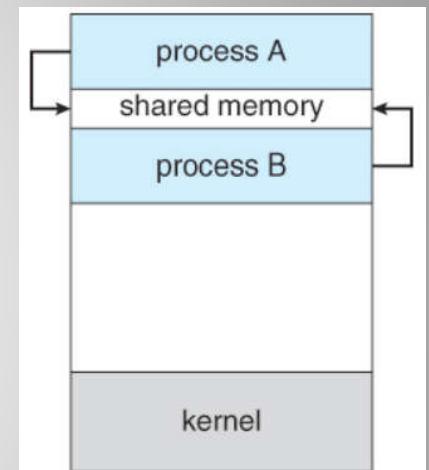
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Trao đổi giữa các tiến trình

- Mô hình bộ nhớ dùng chung
  - Các TT **chia sẻ vùng nhớ chính**
  - Mã cài đặt được viết tường minh bởi người lập trình ứng dụng
  - Ví dụ: Bài toán Producer-Consumer
- Mô hình truyền thông liên TT (Interprocess communication)
  - Là cơ chế cho phép các TT truyền thông và đồng bộ các hoạt động
  - Thường được sử dụng trong các hệ phân tán khi các TT truyền thông nằm trên các máy khác nhau (chat)
  - Đảm bảo bởi **hệ thống truyền thông điệp**



## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Hệ thống truyền thông điệp

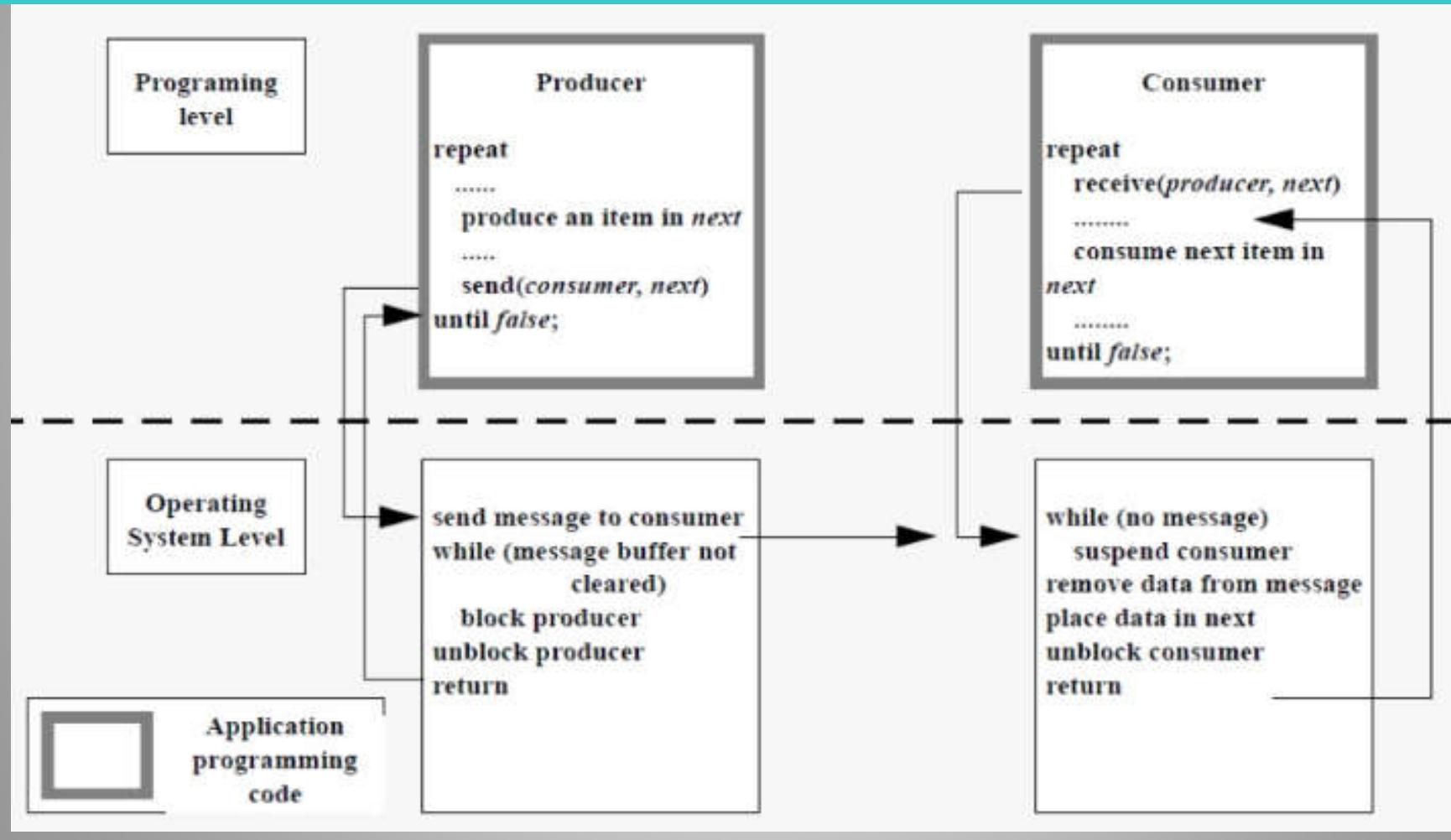
- Cho phép các TT trao đổi với nhau không qua sử dụng các biến dùng chung
- Yêu cầu 2 thao tác cơ bản
  - Send (msg) Các msg có kích thước cố định hoặc thay đổi
    - Cố định : dễ cài đặt mức hệ thống, nhiệm vụ lập trình khó
    - Thay đổi: cài đặt mức hệ thống phức tạp, lập trình đơn giản
  - Receive (msg)
- Nếu 2 TT P và Q muốn trao đổi, chúng cần
  - Thiết lập 1 liên kết truyền thông (vật lý/logic) giữa chúng
  - Trao đổi các messages nhờ các thao tác send/receive

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

## Hệ thống truyền thông địệp và bài toán Producer consumer



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Hệ thống truyền thông điệp

#### ● Các vấn đề cài đặt

- Các liên kết được thiết lập như thế nào?
- Một liên kết có thể dùng cho nhiều hơn 2 TT?
- Bao nhiêu liên kết có thể tồn tại giữa mọi cặp TT?
- Kích thước thông báo mà liên kết chấp nhận cố định/thay đổi?
- Liên kết một hay hai chiều?

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Truyền thông trực tiếp

- Các TT phải gọi tên TT nhận/gửi một cách tường minh
  - send (P, message) - gửi 1 thông báo tới TT P
  - receive(Q, message) - nhận 1 thông báo từ TT Q
- Tính chất của liên kết truyền thông
  - Được thiết lập tự động
  - 1 liên kết gắn chỉ với cặp TT truyền thông
  - Chỉ tồn tại 1 liên kết giữa cặp TT
  - Liên kết có thể là 1 chiều, nhưng thường 2 chiều

## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Truyền thông gián tiếp

- Các **thông điệp** được gửi/nhận tới/từ các **hòm thư** (mailboxes), **cổng** (ports)
  - Mỗi hòm thư có **định danh duy nhất**
  - Các TT có thể trao đổi nếu chúng dùng chung hòm thư
- **Tính chất** các liên kết
  - Được thiết lập chỉ khi các TT dùng chung hòm thư
  - 1 liên kết **có thể** được gắn với **nhiều** TT
  - Mỗi cặp TT có thể dùng chung **nhiều** liên kết truyền thông
  - Liên kết có thể 1 hay 2 chiều
- **Các thao tác**
  - **Tạo** hòm thư
  - **Gửi/nhận** thông báo qua hòm thư
    - send(A, msg): Gửi 1 msg tới hòm thư A
    - receive(A, msg): Nhận 1 msg từ hòm thư A
  - **Hủy bỏ** hòm thư



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Vấn đề đồng bộ hóa

- Truyền thông điệp có thể phải chờ đợi (blocking), hoặc không chờ đợi (non blocking)
  - Blocking Truyền thông đồng bộ
  - Non-blocking Truyền thông không đồng bộ
- Các thủ tục send() và receive() có thể phải chờ đợi hoặc không
  - Blocking send TT gửi thông báo và đợi cho tới khi msg được nhận bởi TT nhận hoặc bởi hòm thư
  - Non-blocking send TT gửi thông báo và tiếp tục làm việc
  - Blocking receive TT nhận phải đợi cho tới khi có thông điệp
  - Non-blocking receive TT nhận trả về hoặc 1 thông điệp có giá trị, hoặc 1 giá trị null

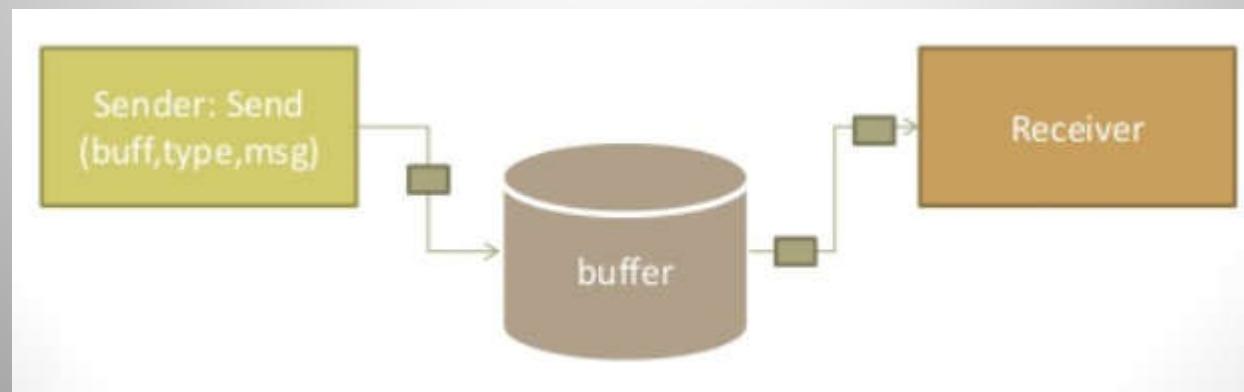
## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

#### Vùng đệm

- Các thông điệp trao đổi giữa các TT  
được lưu trong hàng đợi tạm thời  
(buffer)



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

##### Vùng đệm

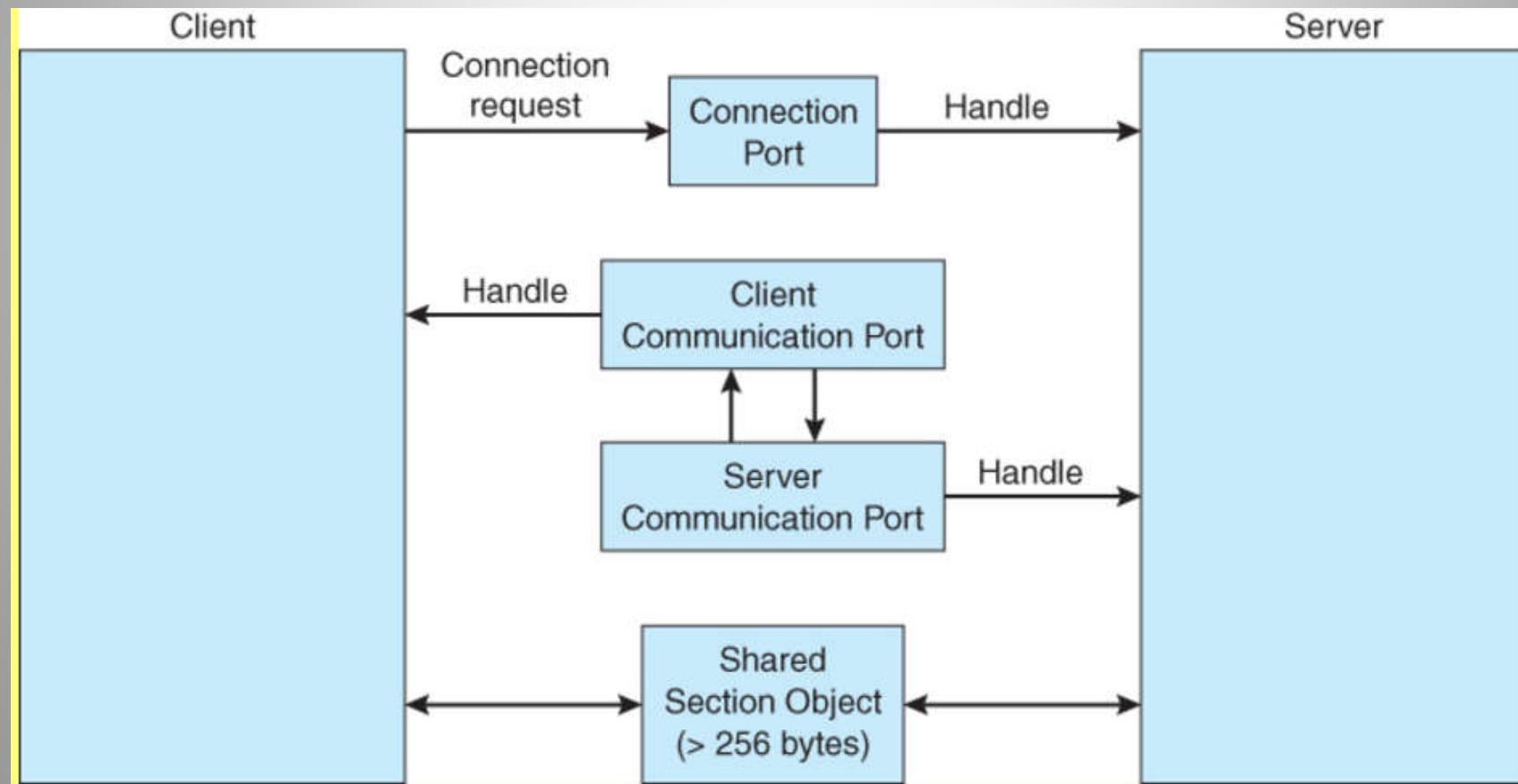
- Hàng đợi có thể được cài đặt theo **khả năng chứa**
  - 0 (Zero capacity): Độ dài hàng đợi là 0
    - Không tồn tại thông điệp trong đường liên kết  $\Rightarrow$  Sender phải đợi cho tới khi thông điệp được nhận
  - Có giới hạn(Bound capacity)
    - Hàng đợi có độ dài  $n \Rightarrow$  chứa nhiều nhất  $n$  thông điệp
    - Nếu hàng đợi không đầy, thông điệp sẽ được lưu vào vùng đệm và Sender tiếp tục bình thường
    - Nếu hàng đợi đầy, sender phải đợi cho tới khi có chỗ trống
  - Không giới hạn (Unbound capacity)
    - Sender không bao giờ phải đợi

## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Windows XP Truyền thông địệp giữa các tiến trình



## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Tổng quan về Truyền thông trong hệ thống Client-Server với Socket

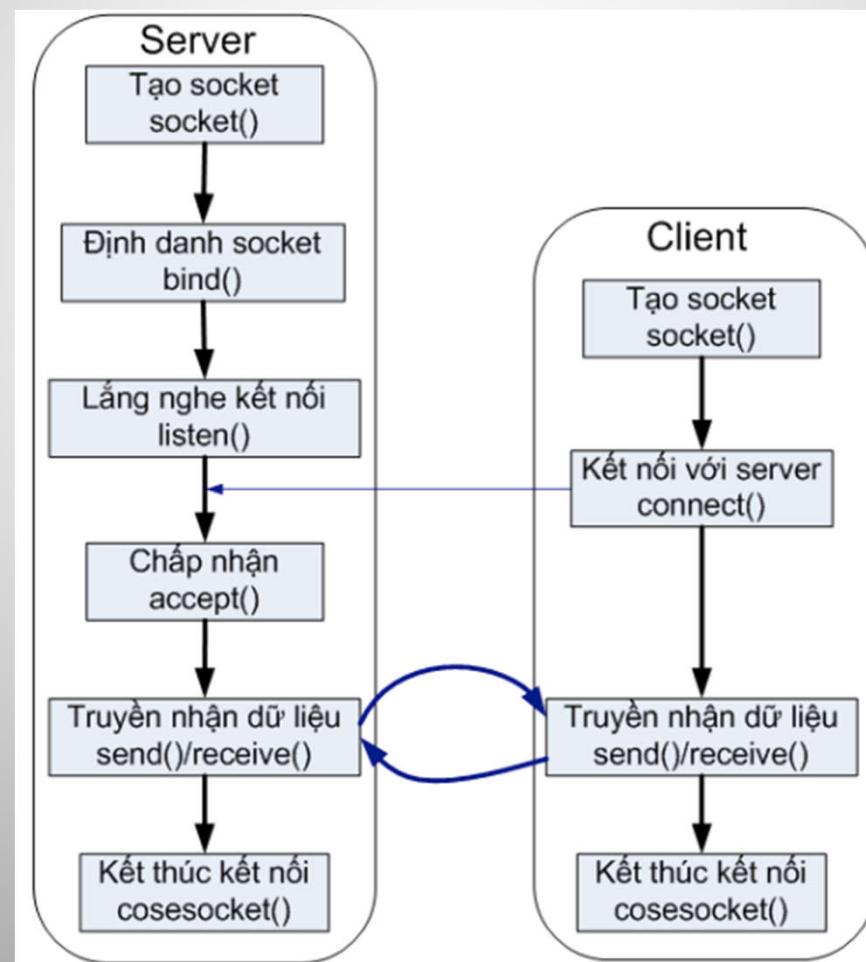
- Được xem như **đầu mút cho truyền thông**, qua đó các ứng dụng gửi/nhận dữ liệu qua mạng
  - Truyền thông thực hiện giữa các **cặp Sockets**
- Bao gồm cặp **địa chỉ IP và cổng**. Ví dụ: 161.25.19.8:1625
  - Địa chỉ IP: Địa chỉ của máy trong mạng
  - Cổng (port): Định danh TT tham gia trao đổi trên máy
- Các loại sockets
  - Stream Socket: Dựa trên **TCP/IP** → Truyền dữ liệu tin cậy
  - Datagram Socket: Dựa trên **UDP/IP** → Truyền dữ liệu không tin cậy
- Win32 API: Winsock
  - Windows Sockets Application Programming Interface

## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Thiết lập quá trình trao đổi dữ liệu



## Chương 2 Quản lý tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

### Một số hàm trong Winsock API 32

socket() Tạo socket truyền dữ liệu

bind() Định danh cho socket vừa tạo (gán cho một cổng)

listen() Lắng nghe một kết nối

accept() Chấp nhận một kết nối

connect() kết nối với server.

send() Gửi dữ liệu với stream socket.

sendto() Gửi dữ liệu với datagram socket.

receive() Nhận dữ liệu với stream socket.

recvfrom() Nhận dữ liệu với datagram socket.

closesocket() Kết thúc một socket đã tồn tại.

.....

## Chương 2 Quản lí tiến trình

### 1. Tiến trình

#### 1.5. Truyền thông liên tiến trình

## Bài tập

Sử dụng Winsock xây dựng chương trình Client-Server

Chương trình Chat.

.....

## Chương 2 Quản lí tiến trình

- ① Tiến trình
- ② Luồng (Thread)
- ③ Điều phối CPU
- ④ Tài nguyên găng và điều độ tiến trình
- ⑤ Bẽ tắc và xử lý bẽ tắc

Chương 2 Quản lý tiến trình

2. Luồng

2.1. Giới thiệu

- Giới thiệu

- Mô hình đa luồng

- Cài đặt luồng với Windows

- Vấn đề đa luồng

## Chương 2 Quản lý tiến trình

### 2. Luồng

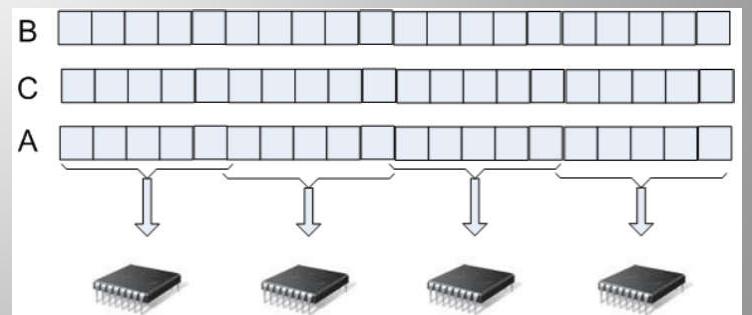
#### 2.1. Giới thiệu

Ví dụ: Tính toán trên vector

## ● Tính toán trên vector kích thước lớn

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

Với hệ thống nhiều vi xử lý

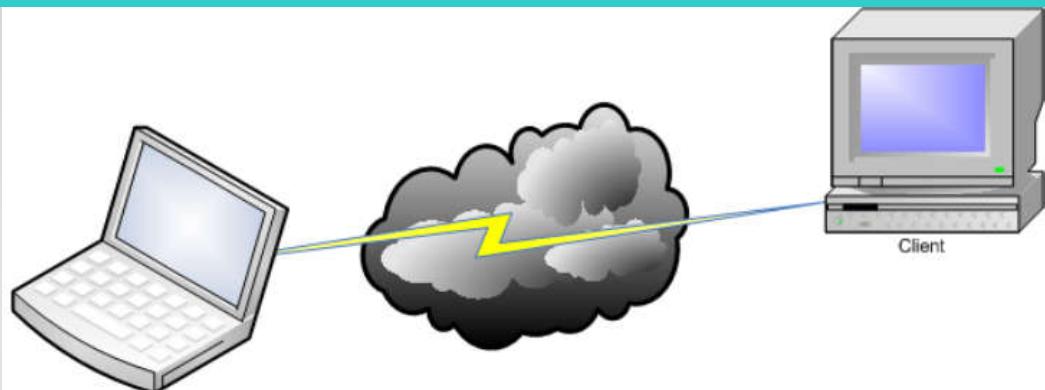


## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

#### Ví dụ: Chat



##### Tiến trình P

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

##### Vấn đề nhận Msg

- Blocking Recieve
- Non-blocking Receive

Giải quyết  
Thực hiện song song  
Receive & Send

##### Tiến trình Q

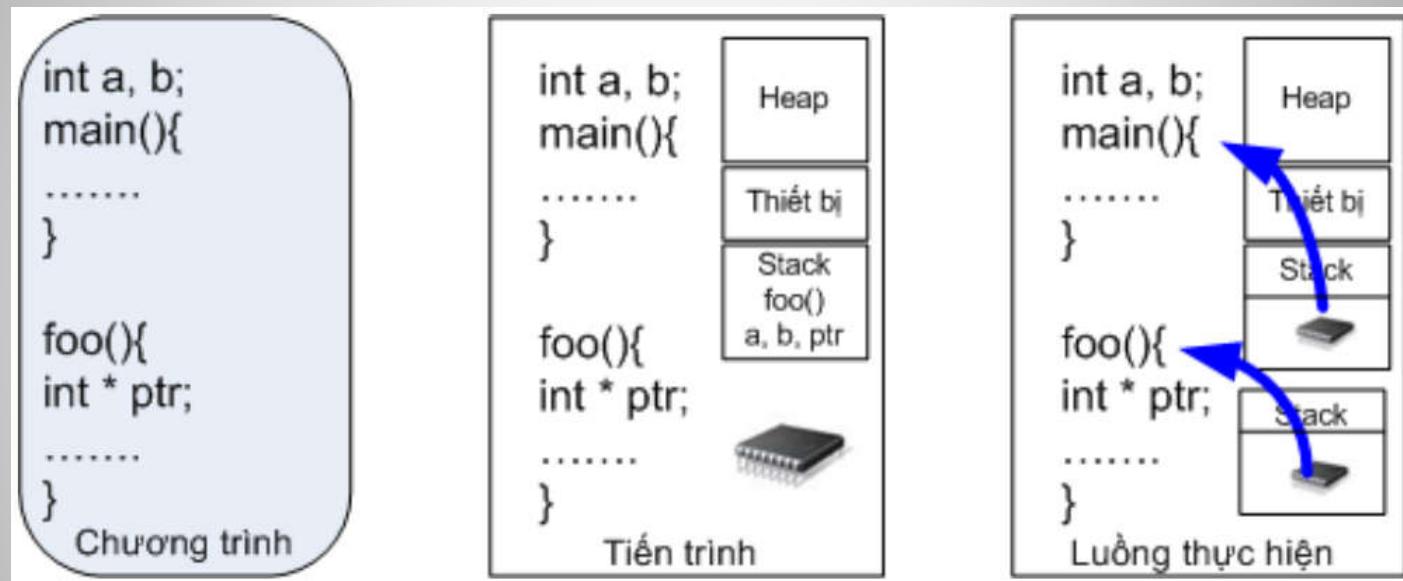
```
While (1) {  
  
    Receive(P,Msg);  
    PrintLine(Msg);  
    ReadLine(Msg);  
    Send(P,Msg);  
}
```

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

## Chương trình - Tiến trình - Luồng



- Chương trình: Dãy lệnh, các biến,..
- Tiến trình: Chương trình đang thực hiện: Stack, t/bị, VXL,..
- Luồng: C/trình đang thực hiện trong ngữ cảnh tiến trình
  - Nhiều processor → Nhiều luồng, mỗi luồng trên một VXL
    - Khác nhau về giá trị các thanh ghi, nội dung stack

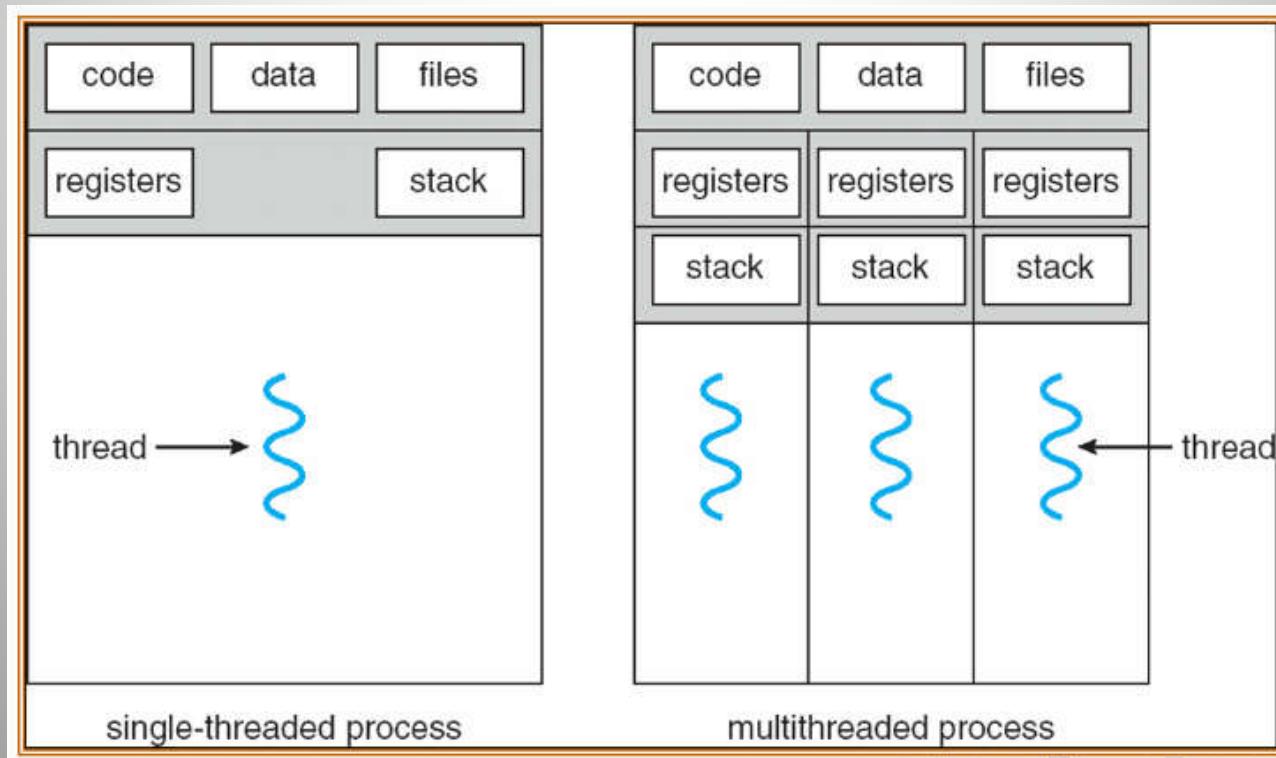
## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

## Tiến trình đơn luồng và đa luồng

- HĐH truyền thống (MS-DOS, UNIX)
  - TT có 1 luồng điều khiển (heavyweight process)
- HĐH hiện nay (Windows, Linux)
  - TT có thể gồm nhiều luồng
  - Có thể thực hiện nhiều nhiệm vụ tại 1 thời điểm

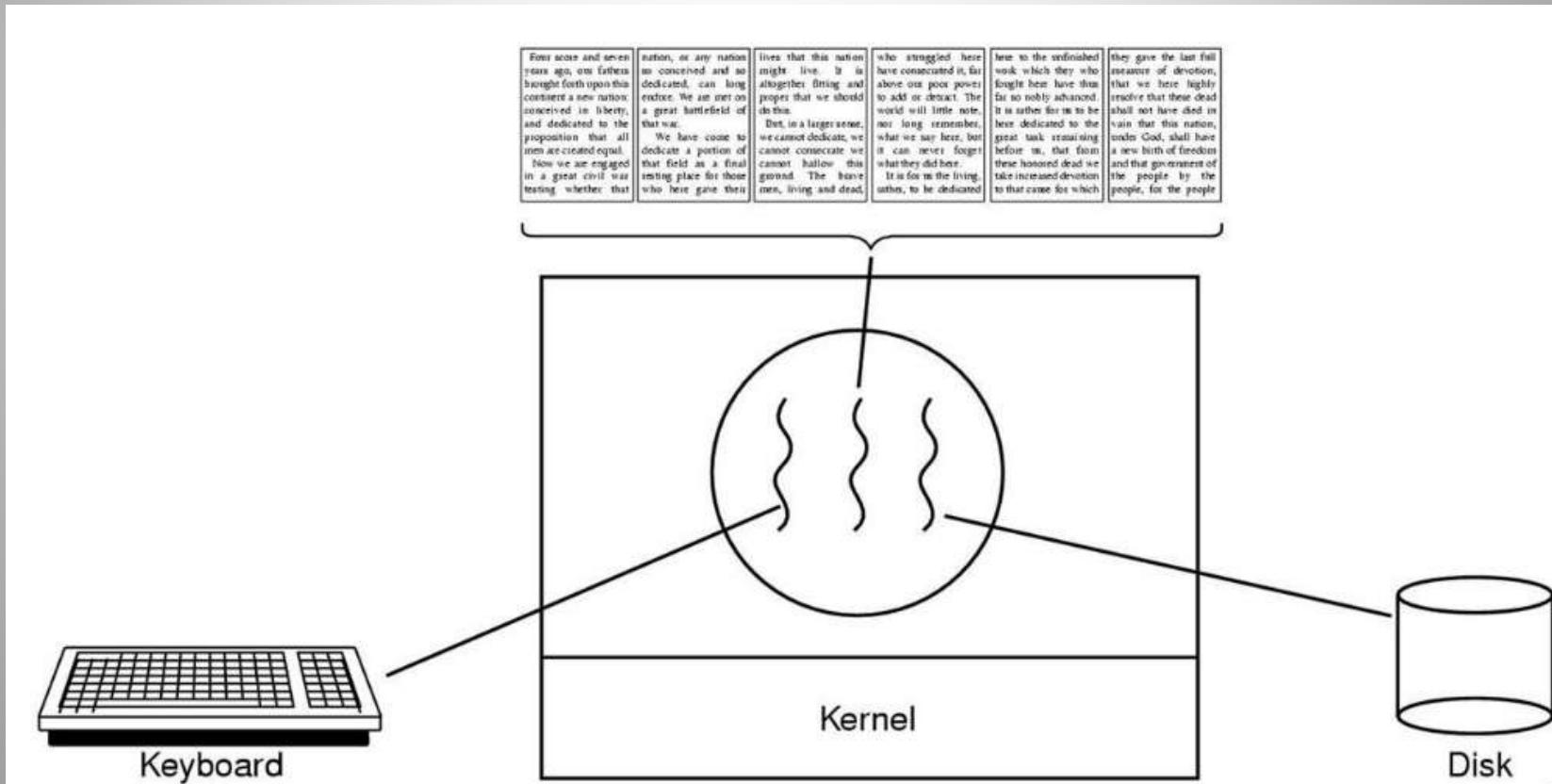


## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

Ví dụ: Word processor (Tanenbaum 2001)



## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

#### Khái niệm luồng

- Là đơn vị sử dụng CPU cơ bản, gồm
  - Định danh luồng (ID Thread)
  - Bộ đếm chương trình (Program Counter)
  - Tập các thanh ghi (Registers)
  - Không gian stack
- Chia sẻ cùng **các luồng khác** trong cùng 1 TT
  - Đoạn mã lệnh
  - Đoạn dữ liệu (đối tượng toàn cục)
  - Các tài nguyên HĐH khác (file đang mở)
- Các luồng có thể thực hiện **cùng đoạn mã** với **ngữ cảnh** (*Tập thanh ghi, Bộ đếm chương trình, stack*) khác nhau
- Còn được gọi là TT nhẹ (LWP: Lightweight Process)
- 1 TT có ít nhất là 1 luồng

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Phân biệt tiến trình và luồng

#### Tiến trình

TT có đoạn mã/dữ liệu/heap & các đoạn khác

Phải có ít nhất 1 luồng trong mỗi TT

Các luồng trong phạm vi 1 TT chia sẻ mã/dữ liệu/heap, vào/ra nhưng có stack và tập thanh ghi riêng

Thao tác khởi tạo, luân chuyển TT tốn kém

Bảo vệ tốt do có không gian địa chỉ riêng

Khi TT kết thúc, các tài nguyên được đòi lại và các luồng phải kết thúc theo

#### Luồng

Luồng không có đoạn dữ liệu hay heap riêng

Luồng không đứng riêng mà nằm trong 1 TT

Có thể tồn tại nhiều luồng trong mỗi TT. Luồng đầu là luồng chính và sở hữu không gian stack của TT

Thao tác khởi tạo và luân chuyển luồng không tốn kém

Không gian địa chỉ chung, cần phải bảo vệ

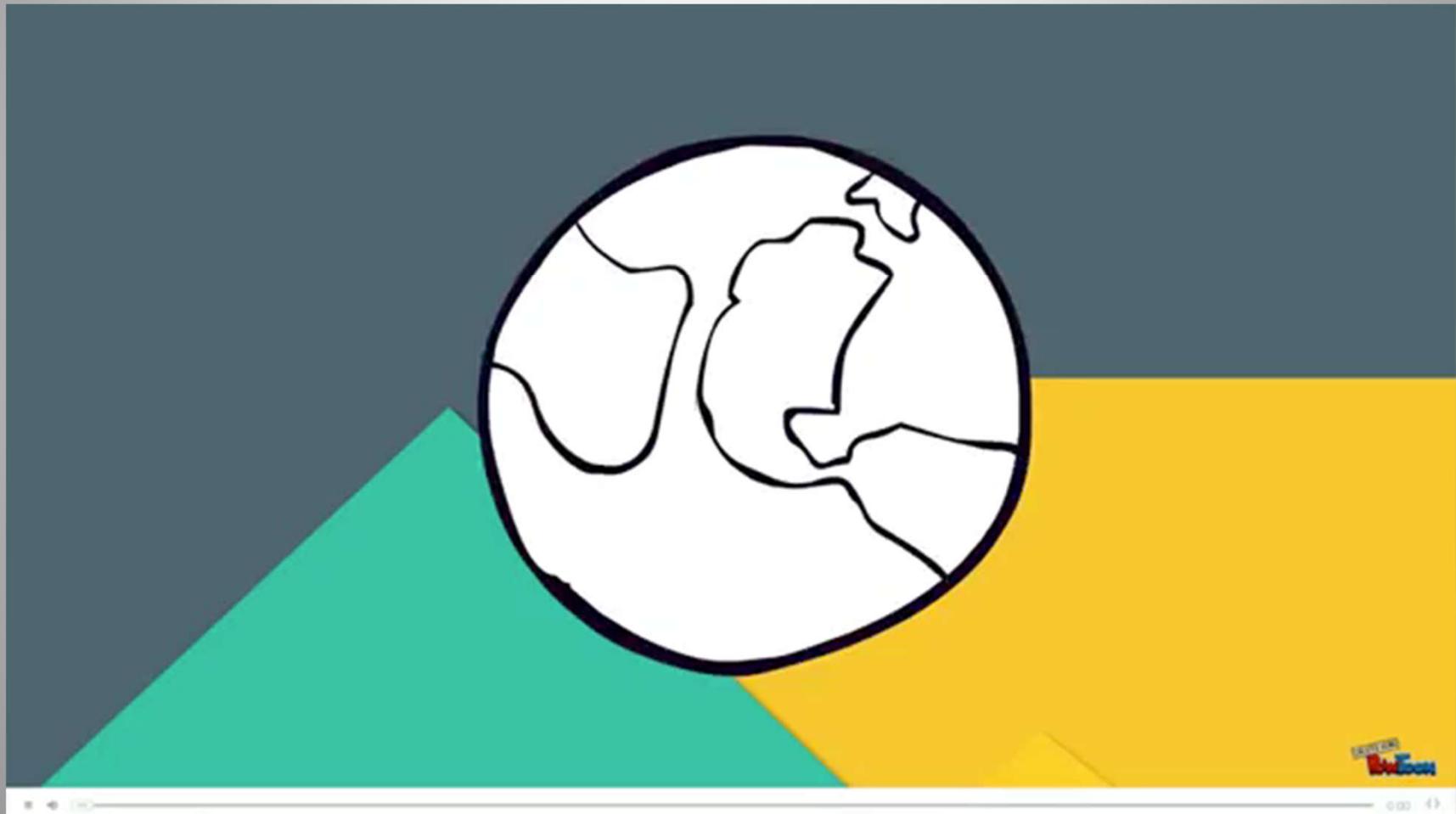
Luồng kết thúc, stack của nó được thu hồi

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

Phân biệt tiến trình và luồng



Chương 2 Quản lý tiến trình

2. Luồng

2.1. Giới thiệu

Lợi ích của lập trình đa luồng

- Tăng tính đáp ứng với người dùng
- Chia sẻ tài nguyên
- Tính kinh tế
- Sử dụng kiến trúc nhiều vi xử lý

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Lợi ích của lập trình đa luồng

- Tăng tính đáp ứng với người dùng

- Cho phép TT vẫn thực hiện ngay khi 1 phần đang chờ đợi (block) hoặc đang thực hiện tính toán tăng cường (lengthy operation)

- Chia sẻ tài nguyên

- Các luồng chia sẻ bộ nhớ và tài nguyên của TT chứa nó
  - Tốt cho các thuật toán song song (sử dụng chung các CTDL)
  - Trao đổi giữa các luồng thông qua bộ nhớ dùng chung
- Cho phép 1 ứng dụng chứa nhiều luồng hoạt động trong cùng không gian địa chỉ

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Lợi ích của lập trình đa luồng (tiếp)

- Tính kinh tế

- Các thao tác khởi tạo, hủy bỏ và luân chuyển luồng ít tốn kém
  - Minh họa được tính song song trên bộ đơn VXL do thời gian luân chuyển CPU nhanh (Thực tế chỉ 1 luồng thực hiện)

- Sử dụng kiến trúc nhiều vi xử lý

- Các luồng chạy song song thực sự trên các bộ VXL khác nhau.

## Chương 2 Quản lý tiến trình

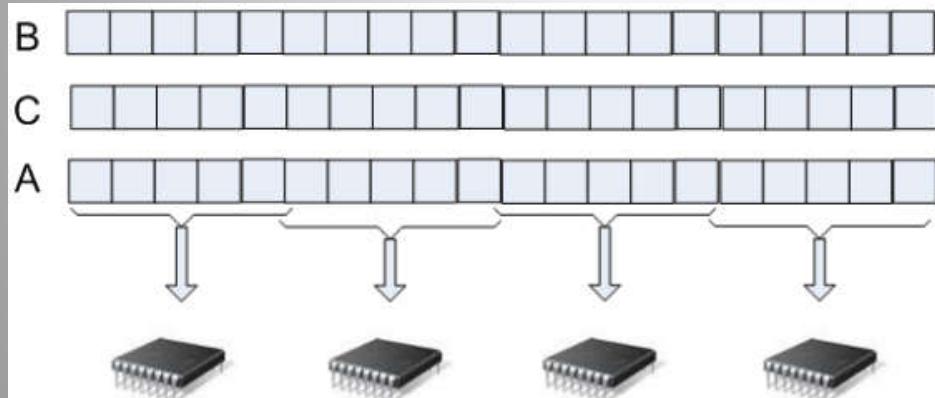
### 2. Luồng

#### 2.1. Giới thiệu

### Lợi ích của lập trình đa luồng -> Ví dụ

#### Tính toán trên vector

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



#### Mô hình đa luồng

```
void fn(a,b)  
for(k = a; k < b; k ++){  
    a[k] = b[k] * c[k];  
}  
void main(){  
    CreateThread(fn(0, n/4));  
    CreateThread(fn(n/4, n/2));  
    CreateThread(fn(n/2, 3n/4));  
    CreateThread(fn(3n/4, n));  
}
```

## Chương 2 Quản lý tiến trình

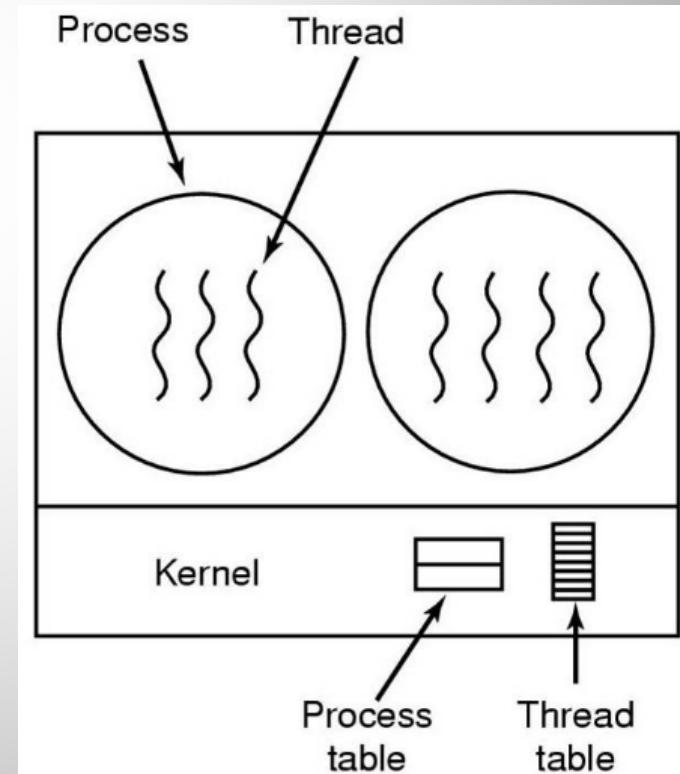
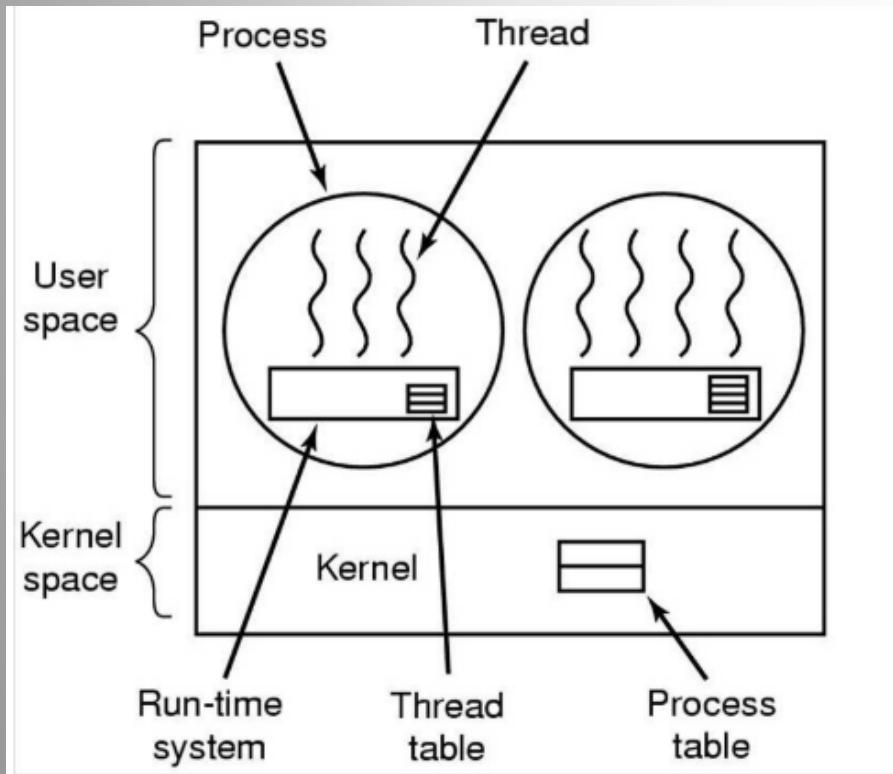
### 2. Luồng

#### 2.1. Giới thiệu

### Cài đặt luồng

Trong không gian người dùng

Trong không gian nhân



## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Luồng người dùng (User -Level Threads)

- Quản lý các luồng được thực hiện bởi chương trình ứng dụng
- Nhân hệ thống không biết gì về sự tồn tại luồng
  - Điều phối TT như 1 đơn vị duy nhất
  - Gán cho mỗi TT 1 t/ thái duy nhất
    - Sẵn sàng, chờ đợi, thực hiện,..
- Chương trình ứng dụng được lập trình theo mô hình đa luồng  
được hỗ trợ bởi thư viện luồng

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Luồng người dùng (User -Level Threads)

Thư viện: POSIX Pthreads, Mach C-threads, Solaris 2 UI-threads, Win32 threads

- **Ưu điểm**

- Nhanh chóng trong **tạo** và quản lý luồng

- **Nhược điểm**

- Khi 1 luồng rơi vào t/thái **chờ đợi**, tất cả các luồng trong cùng TT bị **chờ đợi theo**  
⇒ Không tận dụng được ưu điểm của mô hình lập trình đa luồng

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.1. Giới thiệu

### Luồng mức hệ thống (Kernel - Level threads)

- Nhân duy trì thông tin về TT và các luồng
- Quản lý luồng được thực hiện bởi nhân
  - Không tồn tại các mã quản lý luồng trong ứng dụng
  - Điều phối luồng được thực hiện bởi nhân, dựa trên các luồng
- Nhược điểm:
  - Chậm trong tạo và quản lý luồng
- Ưu điểm:
  - 1 luồng chờ đợi vào ra, không ảnh hưởng tới luồng khác
  - Trong môi trường đa VXL, nhân có thể điều phối các luồng cho các VXL khác nhau
- HĐH: Windows NT/2000/XP, Linux, OS/2,..

Chương 2 Quản lý tiến trình

2. Luồng

2.2. Mô hình đa luồng

● **Giới thiệu**

● **Mô hình đa luồng**

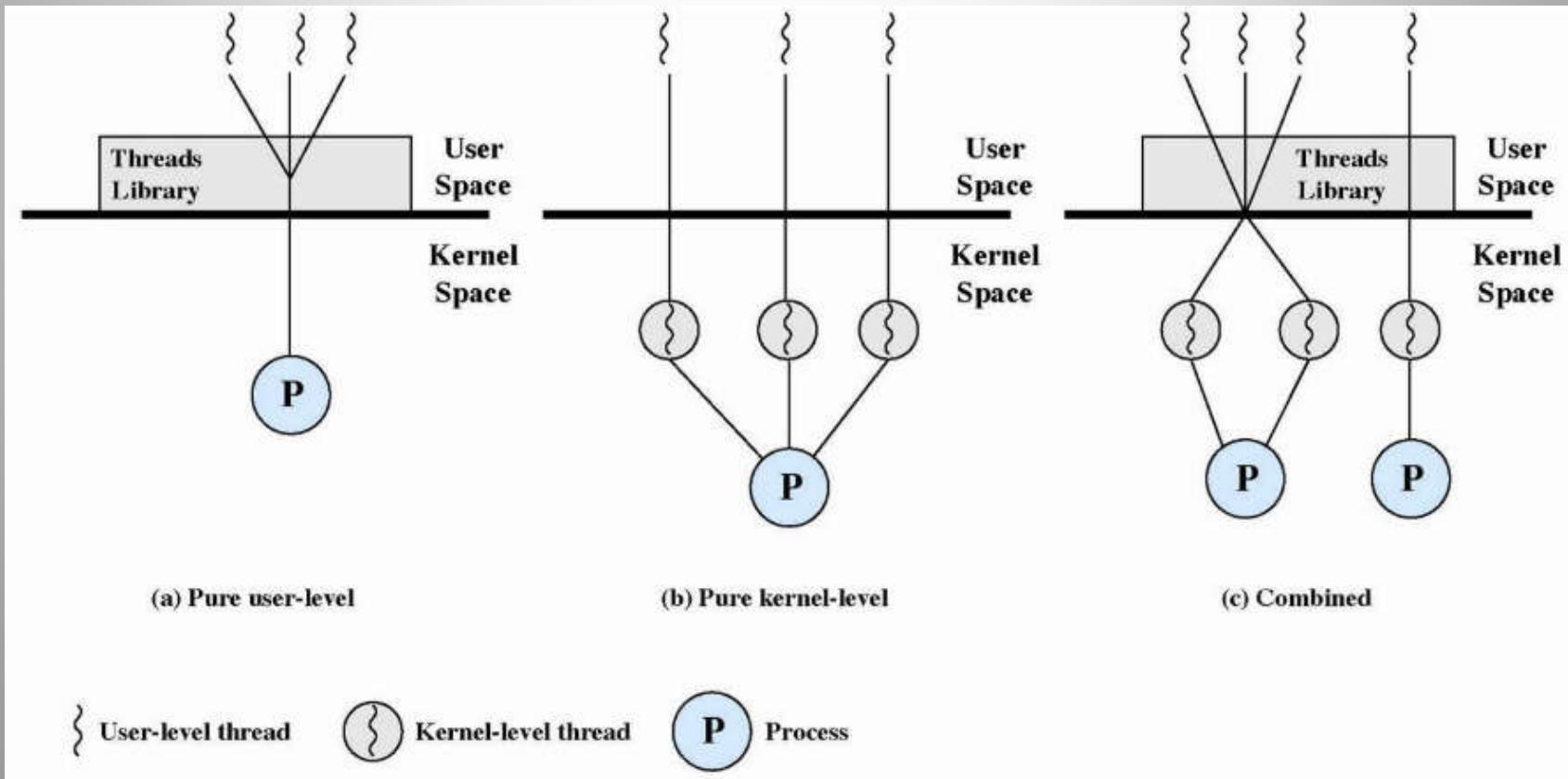
● **Cài đặt luồng với Windows**

● **Vấn đề đa luồng**

Chương 2 Quản lý tiến trình  
2. Luồng  
2.2. Mô hình đa luồng

## Giới thiệu

- Nhiều hệ thống hỗ trợ cả luồng mức người dùng và luồng mức hệ thống  $\Rightarrow$  Nhiều mô hình đa luồng khác nhau



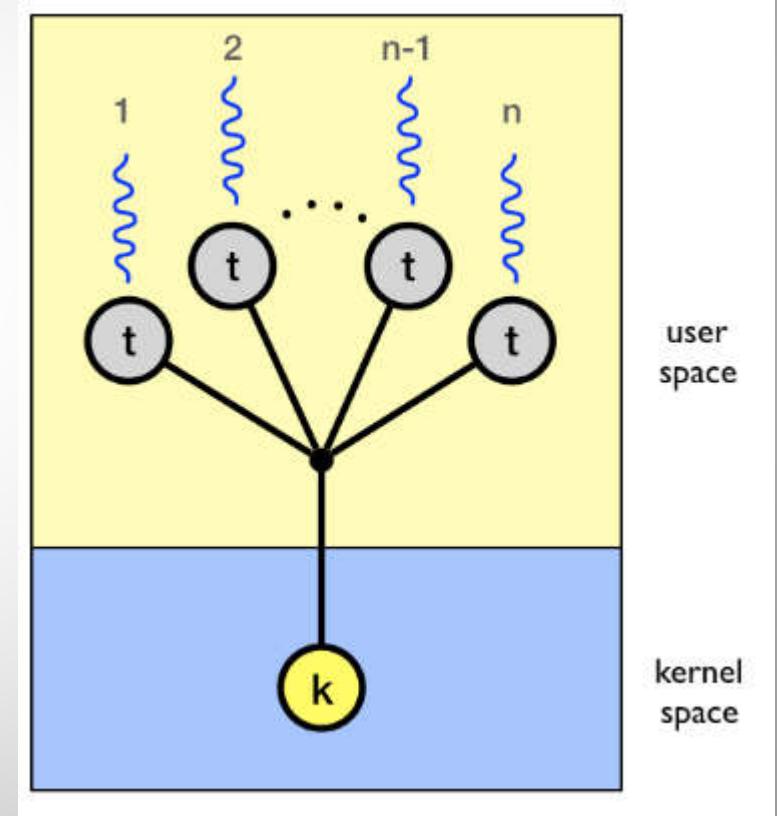
## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.2. Mô hình đa luồng

##### Mô hình nhiều-một

- Ánh xạ **nhiều luồng** mức **người dùng**  
->1 luồng mức **hệ thống**
- Quản lý luồng được thực hiện trong không gian người dùng
  - Hiệu quả
  - Cho phép tạo nhiều luồng tùy ý
  - Toàn bộ TT sẽ bị khóa nếu 1 luồng bị khóa
- Không thể chạy song song trên các máy nhiều vi xử lý (Chỉ 1 luồng có thể truy nhập nhân tại 1 thời điểm)
- Dùng trong HĐH không hỗ trợ luồng hệ thống

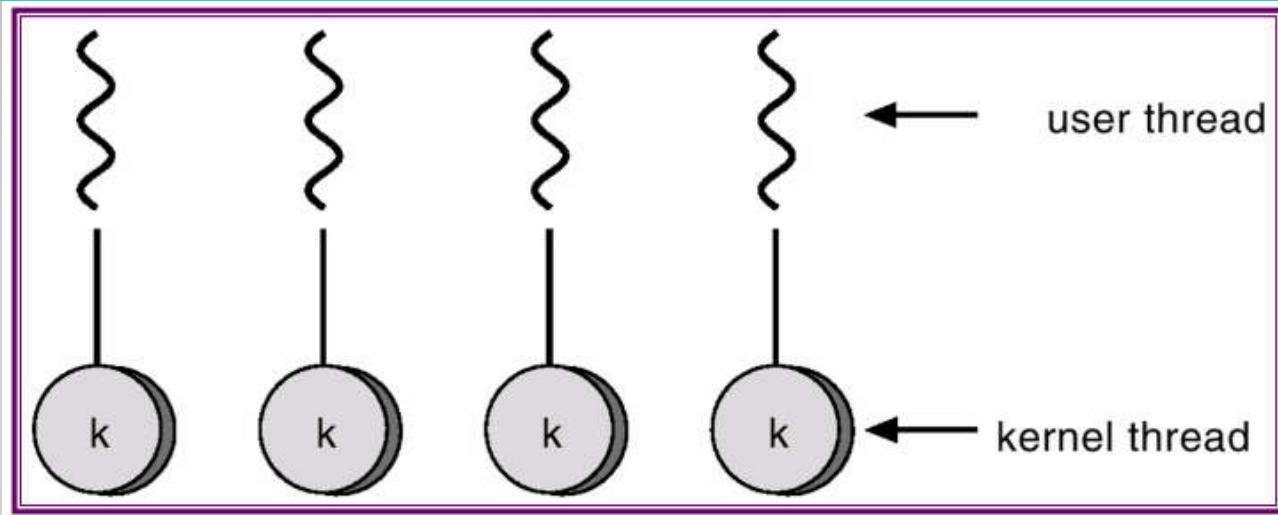


## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.2. Mô hình đa luồng

#### Mô hình một-một



- Ánh xạ mỗi luồng mức người dùng -> 1 luồng hệ thống
  - Cho phép thực hiện luồng khác khi 1 luồng bị chờ đợi
  - Cho phép chạy song song đa luồng trên máy nhiều vi xử lý
- Tạo luồng mức người dùng đòi hỏi tạo 1 luồng mức hệ thống tương ứng
  - Ảnh hưởng tới hiệu năng của ứng dụng
  - Chi phí cao ⇒ Giới hạn số luồng được hệ thống hỗ trợ
- Được sử dụng trong Window NT/2000/XP

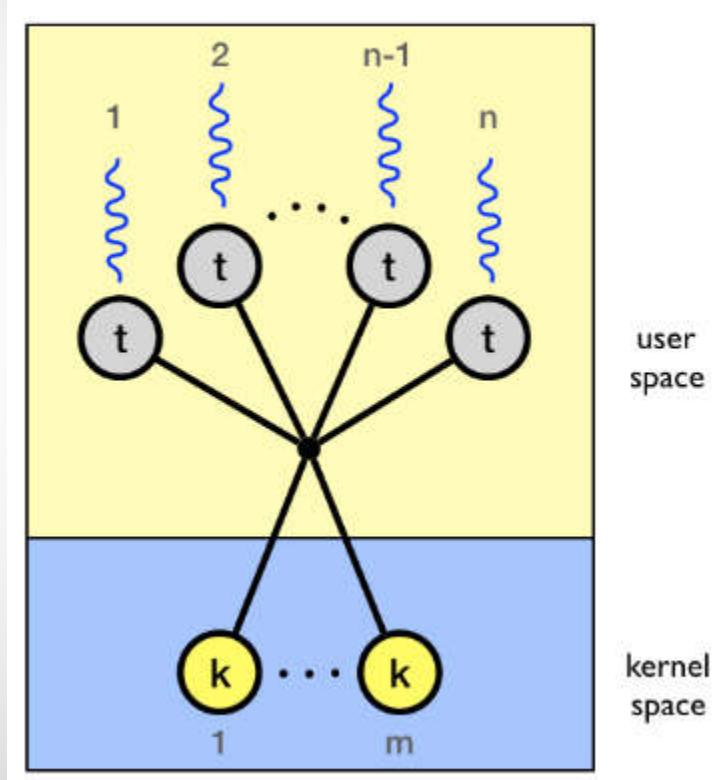
## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.2. Mô hình đa luồng

## Mô hình nhiều luồng

- Nhiều luồng mức người dùng ánh xạ tới một số nhỏ luồng mức hệ thống
- Số lượng luồng nhân có thể được xác định theo máy hoặc theo ứng dụng
  - VD: Được cấp nhiều luồng nhân hơn trên hệ thống nhiều VXL
- Có được ưu điểm của 2 mô hình trên
  - Cho phép tạo nhiều luồng mức ứng dụng theo yêu cầu
  - Các luồng nhân tương ứng có thể chạy song song trên hệ nhiều VXL
  - 1 luồng bị khóa, nhân có thể cho phép luồng khác thực hiện
- Ví dụ: UNIX



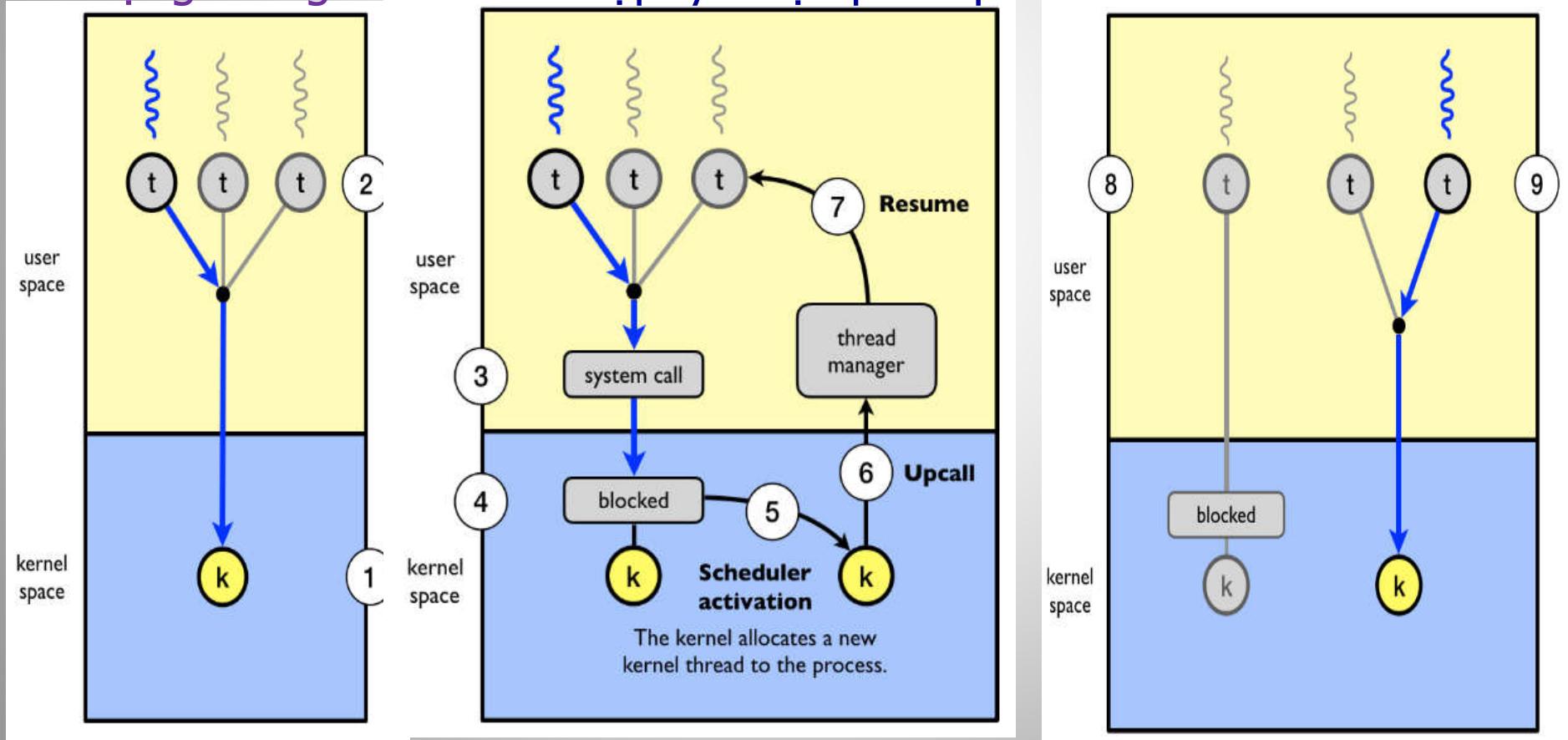
## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.2. Mô hình đa luồng

### Cơ chế kích hoạt bộ điều phối (scheduler activation)

Cách thức để kernel liên lạc với trình quản lý luồng mức user để duy trì số lượng luồng mức nhân hợp lý được phân phối cho TT



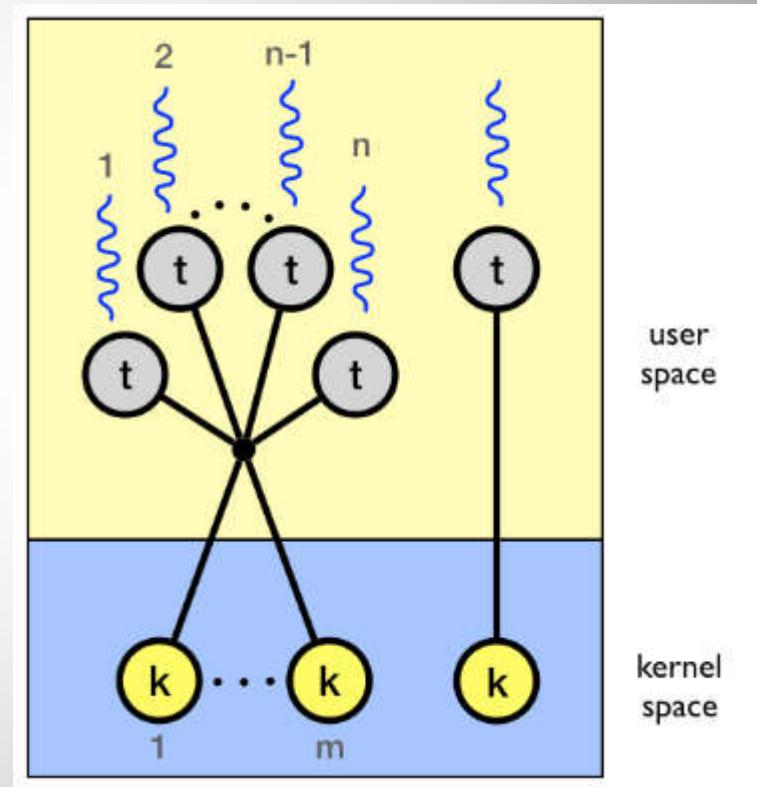
## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.2. Mô hình đa luồng

## Mô hình 2 mức (two-level)

- Kết hợp nhiều-nhiều và một-một
- Ưu tiên một số luồng cần được phục vụ nhiều hơn.



## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.3. Cài đặt luồng với Windows

- **Giới thiệu**
- **Mô hình đa luồng**
- Cài đặt luồng với Windows**
- **Vấn đề đa luồng**

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.3. Cài đặt luồng với Windows

### Một số hàm với luồng trong WIN32 API

- HANDLE CreateThread(. . .);
  - LPSECURITY\_ATTRIBUTES lpThreadAttributes,  
⇒ Trỏ tới cấu trúc an ninh: thẻ trả về có thể được kế thừa?
  - DWORD dwStackSize,  
⇒ Kích thước ban đầu của stack cho luồng mới
  - LPTHREAD\_START\_ROUTINE lpStartAddress,  
⇒ Trỏ tới hàm được thực hiện bởi luồng mới
  - LPVOID lpParameter,  
⇒ Trỏ tới các biến được gửi tới luồng mới (tham số của hàm)
  - DWORD dwCreationFlags,  
⇒ Phương pháp tạo luồng
    - CREATE\_SUSPENDED : Luồng ở trạng thái tạm ngừng
    - 0: Luồng được thực hiện ngay lập tức
  - LPDWORD lpThreadId  
⇒ Biến ghi nhận định danh luồng mới
- Kết quả trả về: Thẻ của luồng mới hoặc giá trị NULL nếu không tạo được luồng mới

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.3. Cài đặt luồng với Windows

#### Ví dụ

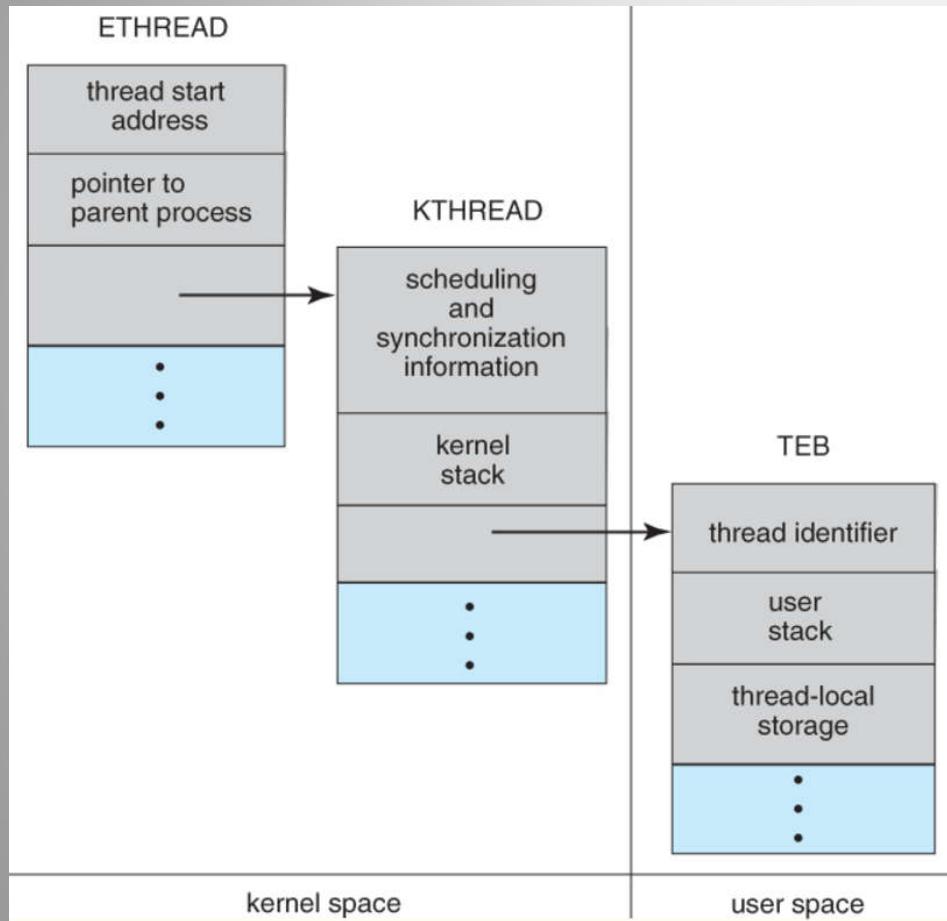
```
#include <windows.h>
#include <stdio.h>
void Routine(int *n){
printf("My argument is %d\n", *n);
}
int main(){
    int i, P[5]; DWORD Id;
    HANDLE hHandles[5];
    for (i=0;i < 5;i++) {
        P[i] = i;
        hHandles[i] = CreateThread(NULL,0,
                                (LPTHREAD_START_ROUTINE)Routine,&P[i],0,&Id);
        printf("Thread %d was created\n",Id);
    }
    for (i=0;i < 5;i++) WaitForSingleObject(hHandles[i],INFINITE);
    return 0;
}
```

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.3. Cài đặt luồng với Windows

## Luồng trong Windows XP



Thread bao gồm

- Thread ID
- Registers
- user stack dùng trong user mode, kernel stack dùng trong kernel mode.
- Vùng lưu trữ riêng được dùng bởi các thư viện khi thực hiện (run-time) và thư viện liên kết động

executive thread block

Kernel thread block

Thread environment block

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.4. Vấn đề đa luồng

- Giới thiệu
- Mô hình đa luồng
- Cài đặt luồng với Windows
- **Vấn đề đa luồng**

## Chương 2 Quản lí tiến trình

### 2. Luồng

#### 2.4. Vấn đề đa luồng

##### Ví dụ

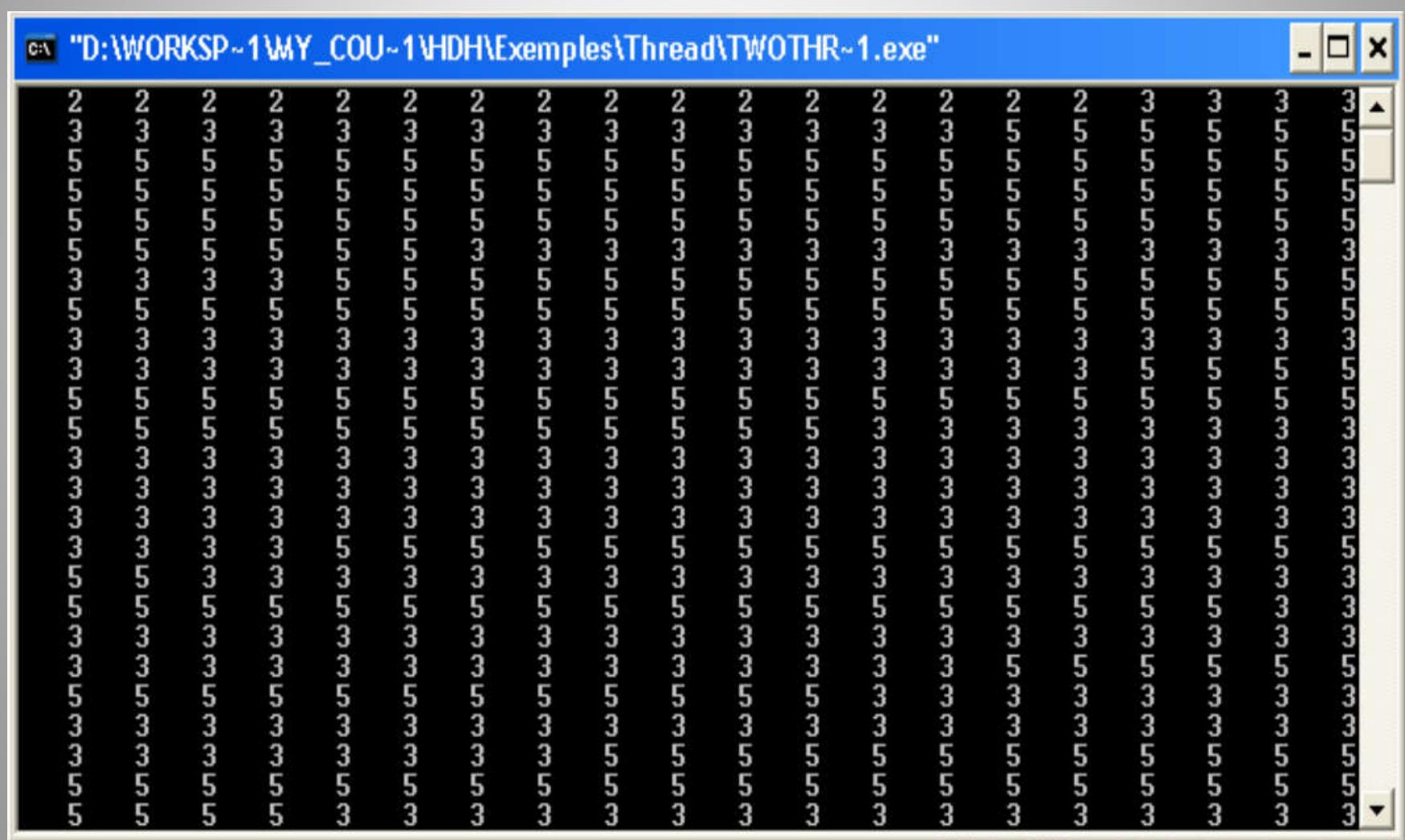
```
#include <windows.h>
#include <stdio.h>
int x = 0, y = 1;
void T1(){
    while(1){ x = y + 1; printf("%4d", x); }
}
void T2(){
    while(1){ y = 2; y = y * 2; }
}
int main(){
    HANDLE h1, h2; DWORD Id;
    h1=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)T1,NULL,0,&Id);
    h2=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)T2,NULL,0,&Id);
    WaitForSingleObject(h1,INFINITE);
    WaitForSingleObject(h2,INFINITE);
    return 0;
}
```

## Chương 2 Quản lý tiến trình

### 2. Luồng

#### 2.4. Vấn đề đa luồng

# Kết quả thực hiện



## Chương 2 Quản lý tiến trình

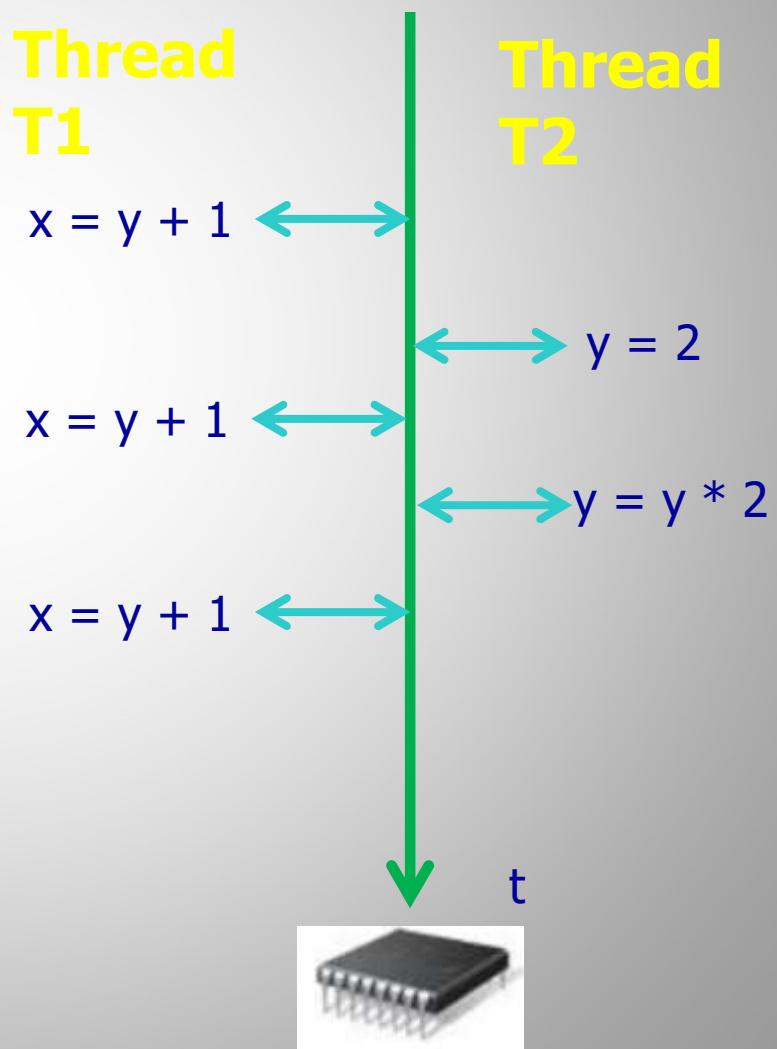
### 2. Luồng

#### 2.4. Vấn đề đa luồng

### Giải thích

Shared int $y = 1$	
<b>Thread <math>T_1</math></b>	<b>Thread <math>T_2</math></b>
$x \leftarrow y + 1$	$y \leftarrow 2$ $y \leftarrow y * 2$
$x = ?$	

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



## Chương 2 Quản lí tiến trình

- ① Tiến trình
- ② Luồng (Thread)
- ③ Điều phối CPU
- ④ Tài nguyên găng và điều độ tiến trình
- ⑤ Bẽ tắc và xử lý bẽ tắc

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

- Các khái niệm cơ bản
- Tiêu chuẩn điều phối
- Các thuật toán điều phối
- CPU Điều phối đa xử lý

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

#### Giới thiệu

- Hệ thống có 1 *processor* → Chỉ có 1 TT được thực hiện tại 1 thời điểm
- TT được thực hiện (*chiếm dụng VXL*) cho tới khi phải chờ đợi 1 thao tác vào ra
- Hệ đơn c/trình: CPU không được sử dụng ⇒ Lãng phí
- Hệ đa c/trình: cố gắng sử dụng CPU (đang rảnh rõi)
  - Cần nhiều TT sẵn sàng trong bộ nhớ tại 1 thời điểm
  - Khi 1 TT phải chờ, HĐH lấy lại processor để phân cho TT khác

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

#### Giới thiệu

- Điều phối processor quan trọng với HĐH đa nhiệm
- Luân chuyển CPU giữa các TT → khai thác hệ thống hiệu quả hơn
- *Điều phối processor là nền tảng trong thiết kế HĐH*

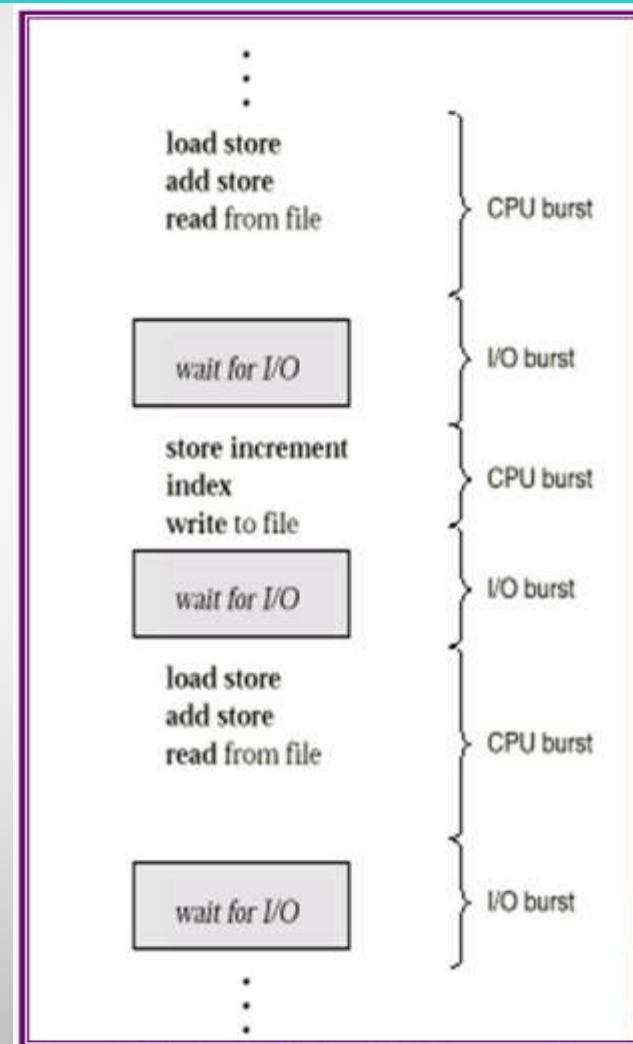
## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

### Chu kỳ thực hiện CPU - I/O

- TT là chuỗi luân phiên giữa chu kỳ tính toán và chờ đợi vào/ra
  - Bắt đầu bởi chu kỳ tính toán
  - Tiếp theo chu kỳ đợi vào/ra
  - Tính toán → đợi vào/ra → tính toán → đợi vào/ra → ...
  - Kết thúc: Tính toán (yêu cầu hệ thống kết thúc thực hiện)



## Chương 2 Quản lý tiến trình

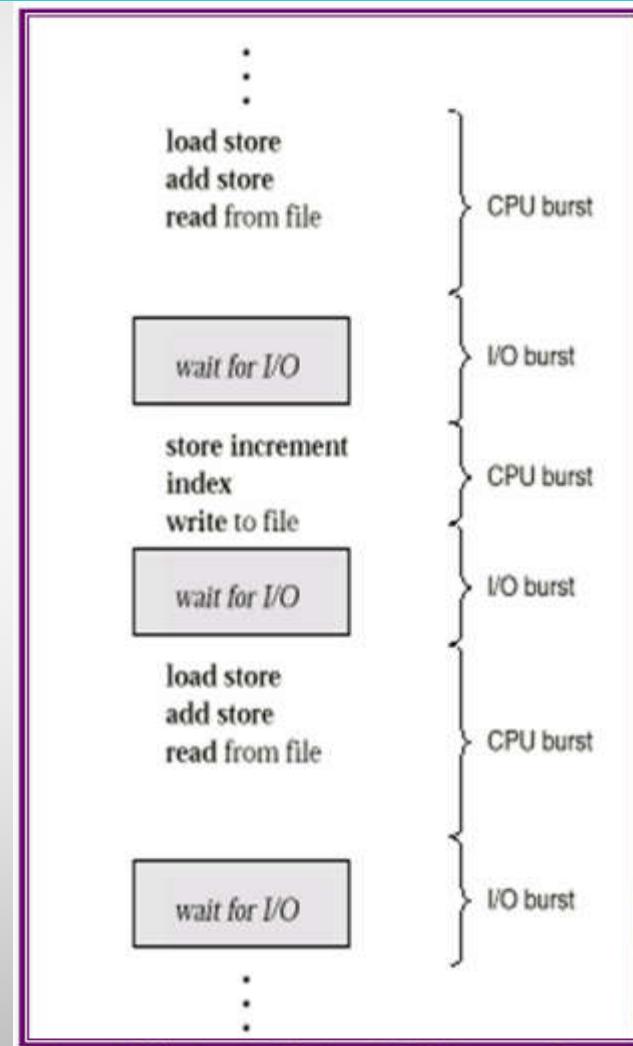
### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

## Chu kỳ thực hiện CPU - I/O

### ● Phân biệt các kiểu TT

- Dựa trên sự phân bổ thời gian cho các chu kỳ CPU & vào/ra
  - TT tính toán (CPU-bound process) có vài chu kỳ CPU dài
  - TT vào ra (I/O-bound process) có nhiều chu kỳ CPU ngắn
- Để chọn giải thuật điều phối thích hợp



## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

## Bộ điều phối CPU

- Lựa chọn 1 trong số các TT đang sẵn sàng trong bộ nhớ và cung cấp CPU cho nó
- Các TT phải sắp hàng trong hàng đợi
  - Hàng đợi FIFO, Hàng đợi ưu tiên, DSLK đơn giản . . .

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

## Bộ điều phối CPU

- Quyết định điều phối CPU xảy ra khi TT chuyển từ t/thái
- 1) thực hiện → t/thái chờ đợi (y/c vào/ra)
- 2) thực hiện → t/thái sẵn sàng (CPU → ngắt t/gian)
- 3) chờ đợi → t/thái sẵn sàng (hoàn thành vào/ra)
- 4) Sang kết thúc
- Ghi chú
  - T/hợp 1&4 ⇒ Đ/phối không trưng dụng (non-preemptive)
  - T/hợp khác ⇒ Đ/phối trưng dụng (preemptive)

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

#### Điều phối trưng dụng và không trưng dụng

#### Điều phối không trưng dụng

- TT chiếm CPU cho tới khi chủ động giải phóng:
  - Kết thúc nhiệm vụ
  - Chuyển sang trạng thái chờ đợi
- Không đòi hỏi phần cứng đặc biệt (đồng hồ)
- Ví dụ: DOS, Win 3.1, Macintosh

## Điều phối trung dung và không trung dung

### Điều phối trung dung

- TT chỉ được phép thực hiện trong 1 khoảng t/gian
- Hết t/gian, ngắt t/gian xuất hiện, bộ điều vận (dispatcher) quyết định phục hồi lại TT hay chọn TT khác
- Bảo vệ CPU khỏi các TT "đói-CPU"
- Vấn đề dữ liệu dùng chung
  - TT 1 đang cập nhật DL thì bị mất CPU
  - TT 2, được giao CPU và đọc DL đang cập nhật
- Ví dụ: HĐH đa nhiệm WinNT, UNIX

## Chương 2 Quản lí tiến trình

### 3. Điều phối CPU

#### 3.1. Các khái niệm cơ bản

Điều phối trưng dụng và không trưng dụng



## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.2. Tiêu chuẩn điều phối

- Các khái niệm cơ bản
- Tiêu chuẩn điều phối
- Các thuật toán điều phối
- CPU Điều phối đa xử lý

## Chương 2 Quản lí tiến trình

### 3. Điều phối CPU

#### 3.2. Tiêu chuẩn điều phối

Tiêu chuẩn điều phối

I



## ➤ Tất cả hệ thống nói chung

- Công bằng

- Chia sẻ CPU công bằng giữa các TT
- Không phải chờ đợi vô hạn

➤ Các chiến lược điều phối đề ra phải được tuân thủ

➤ Cân bằng tải: mọi thành phần của hệ thống đều  
bận rộn

## ➤ Hệ thống xử lý theo lô

## ➤ Hệ thống xử lý theo lô

### ➤ Sử dụng CPU (Lớn nhất)

- Mục đích của điều độ là làm CPU hoạt động nhiều nhất có thể
- Độ sử dụng CPU thay đổi từ 40% (hệ thống tải nhẹ) đến 90% (hệ thống tải nặng).

### ➤ Thông lượng (throughput) (Lớn nhất)

- Số lượng TT hoàn thành trong 1 đơn vị thời gian
- Các TT dài: 1 TT/giờ
- Các TT ngắn: 10 TT/giây

## ➤ Hệ thống xử lý theo lô

### ● Thời gian hoàn thành (Nhỏ nhất)

- Khoảng t/gian từ thời điểm gửi đến hệ thống tới khi TT kết thúc. Gồm các khoảng t/gian chờ đợi
  - để đưa TT vào bộ nhớ
  - trong hàng đợi sẵn sàng
  - trong hàng đợi thiết bị
  - thực hiện thực tế

## ➤ **Hệ thống xử lý theo lô**

### ● Thời gian chờ đợi (Nhỏ nhất)

#### ● Tổng t/gian chờ trong hàng đợi sẵn sàng

- (Giải thuật điều độ CPU không ảnh hưởng tới các TT đang thực hiện hay đang đợi thiết bị vào ra)

## ➤ Hệ thống tương tác

- Nhanh chóng phản hồi yêu cầu của người dùng
- Thời gian đáp ứng (Nhỏ nhất)
  - Từ lúc gửi câu hỏi cho tới khi câu trả lời đầu tiên được tạo ra
  - TT có thể tạo kết quả ra từng phần
  - TT vẫn tiếp tục tính toán kết quả mới trong khi kết quả cũ được gửi tới người dùng

## ➤ Hệ thống thời gian thực

- Kịp thời hạn mà không mất mát dữ liệu
- Có khả năng dự đoán được việc giảm hiệu năng trong hệ thống đa phương tiện

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

- Các khái niệm cơ bản
- Tiêu chuẩn điều phối
- Các thuật toán điều phối
- CPU Điều phối đa xử lý

- **Giả thiết:** Các TT chỉ có 1 chu kỳ tính toán (ms)
- **Đo đạc:** Thời gian chờ đợi trung bình

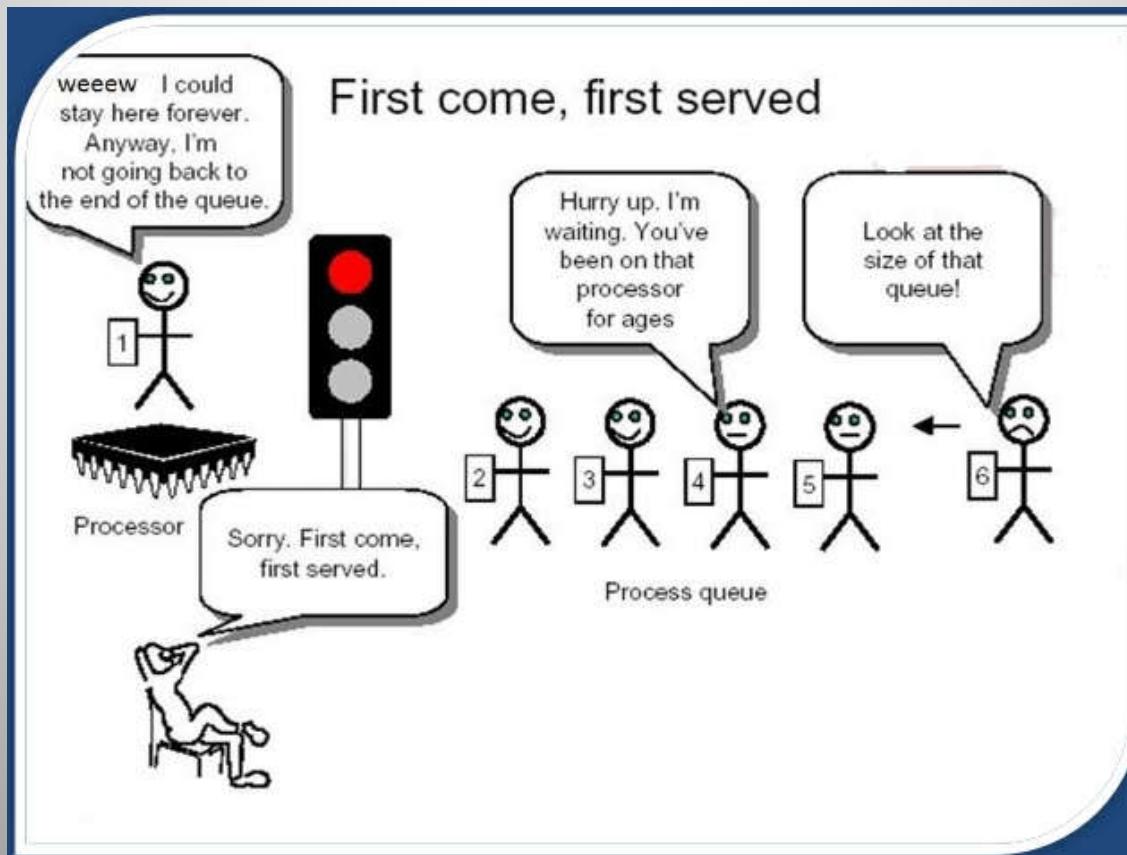
## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

## Đến trước phục vụ trước (FCFS: First Come, First Served)

- Nguyên tắc: TT được quyền sử dụng CPU theo trình tự xuất hiện  
TT sở hữu CPU tới khi kết thúc hoặc chờ đợi vào/ra



## Chương 2 Quản lý tiến trình

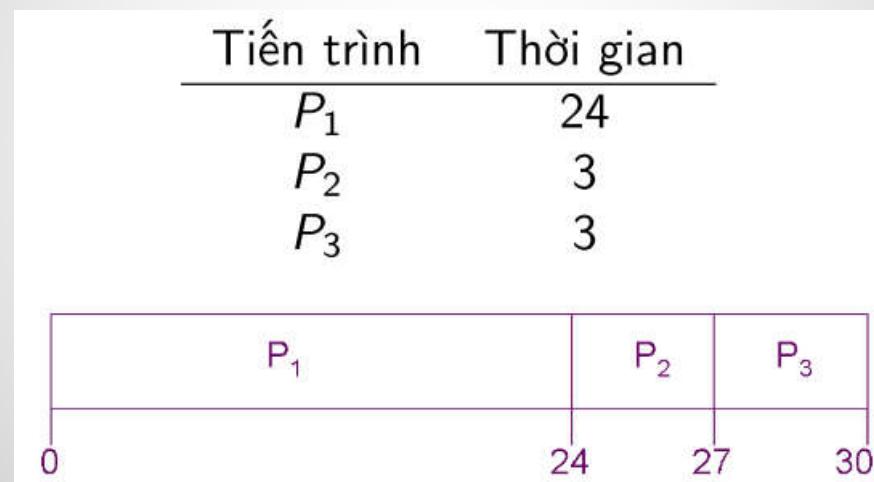
### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Đến trước phục vụ trước (FCFS: First Come, First Served)

- Nguyên tắc: TT được quyền sử dụng CPU theo trình tự xuất hiện  
TT sở hữu CPU tới khi kết thúc hoặc chờ đợi vào/ra

#### Ví dụ



#### Đặc điểm

- Đơn giản, dễ thực hiện
- TT ngắn phải chờ đợi như TT dài

● Nếu  $P_1$  thực hiện sau cùng ?

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Công việc ngắn trước (SJF: Shortest Job First)

- Nguyên tắc: Mỗi TT lưu trữ thời gian của **chu kỳ sử dụng CPU** tiếp theo
  - TT có thời gian sử dụng CPU ngắn nhất sẽ sở hữu CPU
  - 2 phương pháp
    - Không trưng dụng CPU
    - Có trưng dụng CPU (SRTF: Shortest Remaining Time First)

#### Ví dụ

Tiến trình	Thời gian	Thời điểm đến
$P_1$	8	0.0
$P_2$	4	1.0
$P_3$	9	2.0
$P_4$	5	3.0

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

## Điều phối có ưu tiên (Priority Scheduling)

- Mỗi TT gắn với 1 số hiệu ưu tiên (số nguyên)
  - CPU sẽ được phân phối cho TT có **độ ưu tiên cao nhất**
  - SJF: độ ưu tiên gắn liền với thời gian thực hiện
  - 2 phương pháp
    - Không trưng dụng CPU
    - Có trưng dụng CPU
- Ví dụ

Tiến trình	Thời gian	Độ ưu tiên
$P_1$	10	3
$P_2$	1	1
$P_3$	2	4
$P_4$	1	5
$P_5$	5	2

## Chương 2 Quản lý tiến trình

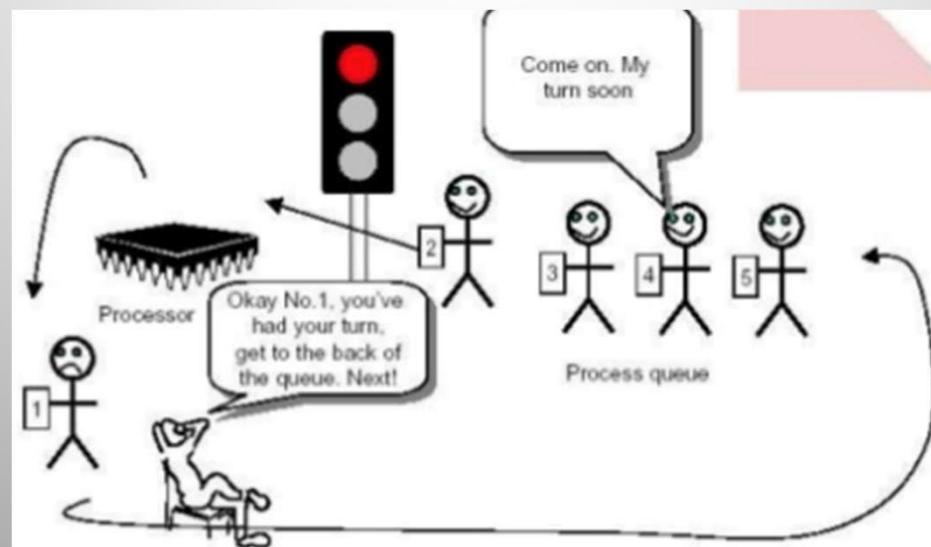
### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

## Vòng tròn (RR: Round Robin Scheduling)

### Nguyên tắc

- Mỗi TT được cấp 1 lượng tử t/gian  $T$  để thực hiện
- Khi hết t/gian, TT bị trưng dụng processor và được đưa vào cuối hàng đợi sẵn sàng
- Nếu có  $n$  TT, t/gian chờ đợi nhiều nhất ( $n - 1$ )  $T$



## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Vòng tròn (RR: Round Robin Scheduling)

#### ● Ví dụ

Tiến trình	Thời gian
$P_1$	24
$P_2$	3
$P_3$	3

Lượng tử thời gian  $\tau = 4$

#### ● Vấn đề: Lựa chọn lượng tử t/gian $\tau$

- $\tau$  lớn: FCFS
- $\tau$  nhỏ: Hay phải luân chuyển CPU
- Thông thường  $\tau = 10-100ms$

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

#### Vòng tròn (RR: Round Robin Scheduling)

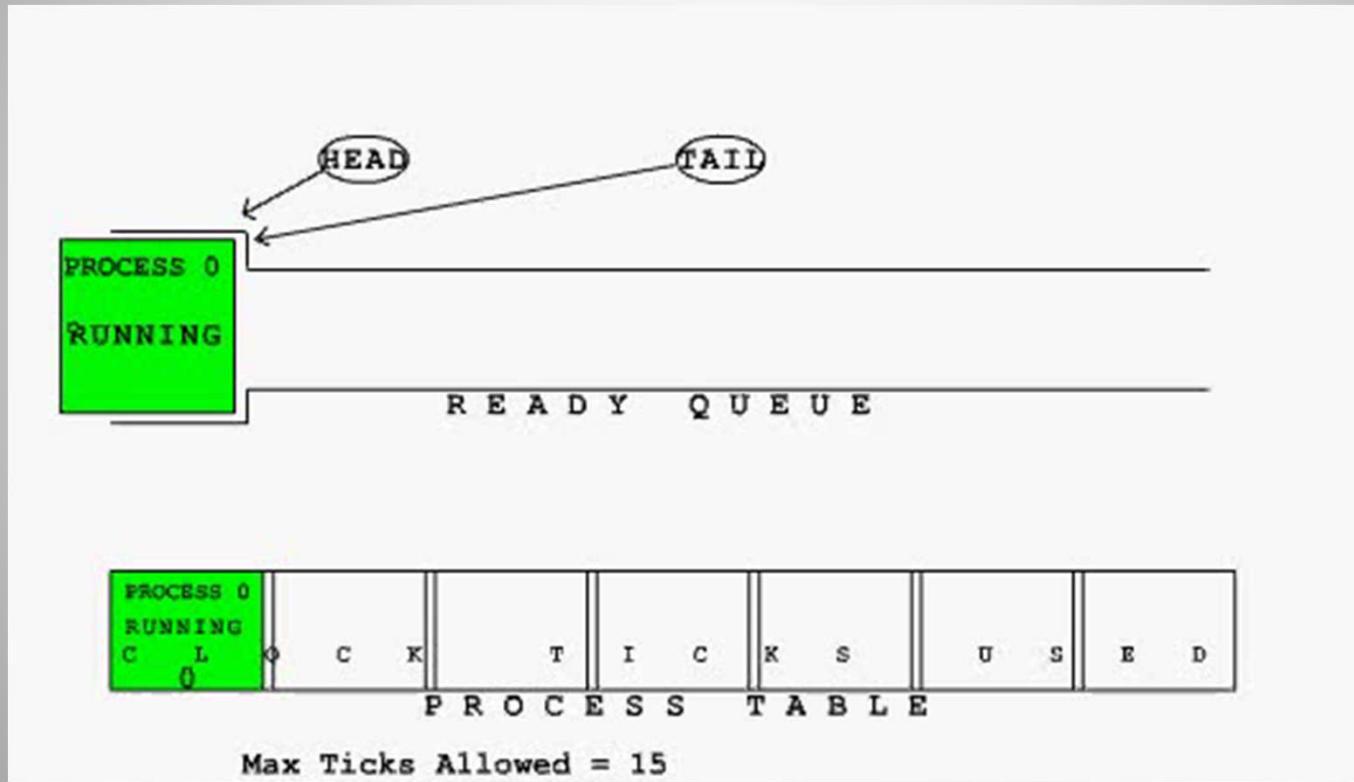
Process	CPU-burst	Arrival Times
P1	6	0
P2	3	1
P3	3	2
P4	5	4
P5	4	12

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

## Vòng tròn (RR: Round Robin Scheduling)



## Điều phối hàng đợi đa mức (Multilevel Queue Scheduling)

- Hàng đợi sẵn sàng được phân chia thành nhiều hàng đợi nhỏ
- Phổ biến nhất là foreground và back ground
  - 2 kiểu TT có các yêu cầu phản hồi khác nhau
- TT được ấn định cố định cho một hàng đợi
  - Dựa vào tính chất như: độ ưu tiên, kiểu TT, kích thước bộ nhớ...
- Mỗi hàng đợi sử dụng thuật toán điều độ riêng

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Điều phối hàng đợi đa mức (Multilevel Queue Scheduling)

#### Giữa các hàng đợi

- Cần điều phối

- Điều phối có trưng dụng, độ ưu tiên cố định

- TT hàng đợi độ ưu tiên thấp chỉ được thực hiện khi các hàng đợi có độ ưu tiên cao rỗng

- TT độ ưu tiên mức cao, trưng dụng TT độ ưu tiên mức thấp

- Có thể gặp tình trạng *starvation*

- Phân chia thời gian giữa các hàng đợi của

- TT foreground, chiếm 80% t/gian CPU cho RR

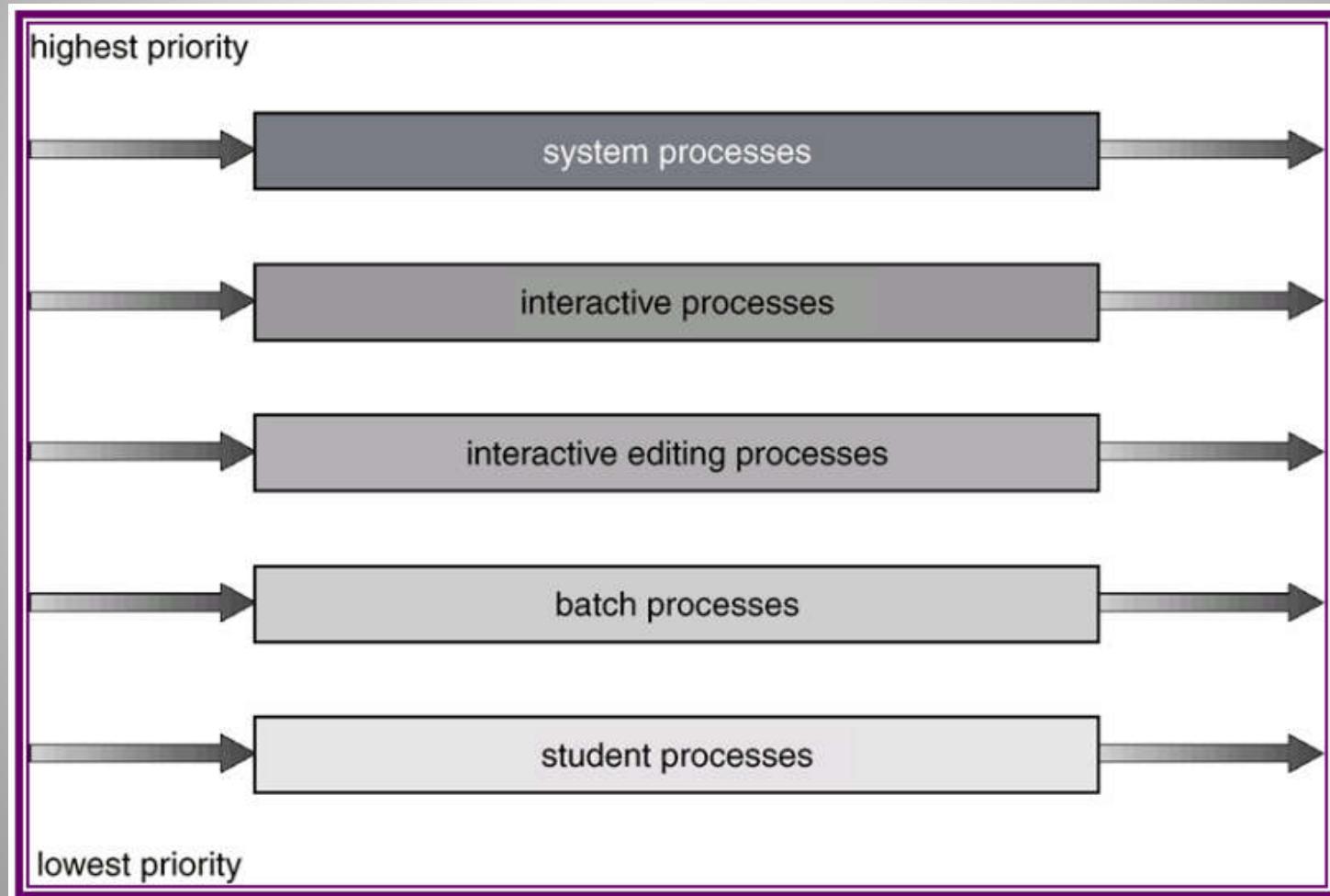
- TT background, chiếm 20% t/gian CPU cho FCFS

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Điều phối hàng đợi đa mức (Ví dụ)



## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

### Điều phối hàng đợi đa mức (Multilevel Queue Scheduling)

Ví dụ: Queue 1 dung RR với quantum =2

Hàng đợi 2 dùng FCFS

Process	Queue No	Arrival Time	Burst Time
P1	1	0	4
P2	1	0	3
P3	2	0	8
P4	1	10	5

## Hàng đợi hồi tiếp đa mức (Multilevel Feedback Queue)

- Cho phép các TT được **dịch chuyển giữa các hàng đợi**
- Phân chia TT theo đặc điểm sử dụng VXL
  - Nếu dùng quá nhiều t/gian của VXL → Chuyển xuống hàng đợi có độ ưu tiên thấp
  - vào ra nhiều → hàng đợi có độ ưu tiên cao
  - đợi quá lâu tại hàng đợi có độ ưu tiên thấp → Chuyển lên hàng đợi độ ưu tiên cao
- Ngăn ngừa tình trạng "đói CPU"

## Hàng đợi hồi tiếp đa mức (Multilevel Feedback Queue)

● Bộ điều phối được xây dựng dựa trên các tham số

- Số hàng đợi
- Thuật toán điều độ cho mỗi hàng đợi
- Điều kiện TT được chuyển lên/xuống hàng đợi có độ ưu tiên cao/thấp hơn
- Phương pháp xác định một hàng đợi khi TT cần phục vụ

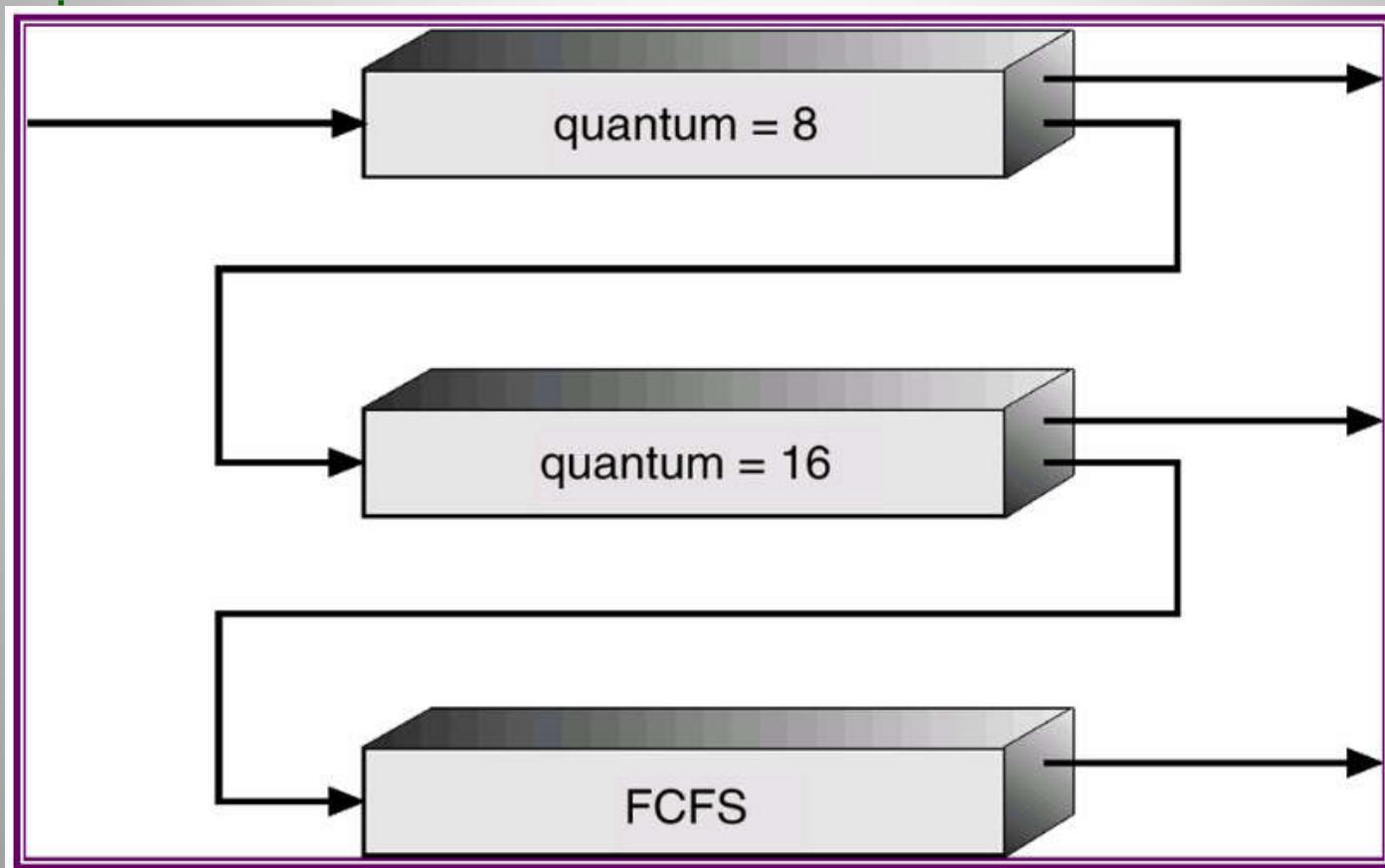
## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

## Hàng đợi hồi tiếp đa mức

### Ví dụ



## Chương 2 Quản lý tiến trình

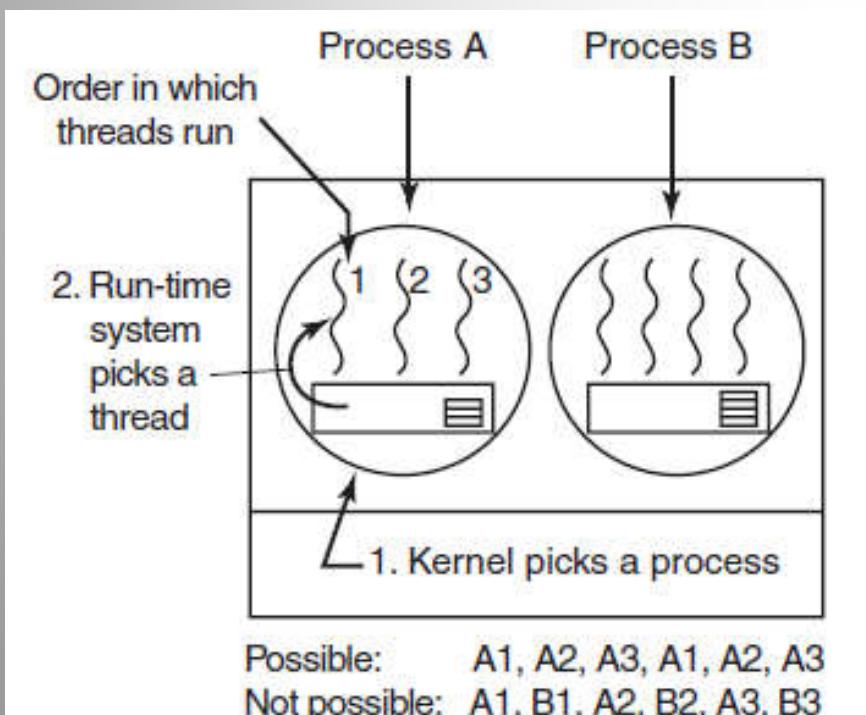
### 3. Điều phối CPU

#### 3.3. Các thuật toán điều phối

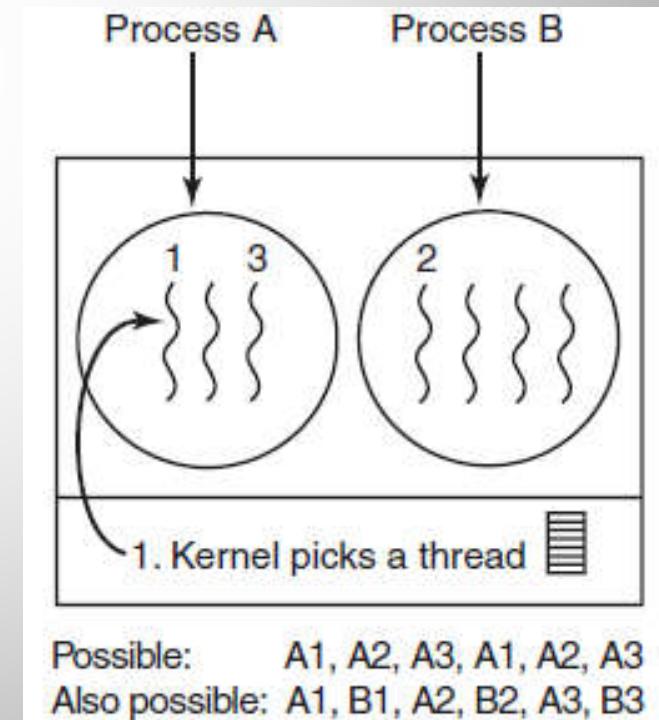
## Điều phối luồng

● 2 cấp độ song song: TT và luồng

● Phổ biến là RR



User level thread



Kernel level thread

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.4. Điều phối đa xử lý

- Các khái niệm cơ bản
- Tiêu chuẩn điều phối
- Các thuật toán điều phối
- CPU Điều phối đa xử lý

## Vấn đề

- Điều phối phức tạp hơn so với trường hợp có 1 VXL
- Vấn đề chia sẻ tài
- 2 mô hình
  - Đa xử lý đối xứng
  - Đa xử lý không đối xứng

## Chương 2 Quản lý tiến trình

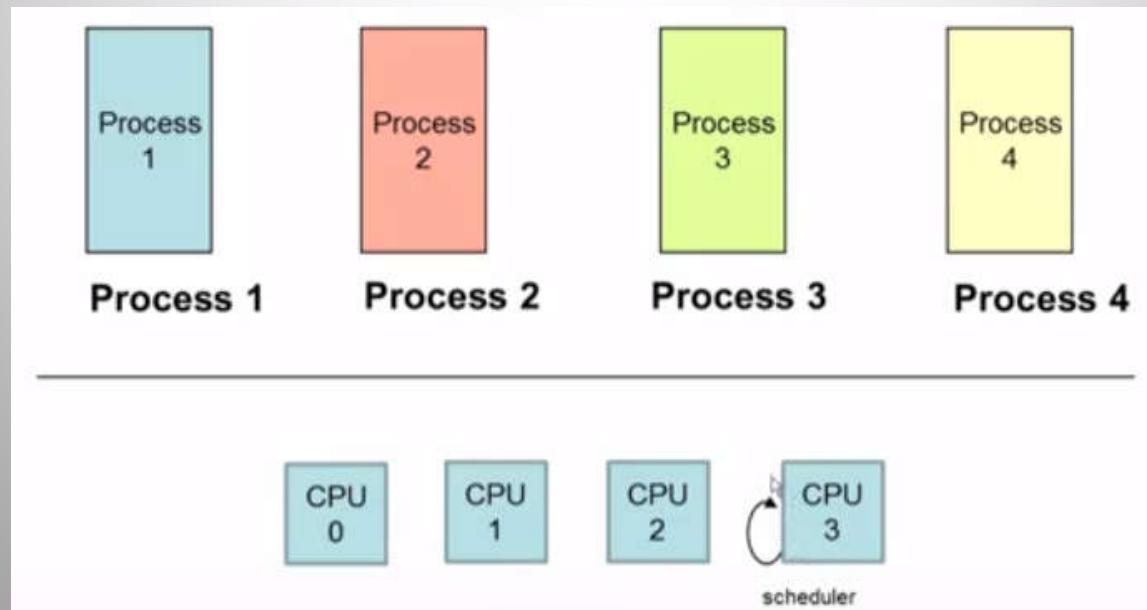
### 3. Điều phối CPU

#### 3.4. Điều phối đa xử lý

#### Vấn đề

##### ● Đa xử lý không đối xứng

- 1 processor chạy c/trình điều phối
- Chỉ có 1 processor truy nhập hàng đợi
- -> Có thể tắc nghẽn tại 1 processor



## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

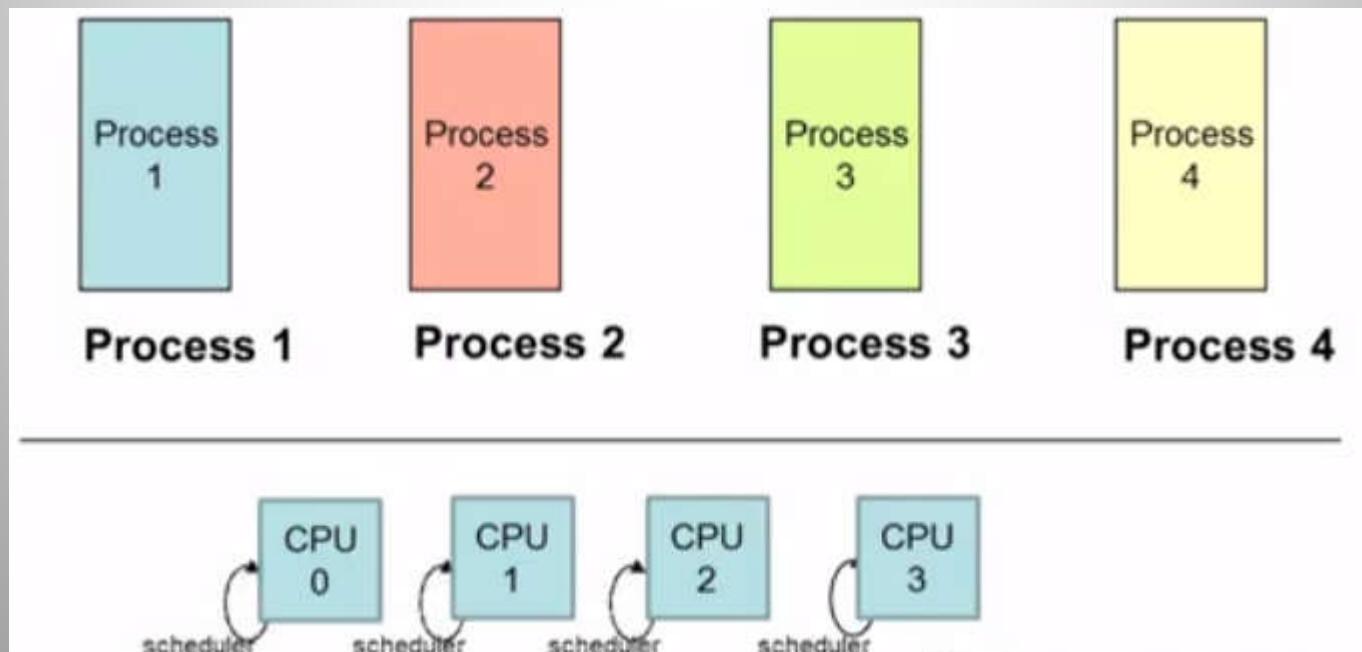
#### 3.4. Điều phối đa xử lý

### Vấn đề

#### ● Đa xử lý đối xứng

- Mỗi processor chạy c/trình điều phối riêng

- Độc lập lựa chọn TT sẵn sàng trong hàng đợi



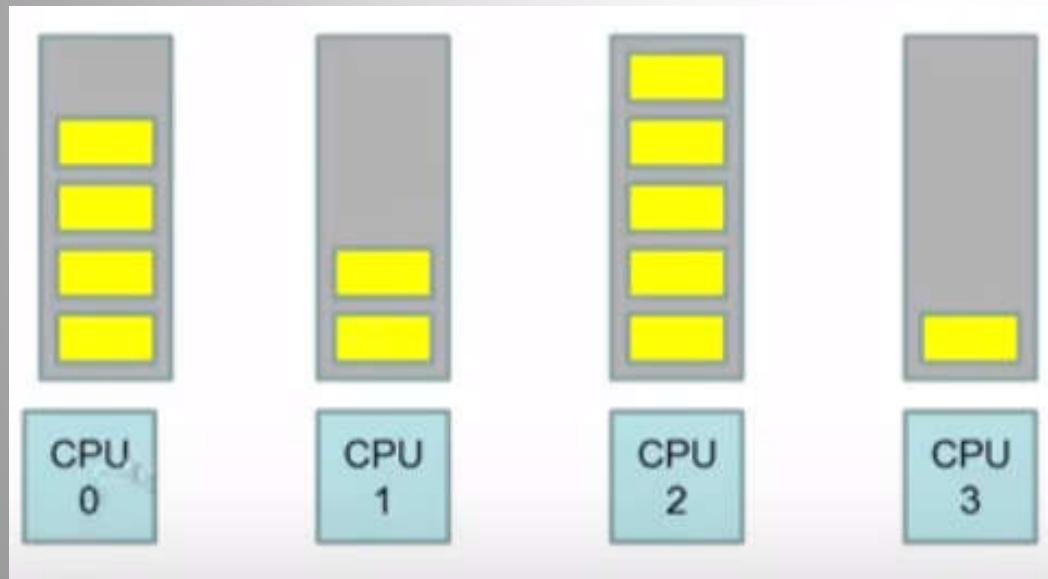
## Chương 2 Quản lí tiến trình

### 3. Điều phối CPU

#### 3.4. Điều phối đa xử lý

### Đa xử lý đối xứng

- Mỗi VXL có 1 hàng đợi sẵn sàng riêng
- chia các TT vào hàng đợi cố định



- **Ưu:** Dễ tổ chức t/hiện
- **Nhược:** Tồn tại VXL rảnh rỗi với hàng đợi rỗng trong khi VXL khác phải tính toán nhiều

## Chương 2 Quản lý tiến trình

### 3. Điều phối CPU

#### 3.4. Điều phối đa xử lý

## Đa xử lý đối xứng

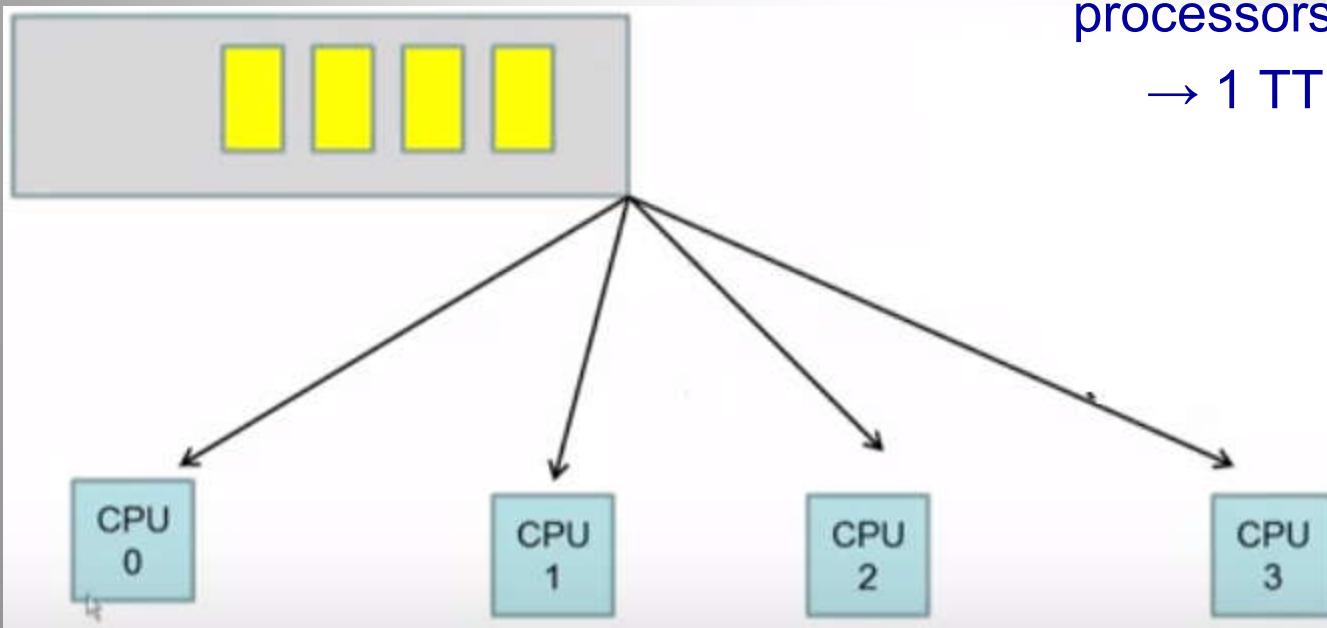
- Hàng đợi sẵn sàng dùng chung (Global queue)

### Ưu điểm:

- Sử dụng hiệu quả CPU
- Công bằng với các TT

### Nhược:

- Vấn đề dùng chung cấu trúc dữ liệu (hàng đợi):
  - 1 TT được lựa chọn bởi 2 processors hoặc
  - 1 TT bị thất lạc trên hàng đợi

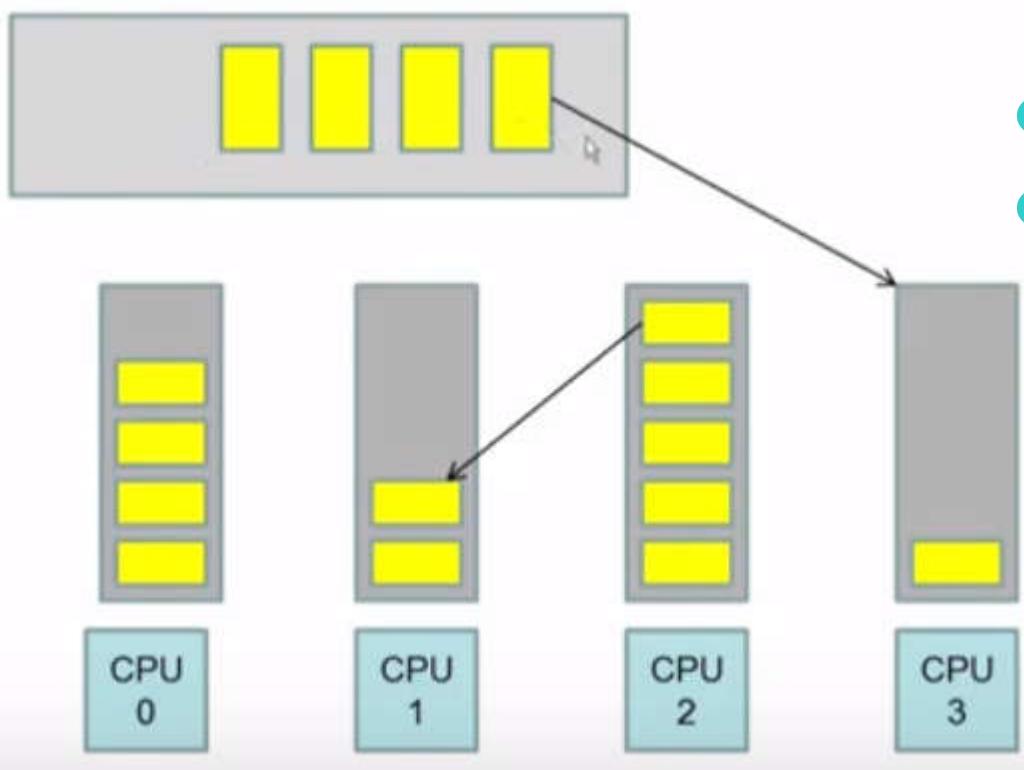


Chương 2 Quản lí tiến trình  
3. Điều phối CPU  
3.4. Điều phối đa xử lý

## Đa xử lý đối xứng

- Kết hợp 2 kiểu hàng đợi

- Dùng cả local và global queue
- Chia sẻ tải từ các hàng đợi
- Mỗi CPU làm việc với local queue của mình



Chương 2 Quản lý tiến trình  
3. Điều phối CPU

Bài tập

- Viết chương trình mô phỏng hàng đợi hồi tiếp đa mức

## Chương 2 Quản lí tiến trình

- ① Tiến trình
- ② Luồng (Thread)
- ③ Điều phối CPU
- ④ Tài nguyên găng và điều độ tiến trình
- ⑤ Bẽ tắc và xử lý bẽ tắc

## Chương 2 Quản lý tiến trình

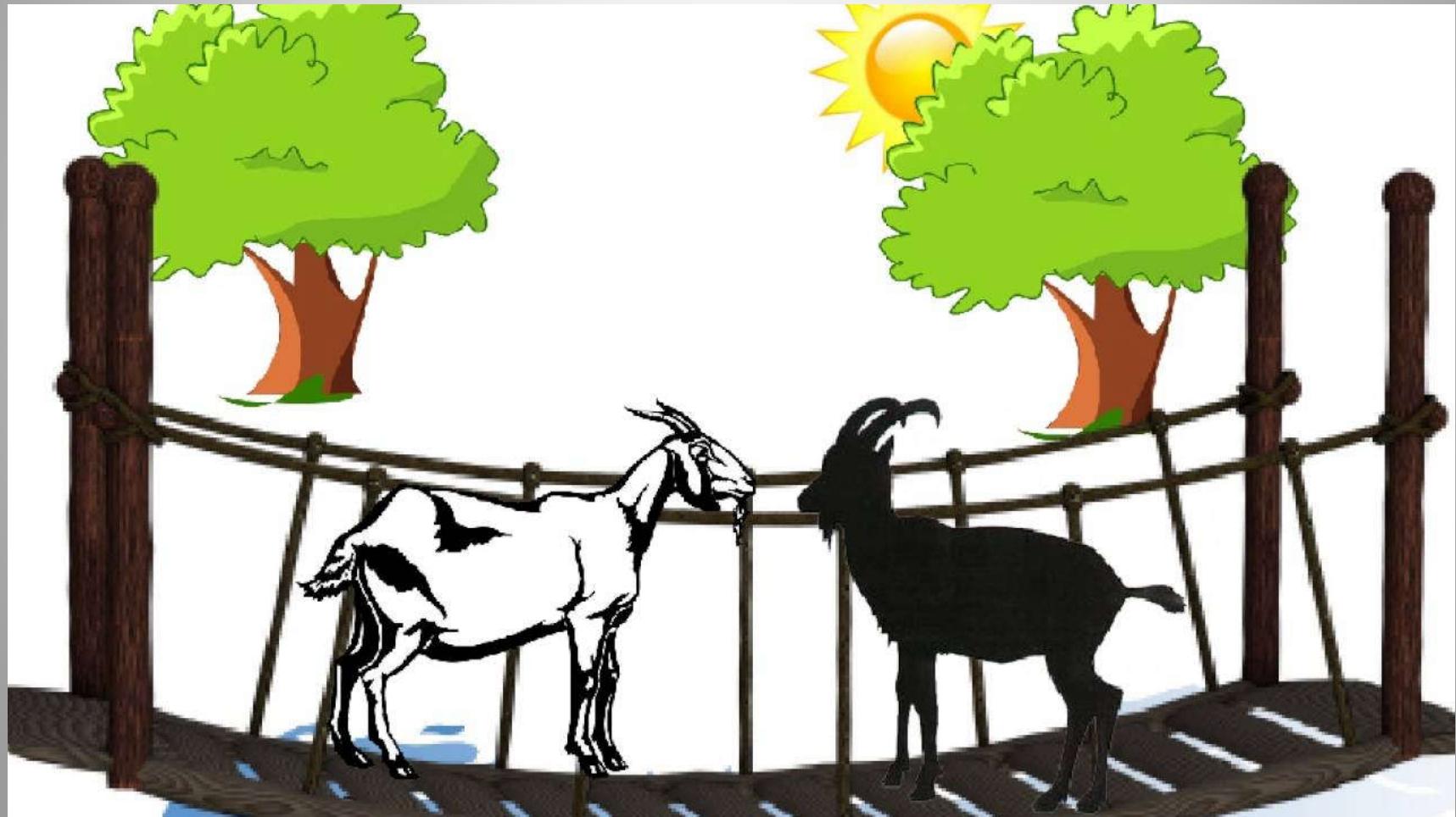
### 4. Tài nguyên găng và điều độ tiến trình

- Khái niệm tài nguyên găng
- Phương pháp khóa trong
- Phương pháp kiểm tra và xác lập
- Kỹ thuật đèn báo
- Ví dụ về đồng bộ tiến trình
- Công cụ điều độ cấp cao

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng



## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1. Khái niệm tài nguyên găng

#### Ví dụ

```
#include <windows.h>
#include <stdio.h>
int x = 0, y = 1;
void T1(){
    while(1){ x = y + 1; printf("%4d", x); }
}
void T2(){
    while(1){ y = 2; y = y * 2; }
}
int main(){
    HANDLE h1, h2; DWORD Id;
    h1=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)T1,NULL,0,&Id);
    h2=CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)T2,NULL,0,&Id);
    WaitForSingleObject(h1,INFINITE);
    WaitForSingleObject(h2,INFINITE);
    return 0;
}
```

## Chương 2 Quản lí tiến trình

#### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

# Kết quả thực hiện

## Chương 2 Quản lý tiến trình

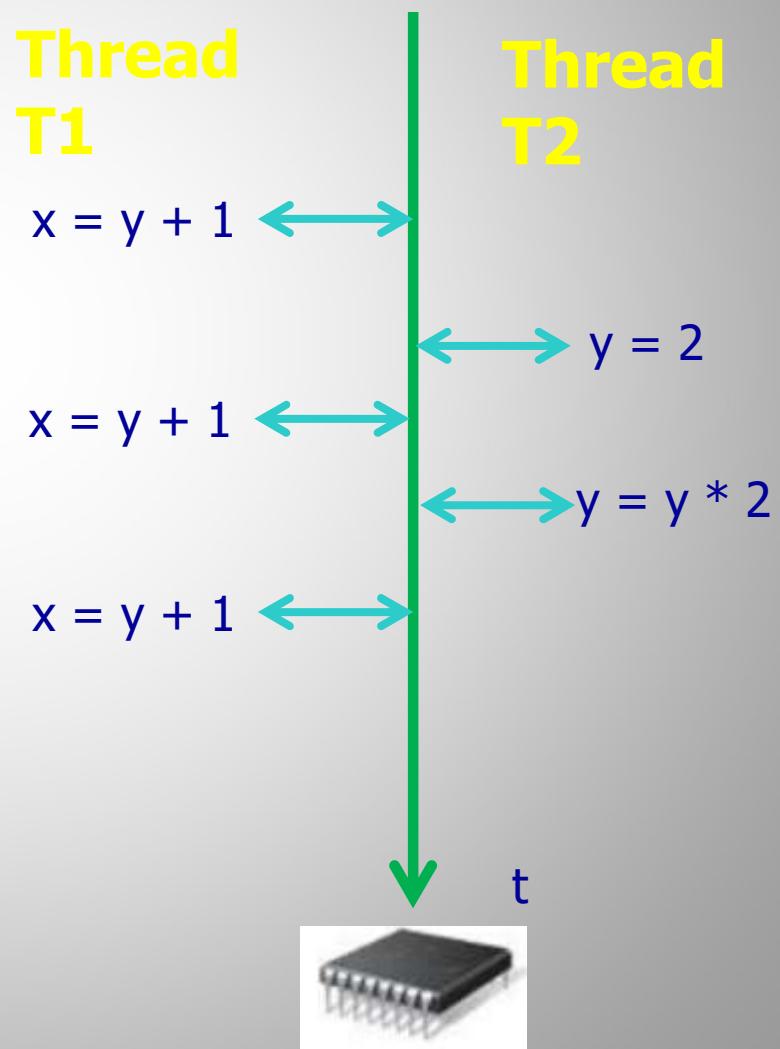
### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Giải thích

Shared int $y = 1$	
<b>Thread <math>T_1</math></b>	<b>Thread <math>T_2</math></b>
$x \leftarrow y + 1$	$y \leftarrow 2$ $y \leftarrow y * 2$
$x = ?$	

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



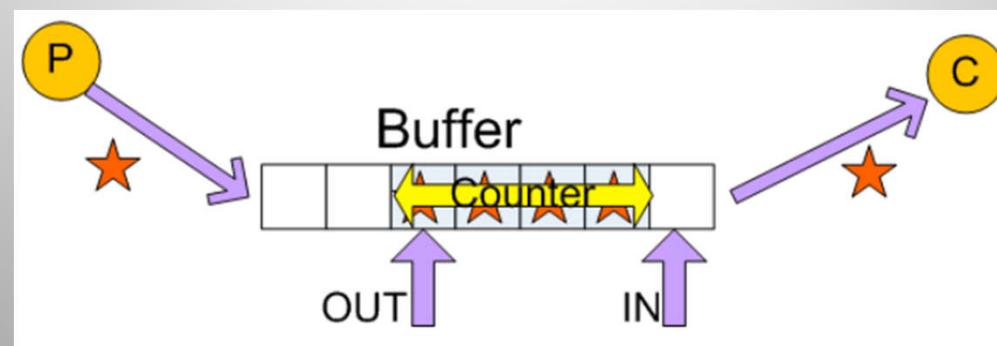
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) I

- Hệ thống gồm 2 TT
  - Producer sản xuất ra các sản phẩm
  - Consumer tiêu thụ các sản phẩm được sản xuất ra
- Công việc của Producer phải đồng bộ với Consumer.
  - Không đưa ra sản phẩm khi hết chỗ trống
  - Không lấy được sản phẩm khi chưa có



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

#### Producer

```
while(1) {  
    /*produce an item in  
nextProduced*/  
    while (Counter ==  
BUFFER_SIZE) ;  
        /*do nothing*/  
    Buffer[IN] = nextProduced;  
    IN = (IN + 1) % BUFFER_SIZE;  
    Counter++;
```

```
while(1){  
    Consumer  
    while(Counter == 0) ;  
        /*do nothing*/  
    nextConsumed =  
    Buffer[OUT];  
    OUT =(OUT + 1) %  
    BUFFER_SIZE;  
    Counter--; /*consume the  
item in  
nextConsumed*/
```

#### Nhận xét

- Producer sản xuất 1 sản phẩm
- Consumer tiêu thụ 1 sản phẩm  
⇒ Số sản phẩm còn trong Buffer không thay đổi

#### Dùng chung

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
*counter++*  
counter  
Inc R1  
Store  
counter,R1

Load R2,  
*counter--*  
counter  
Dec R2  
Store  
counter,R2

Counter  
++

counter-  
-

R1 = ?



counter = 5

R2 = ?



t

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
*counter*  
Inc R1  
Store  
*counter*,R1

Load R2,  
*counter*  
Dec R2  
Store  
*counter*,R2

**Counter**  
++  
Load R1,counter

**counter-**  
-

R1 = 5



counter = 5

R2 = ?



t

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

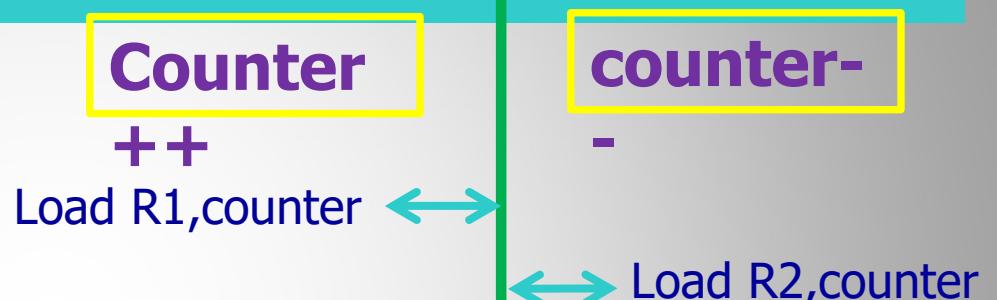
### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
counter  
Inc R1  
Store  
counter,R1

*counter++*

Load R2,  
counter  
Dec R2  
Store  
counter,R2

*counter--*



## Chương 2 Quản lý tiến trình

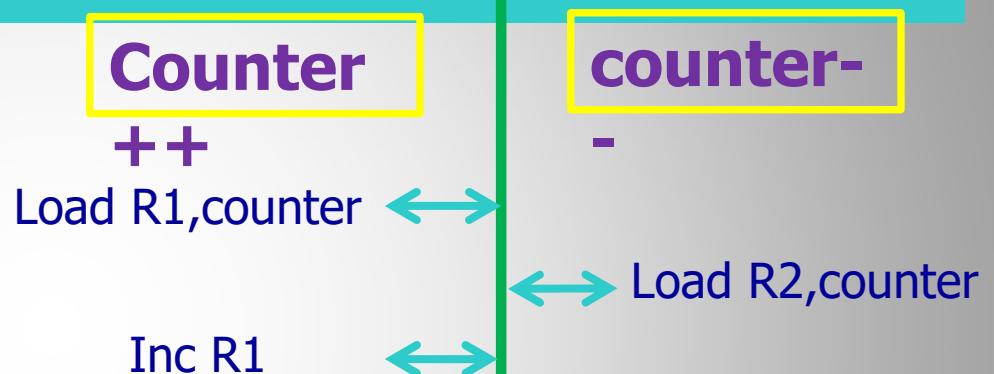
### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
*counter*  
counter++  
Inc R1  
Store  
counter,R1

Load R2,  
*counter*  
Dec R2  
Store  
counter,R2



R1 = 6



counter = 5

R2 = 5



## Chương 2 Quản lý tiến trình

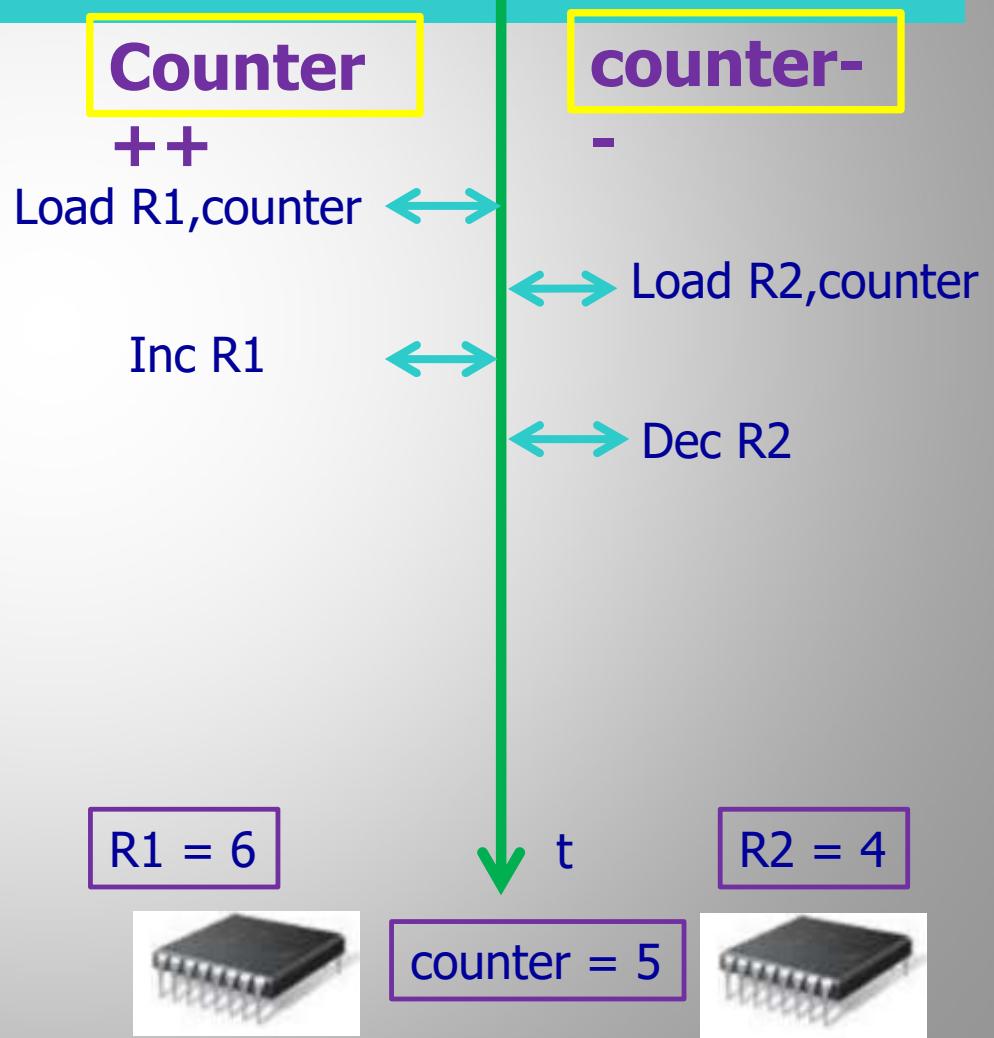
### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

```
Load R1,  
counter  
counter++  
Inc R1  
Store  
counter,R1
```

```
Load R2,  
counter  
counter--  
Dec R2  
Store  
counter,R2
```



## Chương 2 Quản lý tiến trình

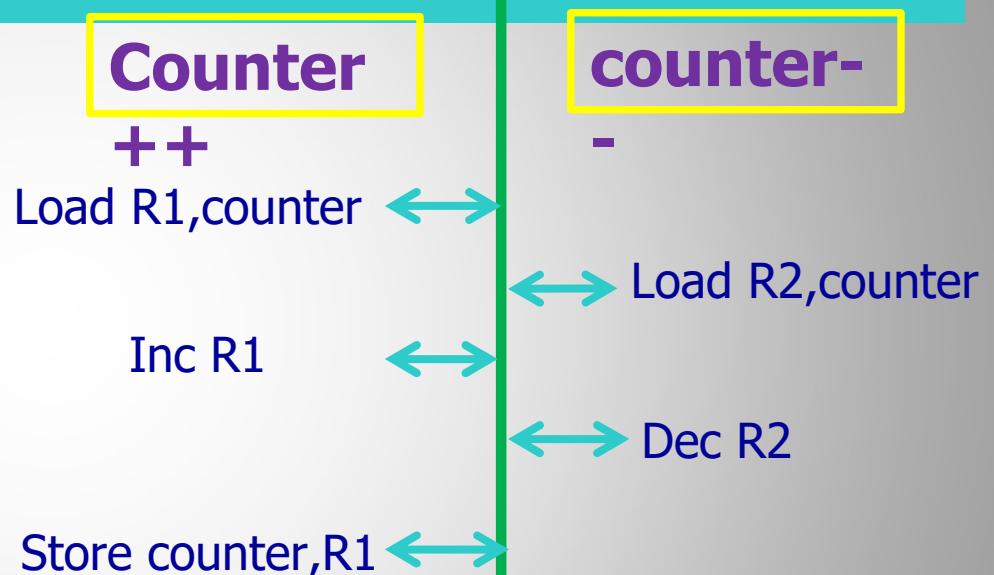
### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
counter  
Inc R1  
Store  
counter,R1

Load R2,  
counter  
Dec R2  
Store  
counter,R2



R1 = 6



counter = 6

R2 = 4



## Chương 2 Quản lý tiến trình

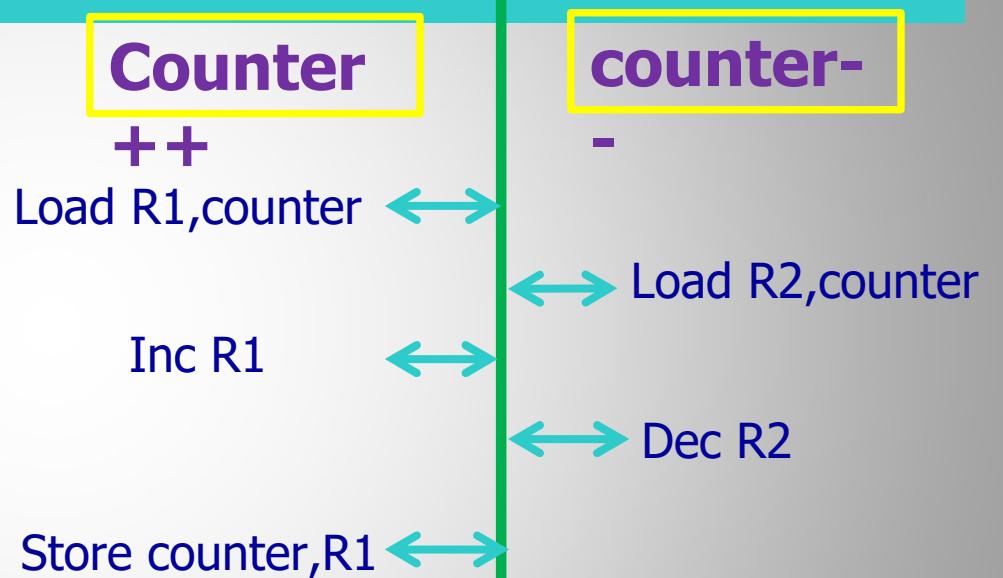
### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer) II

Load R1,  
counter  
Inc R1  
Store  
counter,R1

Load R2,  
counter  
Dec R2  
Store  
counter,R2



R1 = 6



counter = 4

R2 = 4



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Định nghĩa

#### Tài nguyên

Tất cả những gì cần thiết cho thực hiện TT

#### Tài nguyên găng

- Tài nguyên **hạn chế** về khả năng sử dụng chung
- **Cần đồng thời** cho nhiều TT
- Có thể là thiết bị vật lý hay **dữ liệu** dùng chung

#### Vấn đề

Dùng chung tài nguyên găng có thể dẫn đến **không đảm bảo tính toàn vẹn dữ liệu**

⇒ Đòi hỏi cơ chế đồng bộ hóa các TT

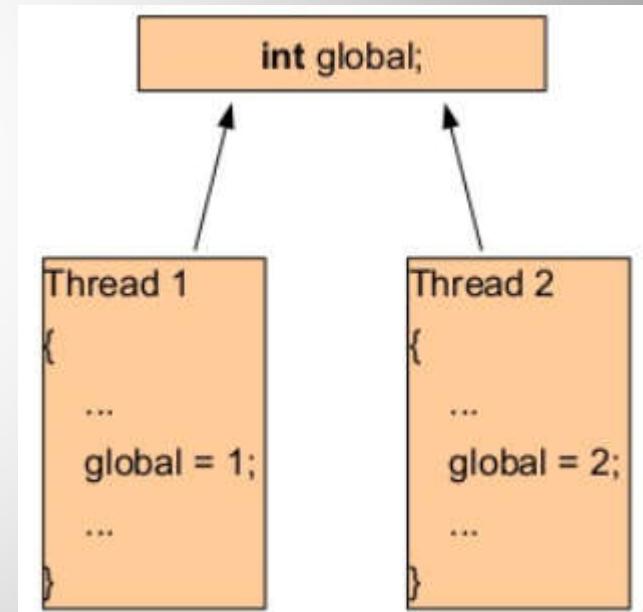
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

## Điều kiện cạnh tranh (Race condition)

- Tình trạng trong đó kết quả của việc nhiều TT cùng truy nhập tới dữ liệu dùng chung phụ thuộc vào trật tự của các truy nhập  
=>Làm cho c/trình không xác định



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

## Điều kiện cạnh tranh (Race condition) II

- Ngăn ngừa điều kiện cạnh tranh = đồng bộ hóa (synchronize) các TT thực hiện đồng thời
  - Chỉ 1 TT truy nhập tới dữ liệu dùng chung tại 1 thời điểm
    - Biến counter trong vấn đề Producer-Consumer
  - Đoạn lệnh truy nhập tới dữ liệu dùng chung trong các TT phải thực hiện theo thứ tự xác định
    - VD: Lệnh  $x \leftarrow y + 1$  trong Thread  $T_1$  chỉ thực hiện khi cả 2 lệnh của Thread  $T_2$  đã thực hiện xong

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Đoạn găng (Critical section)

- Đoạn găng (chỗ hẹp) là đoạn c/trình sử dụng tài nguyên găng
  - Đoạn c/trình thực hiện truy nhập và thao tác trên dữ liệu dùng chung
- Khi có nhiều TT sử dụng tài nguyên găng thì phải điều độ
  - Mục đích: đảm bảo không có quá 1 TT nằm trong đoạn găng

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Yêu cầu của chương trình điều độ

- Loại trừ lẫn nhau (Mutual Exclusion) **Mỗi thời điểm**, tài nguyên găng không phải phục vụ một số lượng TT vượt quá khả năng của nó
  - 1 TT **đang** thực hiện **trong** đoạn găng (*sử dụng tài nguyên găng*)  $\Rightarrow$  Không một TT nào khác được quyền **vào** đoạn găng
- Tiễn triển (Progress) Tài nguyên găng **còn khả năng** phục vụ và tồn tại TT **muốn** vào đoạn găng, thì TT đó phải **được sử dụng** tài nguyên găng
- Chờ đợi hữu hạn (Bounded Waiting) Nếu tài nguyên găng **hết khả năng** phục vụ và vẫn tồn tại TT **muốn** vào đoạn găng, thì TT đó phải **được xếp hàng chờ đợi** và **sự chờ đợi là hữu hạn**

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

## Quy ước

- Có 2 TT *P1&P2* thực hiện đồng thời
- Các TT dùng chung 1 tài nguyên găng
- Mỗi TT đặt **đoạn găng** ở đầu, tiếp theo là phần còn lại
  - TT phải xin phép trước khi vào đoạn găng {*phần vào*}
  - TT khi thoát khỏi đoạn găng thực hiện {*phần ra*}
- Cấu trúc tổng quát của 1 TT

do{

*Phần vào*

    {Đoạn găng của tiến trình}

*Phần ra*

    {Phần còn lại của tiến trình}

}while(1);

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.1 Khái niệm tài nguyên găng

### Phân loại các phương pháp

- Các công cụ cấp thấp
  - Phương pháp khóa trong
  - Phương pháp kiểm tra và xác lập
  - Kỹ thuật đèn báo
- Các công cụ cấp cao
  - Monitor

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

● Khái niệm tài nguyên găng

● Phương pháp khóa trong

● Phương pháp kiểm tra và xác lập

● Kỹ thuật đèn báo

● Ví dụ về đồng bộ tiến trình

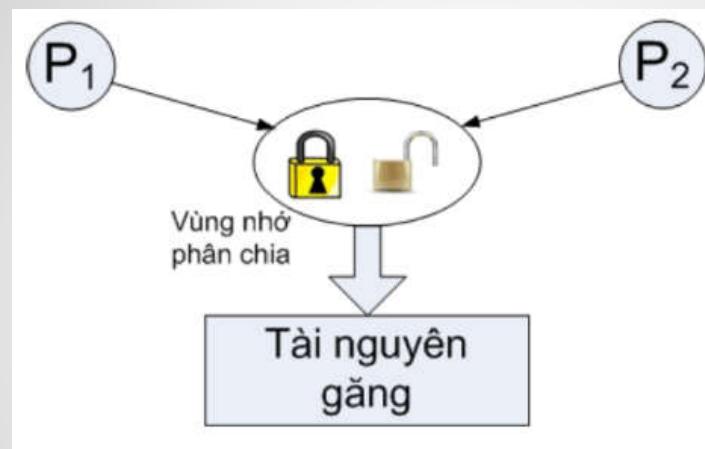
● Công cụ điều độ cấp cao

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

### Nguyên tắc



- Mỗi t/trình dùng 1 byte trong vùng nhớ chung làm khóa
  - TT vào đoạn găng, đóng khóa (byte khóa: true)
  - TT thoát khỏi đoạn găng, mở khóa (byte khóa: false)
- TT muốn vào đoạn găng: kiểm tra khóa của TT còn lại
  - Đang khóa ⇒ Đợi

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

## Thuật toán điều độ

- Share var C1,C2 Boolean // Các biến dùng chung làm khóa
- Khởi tạo C1 = C2 = false // Tài nguyên găng đang tự do

### Tiến trình P1

```
do{
    while( C2 == true );
    C1 ← true;
    {Đoạn găng của tiến trình P1}
    C1 ← false;
    {Phần còn lại của tiến trình P1}
}while(1);
```

### Tiến trình P2

```
do{
    while( C1 == true );
    C2 ← true;
    {Đoạn găng của tiến trình P2}
    C2 ← false;
    {Phần còn lại của tiến trình P2}
}while(1);
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

## Thuật toán điều độ

- Share var C1,C2 Boolean // Các biến dùng chung làm khóa
- Khởi tạo C1 = C2 = false // Tài nguyên găng đang tự do

### Tiến trình P1

```
do{
    C1 ← true;
    while( C2 == true);
    {Đoạn găng của tiến trình P1}
    C1 ← false;
    {Phần còn lại của tiến trình P1}
}while(1);
```

### Tiến trình P2

```
do{
    C2 ← true;
    while( C1 == true);
    {Đoạn găng của tiến trình P2}
    C2 ← false;
    {Phần còn lại của tiến trình P2}
}while(1);
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

## Nhận xét

### ● Điều độ chưa hợp lý

- 2 TT yêu cầu tài nguyên tại cùng 1 thời điểm

- Vấn đề loại trừ lẫn nhau (trường hợp 1)

- Vấn đề tiến triển (trường hợp 2)

### ● Nguyên nhân: Do tách rời giữa

- Kiểm tra quyền vào đoạn găng

- Xác lập quyền sử dụng tài nguyên găng

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

### Thuật toán Dekker

- Sử dụng biến **turn** để chỉ ra TT được quyền ưu tiên

#### Tiến trình P1

```
do{  
    C1 ← true;  
    while(C2==true){  
        if(turn == 2){  
            C1 ← false;  
            while(turn ==2);  
            C1 ← true;  
        }  
    }  
}
```

{Đoạn găng của tiến trình P<sub>1</sub>}

```
turn = 2;  
C1 ← false;
```

{Phần còn lại của tiến trình P<sub>1</sub>}  
}while(1);

#### Tiến trình P2

```
do{  
    C2 ← true;  
    while(C1==true){  
        if(turn == 1){  
            C2 ← false;  
            while(turn ==1);  
            C2 ← true;  
        }  
    }  
}
```

{Đoạn găng của tiến trình P<sub>2</sub>}

```
turn = 1;  
C2 ← false;
```

{Phần còn lại của tiến trình P<sub>2</sub>}  
}while(1);

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

### Nhận xét

- Điều độ hợp lý cho mọi trường hợp
- Không đòi hỏi sự hỗ trợ đặc biệt của phần cứng nên có thể thực hiện bằng ngôn ngữ bất kỳ
- Quá phức tạp khi số TT và số tài nguyên tăng lên

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên gắp và điều độ tiến trình

#### 4.2. Phương pháp khóa trong

### Nhận xét

● Phải chờ đợi tích cực (busy waiting) trước khi vào đoạn gắp

● Khi chờ đợi vẫn phải kiểm tra quyền vào đoạn gắp => Lãng phí thời gian của processor

Ghi chú: Thuật toán có thể thực hiện sai trong một số trường hợp

- CPU cho phép thực hiện các lệnh không đúng trật tự
- Chương trình dịch thực hiện tối ưu hóa khi sinh mã
  - Các mã bắt biến bên trong vòng lặp được đưa ra ngoài

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.3 Phương pháp kiểm tra và xác lập (Test anh Set)

- Khái niệm tài nguyên găng
- Phương pháp khóa trong
- Phương pháp kiểm tra và xác lập
- Kỹ thuật đèn báo
- Ví dụ về đồng bộ tiến trình
- Công cụ điều độ cấp cao

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.3 Phương pháp kiểm tra và xác lập (Test and Set)

### Nguyên tắc

- Sử dụng sự hỗ trợ từ phần cứng
- Phần cứng cung cấp các câu lệnh xử lý không tách rời
  - Kiểm tra và thay đổi nội dung của 1 word
  - Hoán đổi nội dung của 2 word khác nhau
- Xử lý không tách rời (atomically)
  - Khối lệnh không thể bị ngắt trong khi đang thực hiện
  - Được gọi đồng thời, sẽ được thực hiện theo thứ tự bất kỳ

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

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

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.3 Phương pháp kiểm tra và xác lập (Test and Set)

### Thuật toán với lệnh TestAndSet

- Biến dùng chung **Boolean: Lock**: trạng thái của tài nguyên:
  - Bị khóa (*Lock=true*)
  - Tự do (*Lock=false*)
- Khởi tạo: *Lock = false* ⇒ Tài nguyên tự do
- Thuật toán cho TT *Pi*

```
do{  
    while(TestAndSet(Lock));  
    {Đoạn găng của tiến trình}  
    Lock = false;  
    {Phần còn lại của tiến trình}  
}while(1);
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.3 Phương pháp kiểm tra và xác lập (Test and Set)

### Thuật toán với lệnh Swap

- Biến dùng chung **Lock** cho biết trạng thái tài nguyên
- Biến địa phương cho mỗi TT: **Key**: Boolean
- Khởi tạo: *Lock* = *false* ⇒ Tài nguyên tự do
- Thuật toán cho TT *Pi*

**do{**

```
key = true;  
while(key == true)  
    swap(Lock, Key);
```

{Đoạn găng của tiến trình}

Lock = false;

{Phần còn lại của tiến trình}

**}while(1);**

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.3. Phương pháp kiểm tra và xác lập (Test anh Set)

### Nhận xét



Đơn giản, không phức tạp khi số TT và số đoạn găng tăng lên



Các TT phải chờ đợi tích cực trước khi vào đoạn găng



Luôn kiểm tra xem tài nguyên găng đã được giải phóng chưa

⇒ Sử dụng Processor không hiệu quả



Không đảm bảo yêu cầu chờ đợi hữu hạn



TT được vào đoạn găng tiếp theo, sẽ phụ thuộc thời điểm giải phóng tài nguyên của TT đang chiếm giữ

⇒ Cần khắc phục

## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

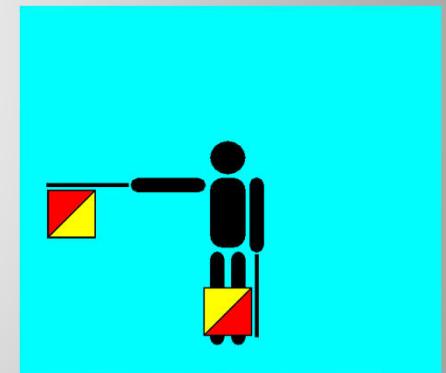
#### 4.3. Phương pháp kiểm tra và xác lập (Test anh Set)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4 Kỹ thuật đèn báo

- Khái niệm tài nguyên găng
- Phương pháp khóa trong
- Phương pháp kiểm tra và xác lập
- **Kỹ thuật đèn báo**
- Ví dụ về đồng bộ tiến trình
- Công cụ điều độ cấp cao



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Đèn báo (Semaphore)

- Là một biến nguyên  $S$ , khởi tạo bằng *khả năng phục vụ* của tài nguyên nó điều độ
  - Số tài nguyên có thể phục vụ tại một thời điểm (*VD 3 máy in*)
  - Số đơn vị tài nguyên có sẵn (*VD 10 chỗ trống trong buffer*)
- Chỉ có thể thay đổi giá trị bởi 2 thao tác cơ bản P và V
  - Thao tác P( $S$ ) (*wait( $S$ )*)

```
wait(S) {  
    while(S ≤ 0) no-op;  
    S --;  
}
```

- Thao tác V( $S$ ) (*signal( $S$ )*)

```
signal(S) {  
    S ++;  
}
```

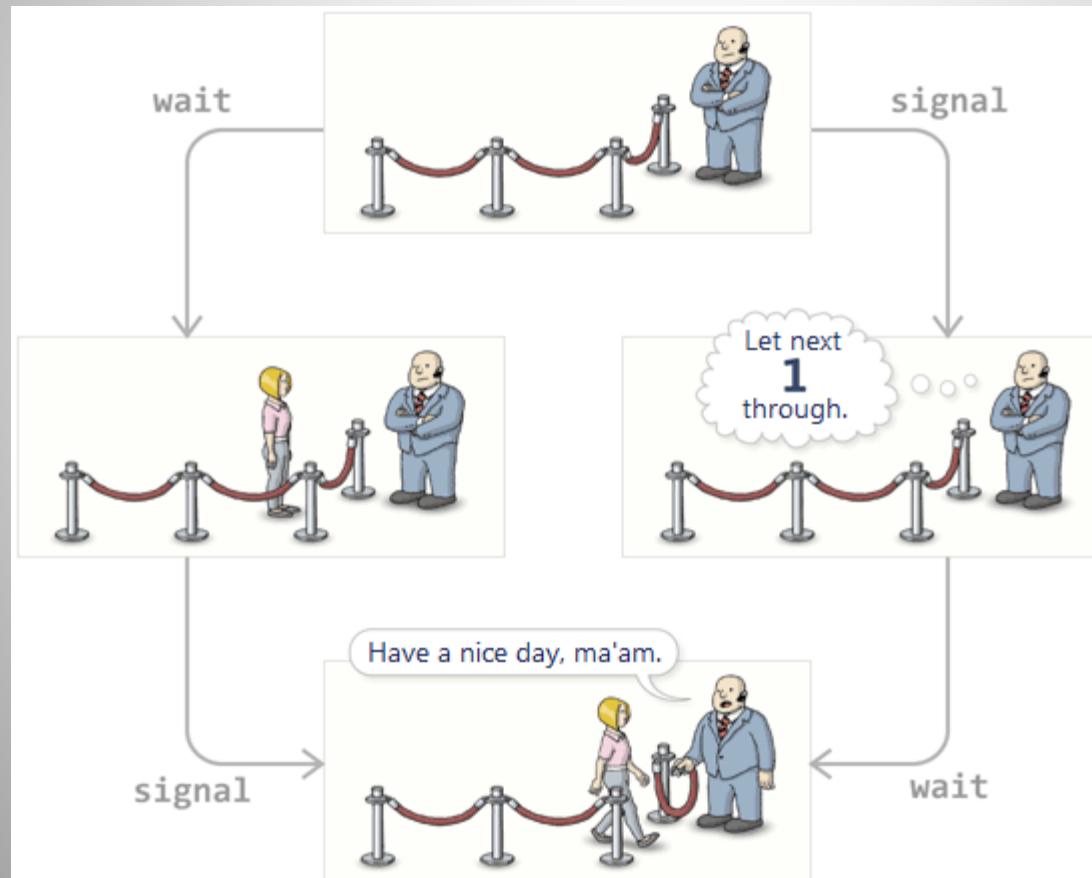
- Các thao tác P( $S$ ) và V( $S$ ) xử lý không tách rời
- Đèn báo là công cụ điều độ tổng quát

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Đèn báo (Semaphore)



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Sử dụng đèn báo I

- Điều độ nhiều tiến trình qua đoạn găng

- Sử dụng biến phân chia mutex kiểu Semaphore
- Khởi tạo mutex bằng 1
- Thuật toán cho tiến trình  $P_i$

```
do{
    wait(mutex);
    {Đoạn găng của tiến trình}
    signal(mutex)
    {Phần còn lại của tiến trình}
}while(1);
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Sử dụng đèn báo II

- Điều độ thứ tự thực hiện bên trong các tiến trình
  - Hai tiến trình  $P_1$  và  $P_2$  thực hiện đồng thời
    - $P_1$  chứa lệnh  $S_1$ ,  $P_2$  chứa lệnh  $S_2$ .
    - Yêu cầu  $S_2$  được thực hiện chỉ khi  $S_1$  thực hiện xong
  - Sử dụng đèn báo *synch* được khởi tạo giá trị 0
  - Đoạn mã cho  $P_1$  và  $P_2$

$P_1$	$P_2$
Phần đầu	Phần đầu
$S_1$	wait(synch)
Signal(synch)	$S_2$
Phần cuối	Phần cuối

## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Hủy bỏ chờ đợi tích cực

- Sử dụng 2 thao tác đơn giản

**block()** Ngừng tạm thời tiến trình đang thực hiện

**wakeup(P)** Thực hiện tiếp t/trình P dừng bởi lệnh *block()*

- Khi tiến trình gọi **P(S)** và đèn báo S không dương

- Tiến trình phải dừng bởi gọi tới câu lệnh *block()*
- Lệnh *block()* đặt tiến trình vào hàng đợi gắn với đèn báo S
- Hệ thống lấy lại CPU giao cho tiến trình khác (*điều phối CPU*)
- Tiến trình chuyển sang trạng thái chờ đợi (*waiting*)
- Tiến trình nằm trong hàng đợi đến khi tiến trình khác thực hiện thao tác **V(S)** trên cùng đèn báo S

- Tiến trình đưa ra lời gọi **V(S)**

- Lấy một tiến trình trong hàng đợi ra (nếu có)
- Chuyển tiến trình lấy ra từ trạng thái chờ đợi sang trạng thái sẵn sàng và đặt lên hàng đợi sẵn sàng bởi gọi tới *wakeup(P)*
- Tiến trình mới sẵn sàng có thể trưng dụng CPU từ tiến trình đang thực hiện nếu thuật toán điều phối CPU cho phép

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Cài đặt đèn báo

### Semaphore S

```
typedef struct{
    int value;
    struct process * Ptr;
}Semaphore;
```

### wait(S)/P(S)

```
void wait(Semaphore S) {
    S.value--;
    if(S.value < 0) {
        Thêm tiến trình vào S.Ptr
        block();
    }
}
```

### signal(S)/V(S)

```
void signal(Semaphore S) {
    S.value++;
    if(S.value ≤ 0) {
        Lấy ra tiến trình P từ S.Ptr
        wakeup(P);
    }
}
```



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Sử dụng đèn báo

# Semaphores

ARELLANO | OLIVA | PENAREDONDO | REYES

---

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

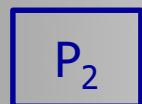
#### 4.4. Kỹ thuật đèn báo

### Ví dụ điều độ

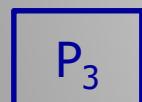
running



running



running



$t$



Semaphore S

$S.Value = 1$

$S.Ptr$

$\rightarrow$  NULL

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

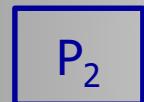
### Ví dụ điều độ

running

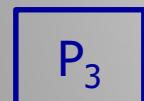


$P_1 \rightarrow P(S)$

running



running



$t$



Semaphore S

$S.Value = 0$

$S.Ptr$

$\rightarrow$  NULL

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

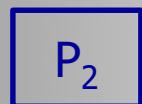
### Ví dụ điều độ

running



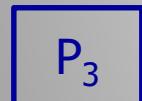
$P_1 \rightarrow P(S)$

block



$P_2 \rightarrow P(S)$

running



$t$



Semaphore S

$S.Value = -1$



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

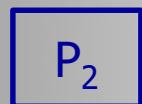
### Ví dụ điều độ

running



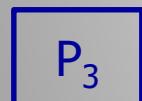
$P_1 \rightarrow P(S)$

block



$P_2 \rightarrow P(S)$

block



$P_3 \rightarrow P(S)$

$t$

Semaphore S

$S.Value = -2$



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Ví dụ điều độ

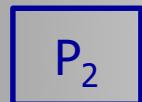
running



$P_1 \rightarrow P(S)$

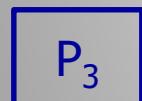
$P_1 \rightarrow V(S)$

running



$P_2 \rightarrow P(S)$

block



$P_3 \rightarrow P(S)$

$t$

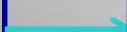


Semaphore S

$S.Value = -1$

$S.Ptr$

PCB3



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Ví dụ điều độ

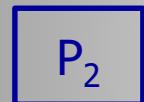
running



$P_1 \rightarrow P(S)$

$P_1 \rightarrow V(S)$

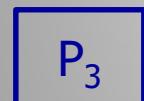
running



$P_2 \rightarrow P(S)$

$P_2 \rightarrow V(S)$

running



$P_3 \rightarrow P(S)$

$t$



Semaphore S

$S.Value = 0$

$S.Ptr$

$\rightarrow$  NULL

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Ví dụ điều độ

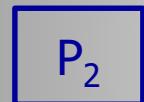
running



$P_1 \rightarrow P(S)$

$P_1 \rightarrow V(S)$

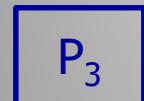
running



$P_2 \rightarrow P(S)$

$P_2 \rightarrow V(S)$

running



$P_3 \rightarrow P(S)$

$P_3 \rightarrow V(S)$

$t$

Semaphore S

$S.Value = 1$

$S.Ptr \rightarrow NULL$

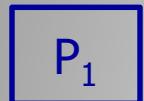
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Ví dụ điều độ

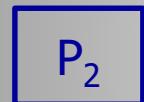
running



$P_1 \rightarrow P(S)$

$P_1 \rightarrow V(S)$

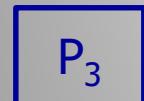
running



$P_2 \rightarrow P(S)$

$P_2 \rightarrow V(S)$

running



$P_3 \rightarrow P(S)$

$P_3 \rightarrow V(S)$

$t$

Semaphore S

$S.Value = 1$

$S.Ptr$

$\rightarrow NULL$

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Nhận xét

- Dễ dàng áp dụng cho các hệ thống phức tạp
- Không tồn tại hiện tượng chờ đợi tích cực
- Hiệu quả sử dụng phụ thuộc vào người dùng

P(S)  
{Đoạn găng}  
V(S)

Điều độ đúng

V(S)  
{Đoạn găng}  
P(S)

Nhầm vị trí

P(S)  
{Đoạn găng}  
P(S)

Nhầm lệnh

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Nhận xét

● Các phép xử lý  $P(S)$  và  $V(S)$  là không phân chia được  
⇒ bản thân  $P(S)$  và  $V(S)$  cũng là 2 tài nguyên găng  
⇒ Cũng cần điều độ.

- Hệ thống một VXL:

- Cấm ngắt khi thực hiện `wait()`, `signal()`

- Hệ thống nhiều VXL:

- Không thể cấm ngắt trên VXL khác

- Có thể dùng phương pháp **khóa trong** ⇒ Hiện tượng chờ đợi tích cực, nhưng thời gian chờ đợi ngắn (10 lệnh)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

## Đối tượng Semaphore trong WIN32 API

- CreateSemaphore( . . . ) : Tạo một Semaphore
  - LPSECURITY\_ATTRIBUTES lpSemaphoreAttributes  
⇒ Trỏ tới cấu trúc an ninh, thẻ trả về được kế thừa?
  - LONG InitialCount, ⇒ Giá trị khởi tạo cho đối tượng Semaphore
  - LONG MaximumCount, ⇒ Giá trị lớn nhất của đối tượng Semaphore
  - LPCTSTR lpName ⇒ Tên của đối tượng Semaphore

Ví dụ `CreateSemaphore(NULL,0,1,NULL);`

- Trả về thẻ (HANDLE) của đối tượng Semaphore hoặc NULL

## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

### Đối tượng Semaphore trong WIN32 API

- **WaitForSingleObject(HANDLE h, DWORD time)**
- **ReleaseSemaphore ( . . . )**
  - HANDLE hSemaphore, ⇐Thẻ của đối tượng Semaphore
  - LONG lReleaseCount, ⇐Giá trị được tăng lên,
  - LPLONG lpPreviousCount ⇐Giá trị trước đó
- **Ví dụ: ReleaseSemaphore(S, 1, NULL);**

## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

#### Ví dụ 1

```
#include <windows.h>
#include <stdio.h>
int x = 0, y = 1;
HANDLE S1, S2;
void T1();
void T2();
int main(){
    HANDLE h1, h2;
    DWORD ThreadId;
    S1 = CreateSemaphore( NULL,0, 1,NULL);
    S2 = CreateSemaphore( NULL,0, 1,NULL);
    h1 = CreateThread(NULL,0,T1, NULL,0,&ThreadId);
    h2 = CreateThread(NULL,0,T2, NULL,0,&ThreadId);
    getch();
    return 0;
}
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.4. Kỹ thuật đèn báo

#### Ví dụ 1

```
void T1(){
    while(1){
        WaitForSingleObject(S1, INFINITE);
        x = y + 1;
        ReleaseSemaphore(S2, 1, NULL);
        printf("%4d", x);
    }
}
void T2(){
    while(1){
        y = 2;
        ReleaseSemaphore(S1, 1, NULL);
        WaitForSingleObject(S2, INFINITE);
        y = 2 * y;
    }
}
```

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

● Khái niệm tài nguyên găng

● Phương pháp khóa trong

● Phương pháp kiểm tra và xác lập

● Kỹ thuật đèn báo

● Ví dụ về đồng bộ tiến trình

● Công cụ điều độ Cấp cao

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên gắt và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

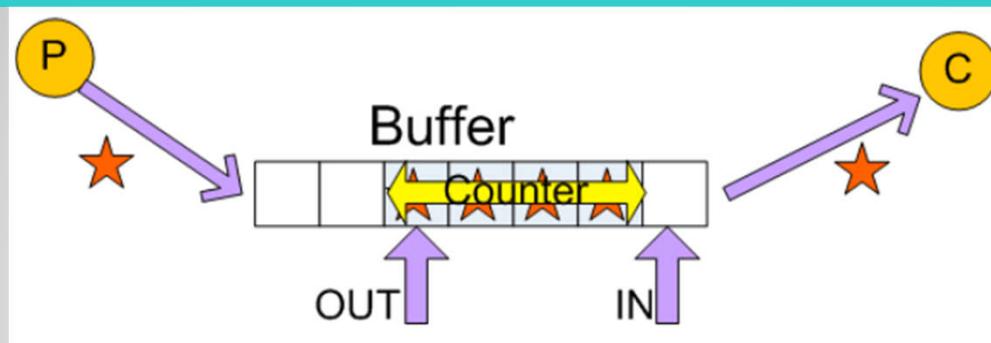
- Người sản xuất-người tiêu thụ (Producer-Consumer)
- Bữa ăn tối của triết gia (Dining Philosophers)
- Người đọc và biên tập viên (Readers-Writers)
- Người thợ cắt tóc ngủ gật (Sleeping Barber)
- Bathroom Problem
- Đồng bộ theo Barriers

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer)



```
while(1) {  
    /*produce an item in Buffer*/  
    while (Counter ==  
BUFFER_SIZE);  
        /*do nothing*/  
    Buffer[IN] = nextProduced;  
    IN = (IN + 1) % BUFFER_SIZE;  
    Counter++;  
}
```

*Producer*

```
while(Counter == 0);  
    /*do nothing*/  
    nextConsumed =  
    Buffer[OUT];  
    OUT = (OUT + 1) %  
    BUFFER_SIZE;  
    Counter--; /*consume the  
item in  
nextConsumed*/  
}
```

*Consumer*

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

Giải pháp: Dùng 1 đèn báo Mutex để điều độ biến

Counter

Khoản trống: Mutex = 1

```
while(1){  
    if(mutex == 1)  
        /*produce a product in Buffer  
     */  
    if(counter == SIZE) block();  
        /*do nothing*/  
    buffer[in] = nextProduced;  
    in = (in + 1) % BUFFER_SIZE;  
    counter++;  
    if(counter == 1) wakeup(Consumer);  
}
```

Producer

while(1){

```
    if(counter == 0) block();  
        /*do nothing*/  
    nextConsumed = buffer[OUT];  
    OUT = (OUT + 1) %  
    BUFFER_SIZE;  
    counter--;  
    if(counter == SIZE - 1)  
        wakeup(Producer);  
    /*consume the item in Buffer*/  
}
```

Consumer

Vấn đề: Giả thiết Counter=0

- Consumer kiểm tra counter => gọi thực hiện lệnh block()
- Producer tăng counter lên 1 và gọi wakeup(Consumer)
- Consumer chưa bị block => Câu lệnh wakeup() bị bỏ qua

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

Giải pháp 2: Dùng 2 đèn báo full, empty được khởi tạo:

full  $\leftarrow 0$  : Số phần tử trong buffer

empty  $\leftarrow \text{BUFFER\_SIZE}$ : Số chỗ trống trong buffer

```
do{
    {Tạo phần tử mới}
    wait(empty);
    {Đặt phần tử mới vào Buffer}
    signal(full);
} while (1);
```

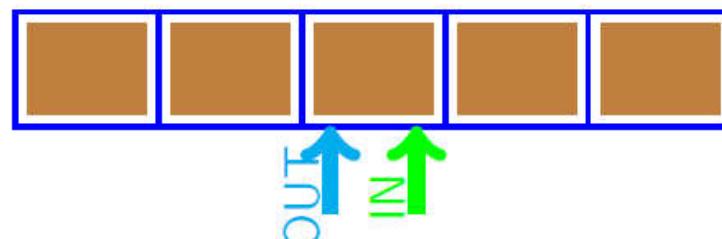
*Producer*

```
do{
    wait(full);
    {Lấy 1 phần tử trong Buffer}
    signal(empty);
    {Xử lý phần tử vừa lấy ra}
} while (1);
```

*Consumer*

**Consumer**  
Running

**Producer**  
Running



empty = 0  
full = 5

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán người sản xuất (producer)-người tiêu thụ(consumer)

**Vấn đề:** Khi có nhiều TT cùng loại Producer và Consumer => biến **IN** và **OUT** → tài nguyên găng

**Giải pháp:** Dùng đèn báo thứ 3 (mutex) để điều độ TT cùng loại

```
do{  
    {Tạo phần tử mới}  
    wait(empty);  
    wait(mutex);  
    {Đặt phần tử mới vào Buffer}  
    signal(mutex);  
    signal(full);  
} while (1);
```

*Producer*

```
do{  
    wait(full);  
    wait (mutex);  
    {Lấy 1 phần tử trong Buffer}  
    signal(mutex);  
    signal(empty);  
    {Xử lý phần tử vừa lấy ra}  
} while (1);
```

*Consumer*

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Người đọc và biên tập viên

- Nhiều TT (Readers) cùng truy nhập 1 cơ sở dữ liệu (CSDL)
- Một số TT (Writers) cập nhật cơ sở dữ liệu
- Cho phép số lượng tùy ý các TT Readers cùng truy nhập CSDL
  - Đang tồn tại 1 TT Reader truy cập CSDL, mọi TT Readers khác mới xuất hiện đều được truy cập CSDL
  - (TT Writers phải xếp hàng chờ đợi)
- Chỉ cho phép 1 TT Writers cập nhật CSDL tại 1 thời điểm.
- Vấn đề không trưng dụng. Các TT ở trong đoạn găng mà không bị ngắt

## Chương 2 Quản lí tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Người thợ cắt tóc ngủ gật (Sleeping Barber)

- Nghế đợi dành cho khách hàng
- Một người thợ chỉ có thể cắt tóc cho một khách hàng tại một thời điểm
- Không có khách hàng đợi, thợ cắt tóc ngủ
- Khi một khách hàng tới
  - Nếu thợ cắt tóc đang ngủ ⇒ Đánh thức anh ta dậy làm việc
  - Nếu thợ cắt tóc đang làm việc
    - Không còn ghế đợi trống ⇒ bỏ đi
    - Còn ghế đợi trống ⇒ Ngồi đợi



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bathroom Problem

#### Bài toán

- A bathroom is to be used by both men and women, but not at the same time
- If the bathroom is empty, then anyone can enter
- If the bathroom is occupied, then only a person of the same gender as the occupant(s) may enter
- The number of people that may be in the bathroom at the same time is limited

- Yêu cầu cài đặt bài toán thỏa mãn các ràng buộc
  - Có 2 kiểu TT male() và female()
  - Mỗi TT ở trong Bathroom một khoảng t/gian ngẫu nhiên

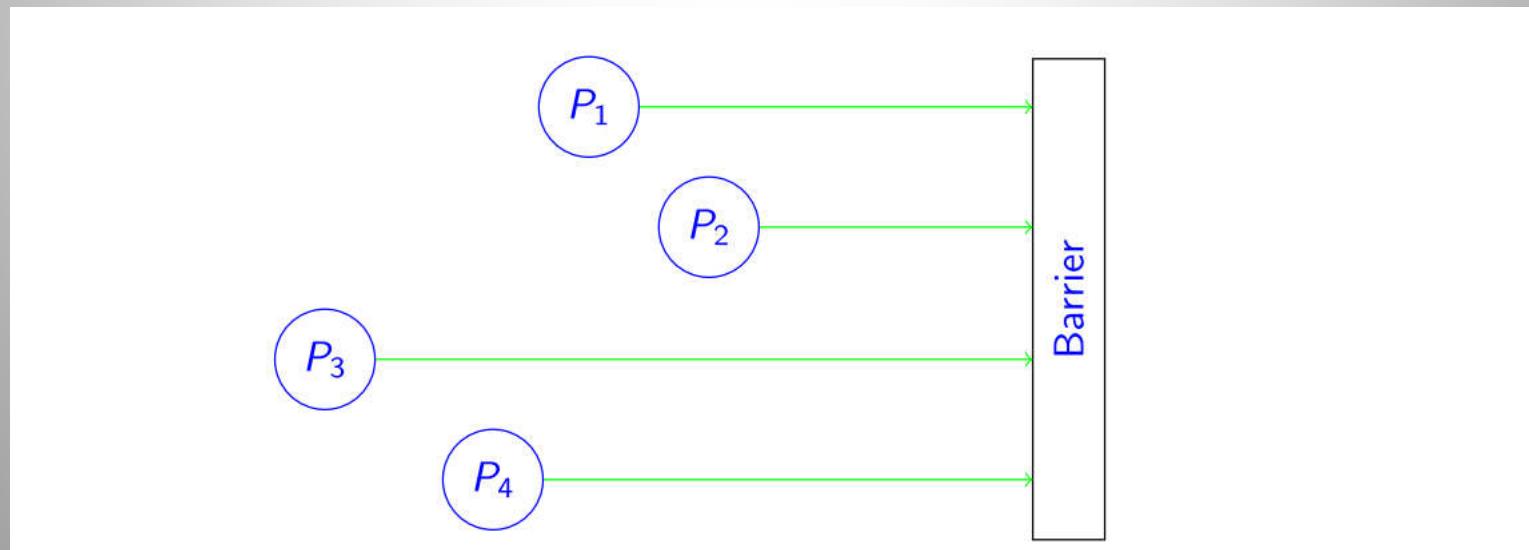
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

## Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung



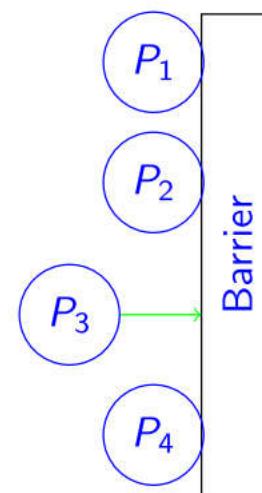
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

## Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung
- Khi đạt tới Ba-ri-e, tất cả các TT đều bị *block* ngoại trừ TT đến cuối cùng



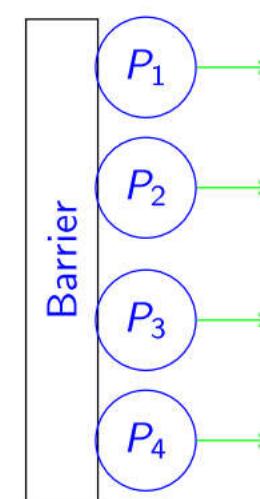
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

## Đồng bộ barriers

- Các TT hướng tới một Ba-ri-e chung
- Khi đạt tới Ba-ri-e, tất cả các TT đều bị *block* ngoại trừ TT đến cuối cùng
- Khi TT cuối tới, đánh thức tất cả các TT đang bị block và cùng vượt qua Ba-ri-e



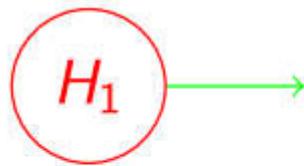
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán tạo phân tử H<sub>2</sub>O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



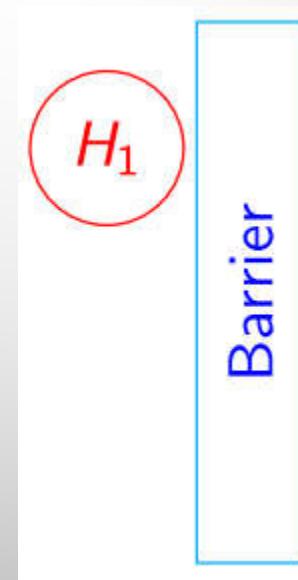
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán tạo phân tử H<sub>2</sub>O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



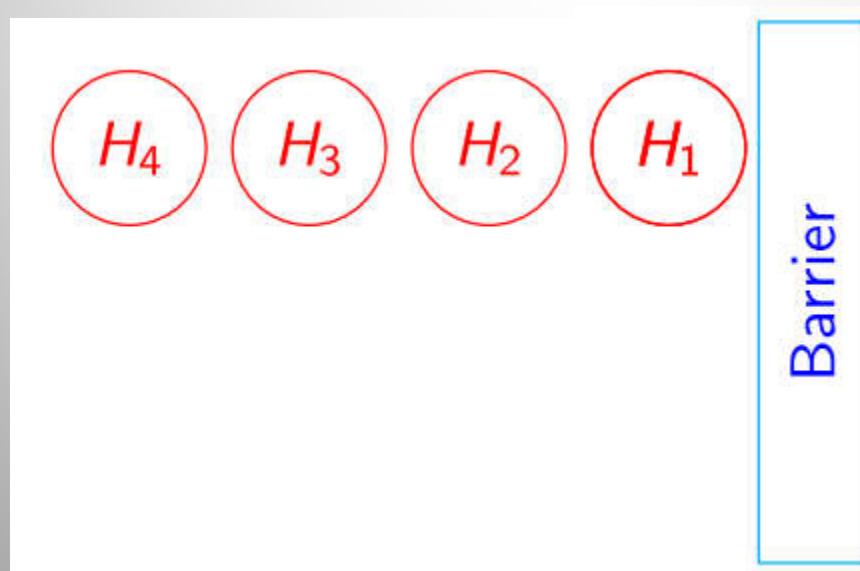
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán tạo phân tử H<sub>2</sub>O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



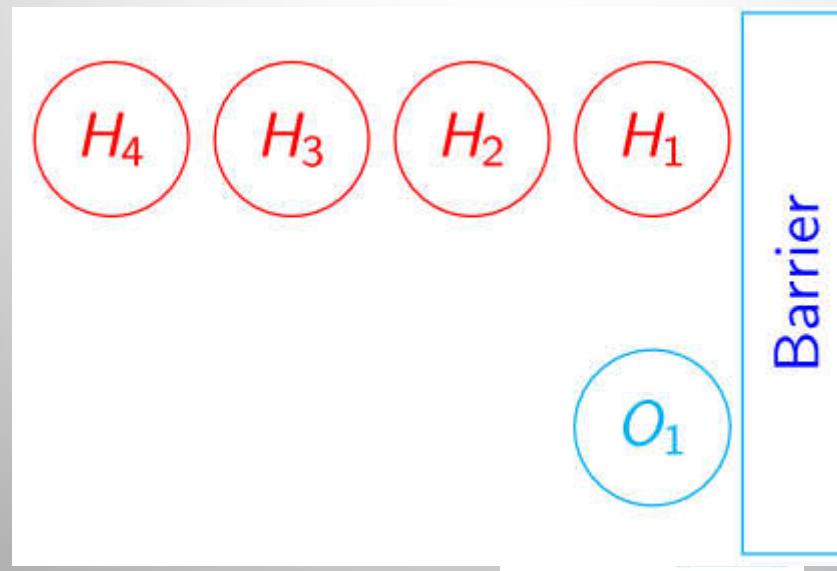
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán tạo phân tử H<sub>2</sub>O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



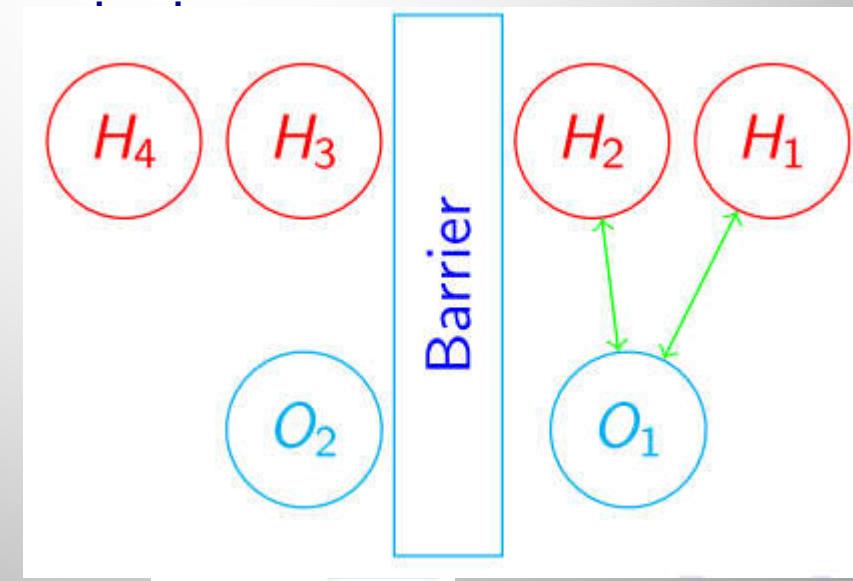
## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bài toán tạo phân tử H<sub>2</sub>O

- Có 2 kiểu TT (luồng): oxygen and hydrogen
- Để kết hợp các TT thành 1 phân tử nước, cần 1 Ba-ri-e để các TT phải đợi cho tới khi 1 phân tử nước sẵn sàng được tạo ra.
- Khi mỗi TT vượt qua Ba-ri-e, nó phải kích hoạt liên kết.
- Tất cả các TT trong cùng một phân tử nước phải tạo liên kết, trước khi 1 TT của phân tử nước khác gọi tới thủ tục tạo liên kết



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Bữa ăn tối của triết gia

*Bài toán đồng bộ hóa tiến trình nổi tiếng, thể hiện tình trạng nhiều tiến trình phân chia nhiều tài nguyên*

- 5 triết gia ăn tối quanh một bàn tròn
- Trước mỗi triết gia là một đĩa mì
- Giữa 2 đĩa kề nhau là một cái dĩa (fork)
- Các triết gia thực hiện luân phiên, liên tục 2 việc: Ăn và Nghỉ
- Mỗi triết gia cần 2 cái dĩa để ăn
- Chỉ lấy 1 dĩa tại 1 thời điểm
- Cái bên trái rồi tới cái bên phải
- Ăn xong, triết gia để dĩa vào vị trí cũ



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên gắp và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Vấn đề triết gia ăn tối: Phương pháp đơn giản

- Mỗi chiếc dĩa là một tài nguyên gắp, được điều độ bởi một đèn báo `fork[i]`
- Semaphore `fork[5] = {1, 1, 1, 1, 1};`
- Thuật toán cho Triết gia Pi

```
do{
    wait(fork[i])
    wait(fork[(i+1)% 5]);
    { Ăn}
    signal(fork[(i+1)% 5]);
    signal(fork[i]);
    { Nghỉ}
} while (1);
```

- Nếu tất cả các triết gia cùng muốn ăn
    - Cùng lấy chiếc dĩa bên trái (gọi tới: `wait(fork[i])`)
    - Cùng đợi lấy chiếc dĩa bên phải (gọi tới: `wait(fork[(i+1)%5])`)
- ⇒ Bế tắc (deadlock)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Vấn đề triết gia ăn tối: Giải pháp 1

- Chỉ cho phép một nhà triết học lấy dĩa tại một thời điểm
- Semaphore mutex  $\leftarrow 1$ ;
- Thuật toán cho Triết gia Pi

```
do{  
    wait(mutex)  
    wait(fork[i])  
    wait(fork[(i+1)% 5]);  
    signal(mutex)  
    { Ăn}  
    signal(fork[(i+1)% 5]);  
    signal(i);  
    { Nghỉ}  
} while (1);
```

- Có thể làm cho 2 triết gia không kề nhau cùng được ăn tại một thời điểm (P1: ăn, P2: chiếm mutex  $\Rightarrow$  P3 đợi)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Vấn đề triết gia ăn tối: Giải pháp 2

- Thứ tự lấy dĩa của các triết gia khác nhau
- Triết gia số hiệu chẵn lấy dĩa trái trước
- Triết gia số hiệu lẻ lấy dĩa phải trước

```
do{
    j = i%2
    wait(fork[(i + j)%5])
    wait(fork[(i+1 - j)% 5]);
    { Ăn}
    signal(fork[(i+1 - j)% 5]);
    signal((i + j)%5);
    { Nghĩ}
} while (1);
```

- Giải quyết được vấn đề bế tắc

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Vấn đề triết gia ăn tối: Một số giải pháp khác

- Trả lại dĩa bên trái nếu không lấy được cái bên phải
  - Kiểm tra dĩa phải sẵn sàng trước khi gọi `wait(fork[(i+1)%5])`
  - Nếu không sẵn có: trả lại dĩa trái, đợi một thời gian rồi thử lại
  - Không bị bế tắc, nhưng không tiến triển: nạn đói (starvation)
  - Thực hiện trong thực tế, nhưng không đảm bảo về lý thuyết của đề bài

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

### Vấn đề triết gia ăn tối: Một số giải pháp khác (tiếp)

- Sử dụng đèn báo đồng thời  $P_{Sim}(S1, S2, \dots, Sn)$ 
  - Thu được tất cả đèn báo cùng một thời điểm hoặc không có bất kỳ đèn báo nào
  - Thao tác  $P_{Sim}(S1, S2, \dots, Sn)$  sẽ block() TT/luồng gọi khi có bất kỳ một đèn báo nào không thể thu được
  - Thuật toán

$P_{Sim}(\text{fork}[i], \text{fork}[(i+1)\% 5]);$

{ Ăn}

$V_{Sim}(\text{fork}[i], \text{fork}[(i+1)\% 5]);$

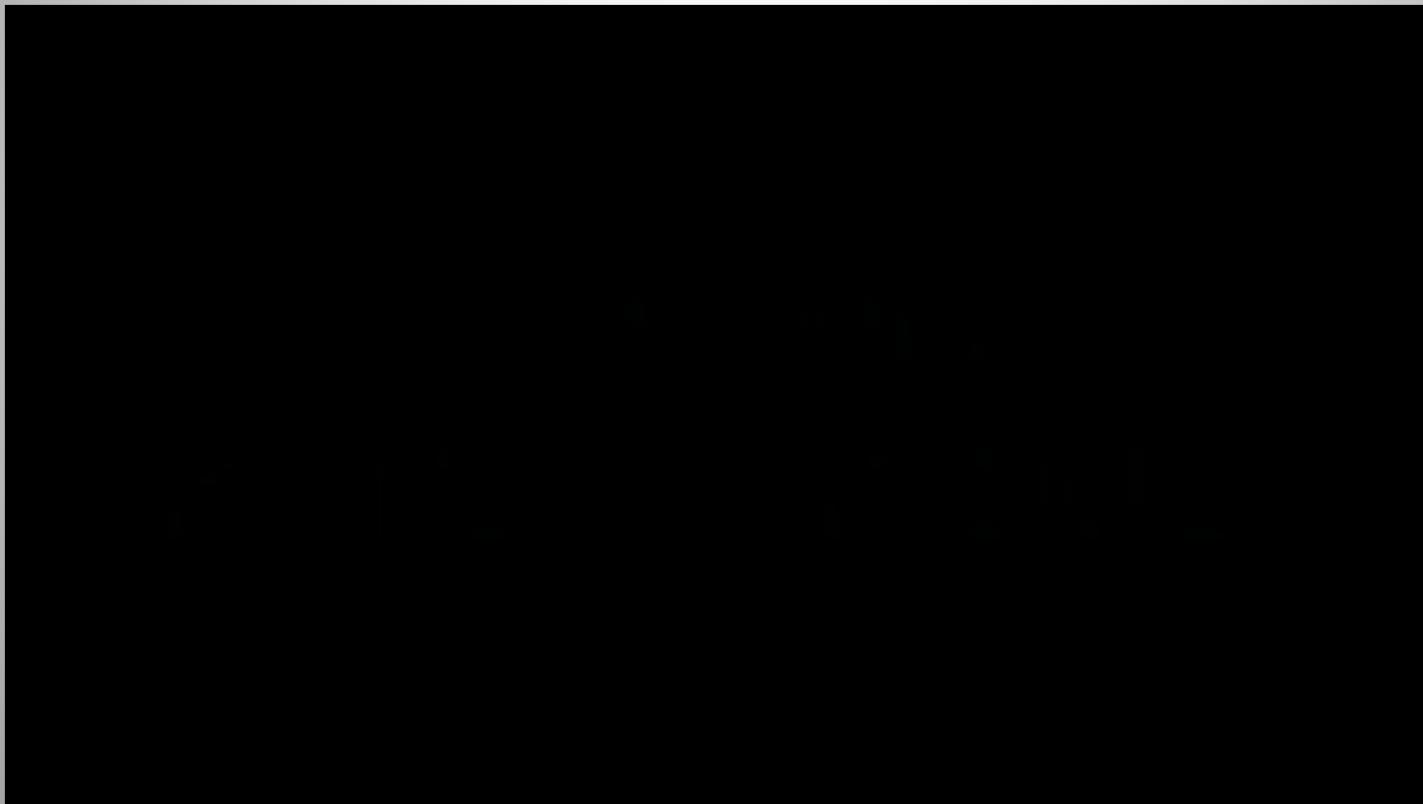
- Khó cài đặt đèn báo đồng thời
- Giải pháp đề xuất bởi Tanenbaum (Tanenbaum 2001)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

Vấn đề triết gia ăn tối: Minh họa

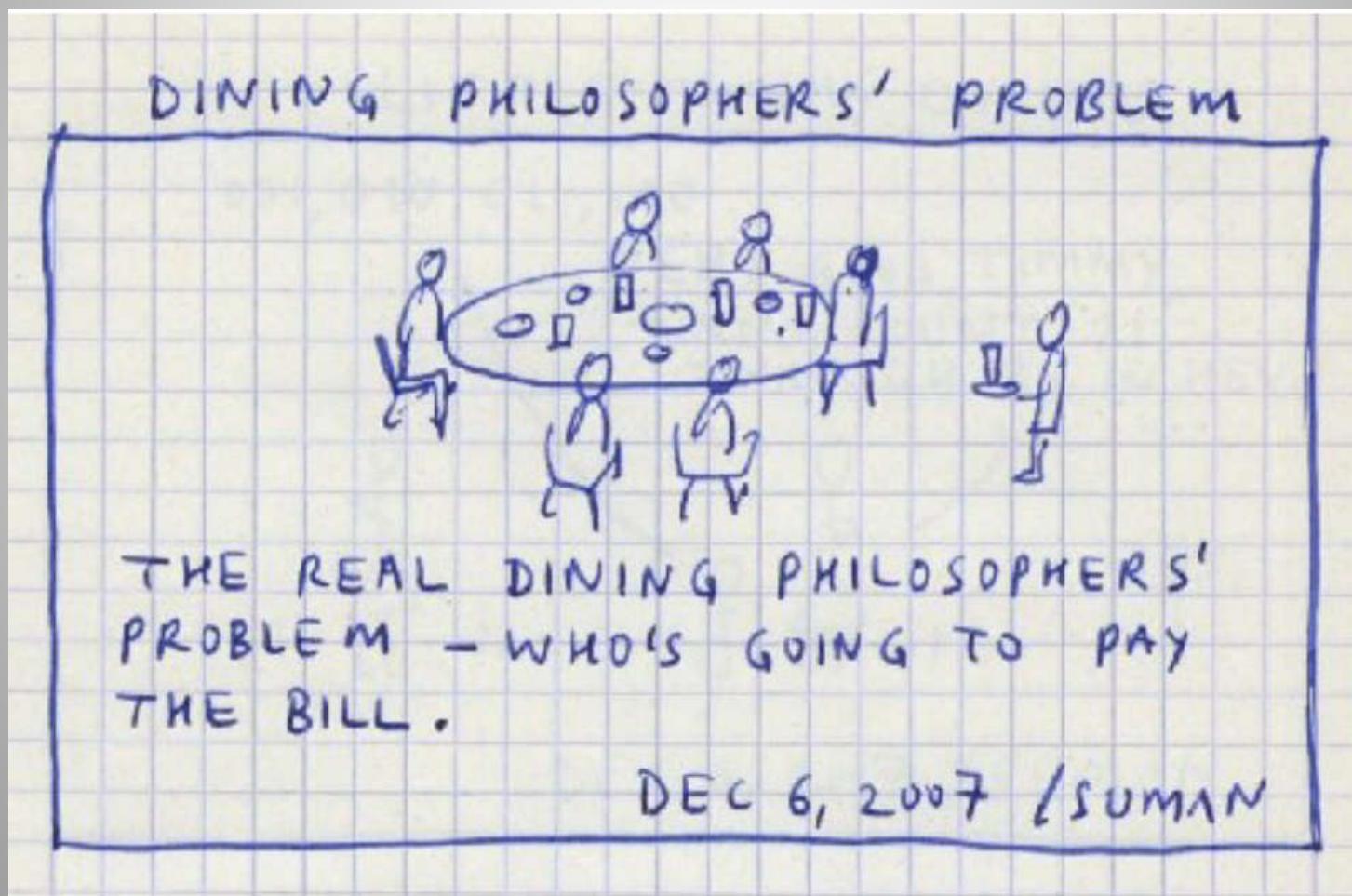


## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.5 Ví dụ về đồng bộ tiến trình

True problem ?



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

- Khái niệm tài nguyên găng
- Phương pháp khóa trong
- Phương pháp kiểm tra và xác lập
- Kỹ thuật đèn báo
- Ví dụ về đồng bộ tiến trình
- Công cụ điều độ cấp cao

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình 4.6 Công cụ điều độ cấp cao

#### Giới thiệu

- Là 1 kiểu dữ liệu đặc biệt, được đề xuất bởi HOARE 1974
- Bao gồm các thủ tục, dữ liệu cục bộ, đoạn mã khởi tạo
- Các TT chỉ có thể truy nhập tới các biến bằng cách gọi tới các thủ tục trong Monitor
- Tại 1 thời điểm chỉ có 1 TT được quyền sử dụng Monitor
  - TT khác muốn sử dụng, phải chờ đợi
- Cho phép các TT đợi trong Monitor
  - Sử dụng các biến điều kiện (condition variable)

```
monitor monitorName{  
    Khai báo các biến dùng chung  
    procedure P1(...){  
        ...  
    }  
    ...  
    procedure Pn(...){  
        ...  
    }  
    {  
        Mã khởi tạo  
    }  
};
```

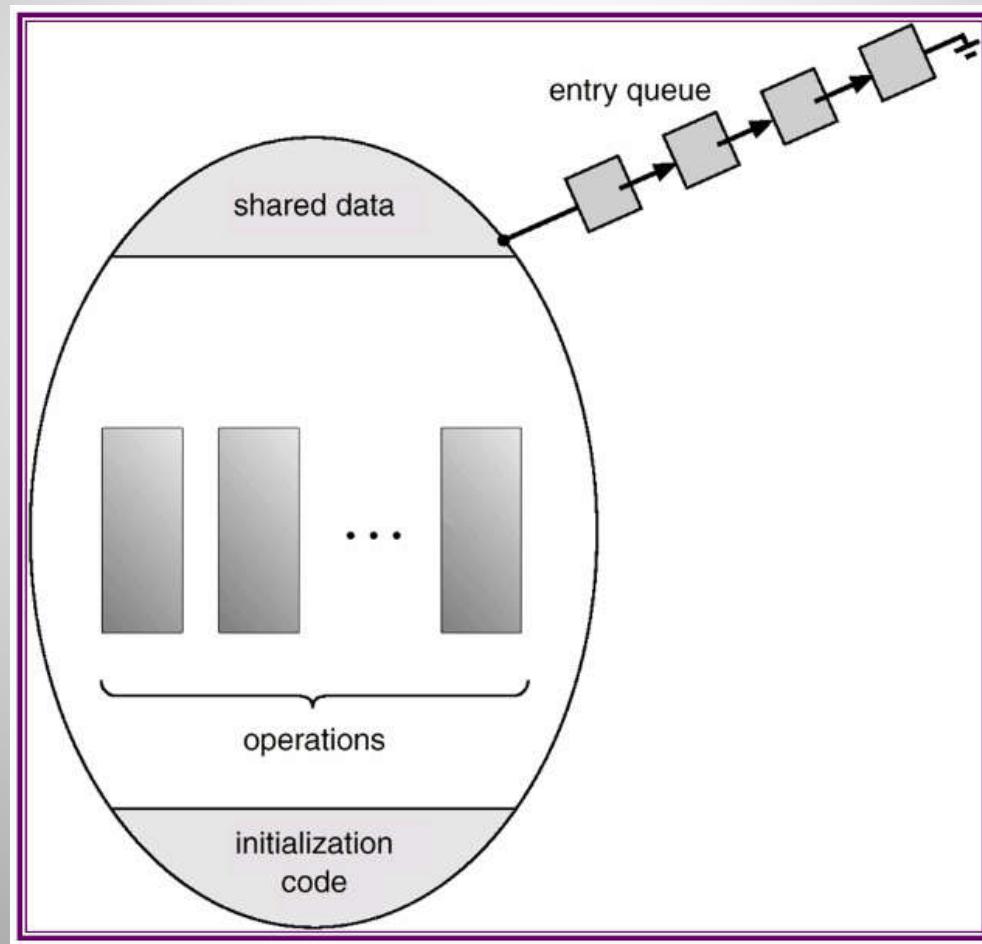
Cú pháp của Monitor

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

## Mô hình



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

#### Biến điều kiện

- Thực chất là tên của 1 hàng đợi
- Khai báo: **condition** x,y;
- Các biến điều khiển chỉ có thể được sử dụng với 2 thao tác `wait()` và `signal()`

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

#### Biến điều kiện

#### wait()

- Được gọi bởi các thủ tục của Monitor
- **Cú pháp:**  $x.wait()$  hoặc  $wait(x)$
- cho phép TT đưa ra lời gọi bị tạm dừng (block)
  - cho tới khi được 1 TT khác kích hoạt bằng cách gọi tới  $signal()$

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

#### Biến điều kiện

#### signal()

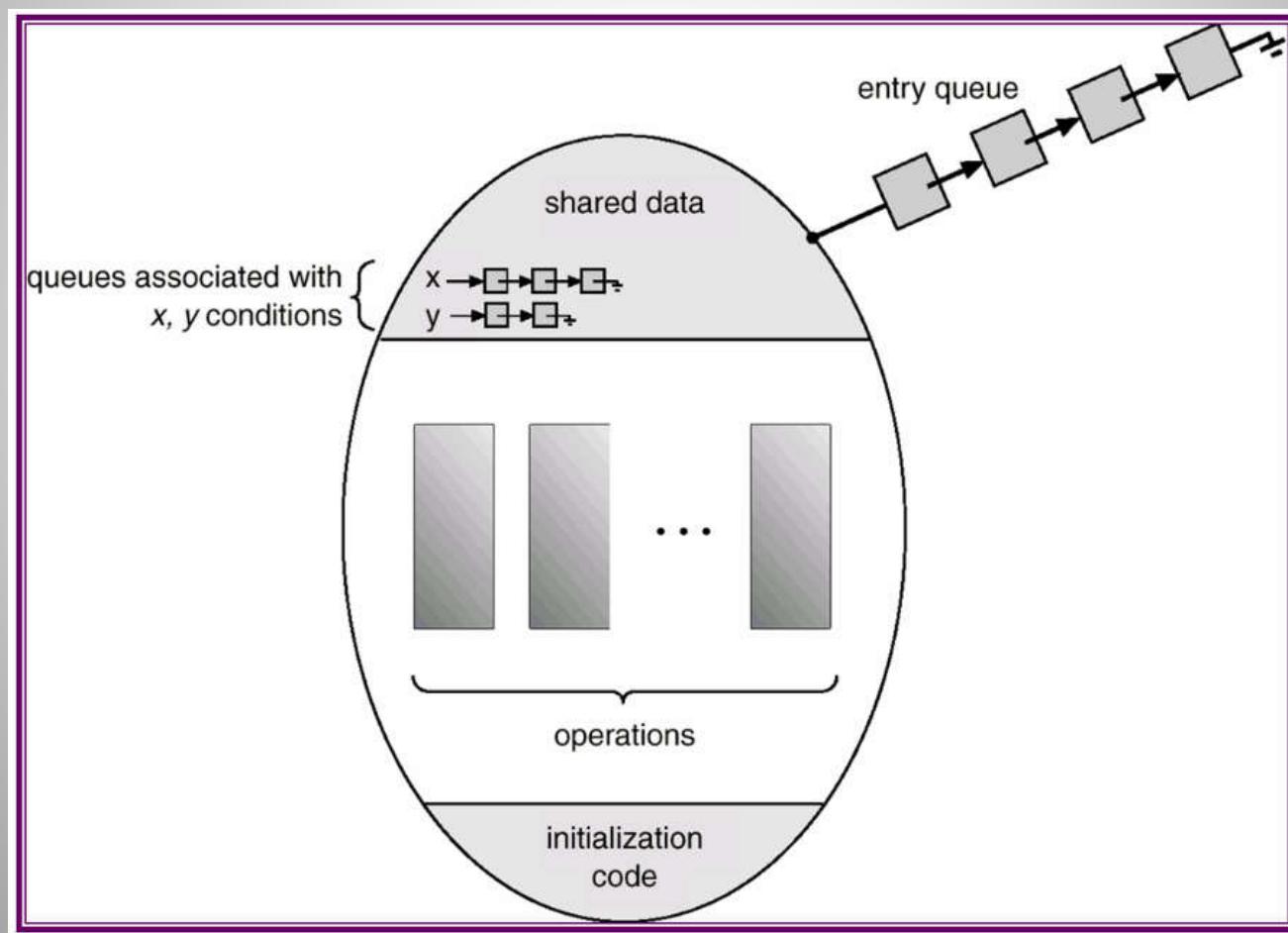
- Được gọi bởi các thủ tục của Monitor
- (**Cú pháp:** *x.signal()* hoặc *signal(x)*)
- kích hoạt chính xác 1 TT đang đợi tại biến điều kiện *x* (nằm trong hàng đợi *x*) ra tiếp tục hoạt động.
  - Nếu không có TT nào đang đợi, thao tác không có hiệu lực (bị bỏ qua)

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

## Mô hình



## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

## Sử dụng Monitor: một tài nguyên chung

```
Monitor Resource{
    Condition Nonbusy;
    Boolean Busy
    //--- Phần dành người dùng ---
    void Acquire(){
        if(busy) Nonbusy.wait();
        busy=true;
    }
    void Release(){
        busy=false
        signal(Nonbusy)
    }
    //---- Phần khởi tạo ----
    busy= false;
    Nonbusy = Empty;
}
```

### Cấu trúc tiến trình

**while(1){**

...

**Resource.Acquire()**

**{Sử dụng tài nguyên}**

**Resource.Release()**

...

**}**

## Chương 2 Quản lý tiến trình

### 4. Tài nguyên găng và điều độ tiến trình

#### 4.6 Công cụ điều độ cấp cao

## Bài toán Producer - Consumer

```
Monitor ProducerConsumer{
    Condition Full, Empty;
    int Counter ;
    void Put(Item){
        if(Counter=N) Full.wait();
        {Đặt Item vào Buffer};
        Counter++;
        if(Counter=1)Empty.signal()
    }
    void Get(Item){
        if(Counter=0) Empty.wait()
        {Lấy Item từ Buffer};
        Counter--;
        if(Counter=N-1)Full.signal()
    }
    Counter=0;
    Full, Empty = Empty;
}
```

ProducerConsumer M;

### Producer

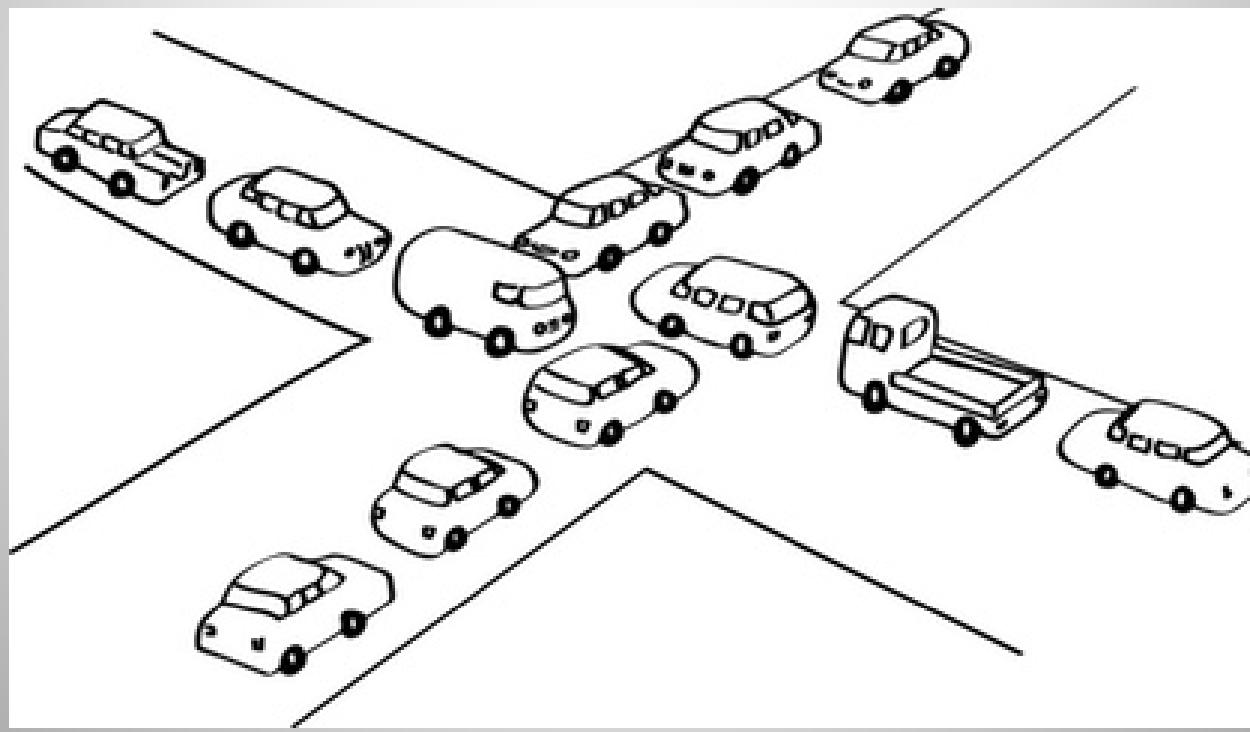
```
while(1){
    Item = Sản phẩm mới
    M.Put(Item)
    ...
}
```

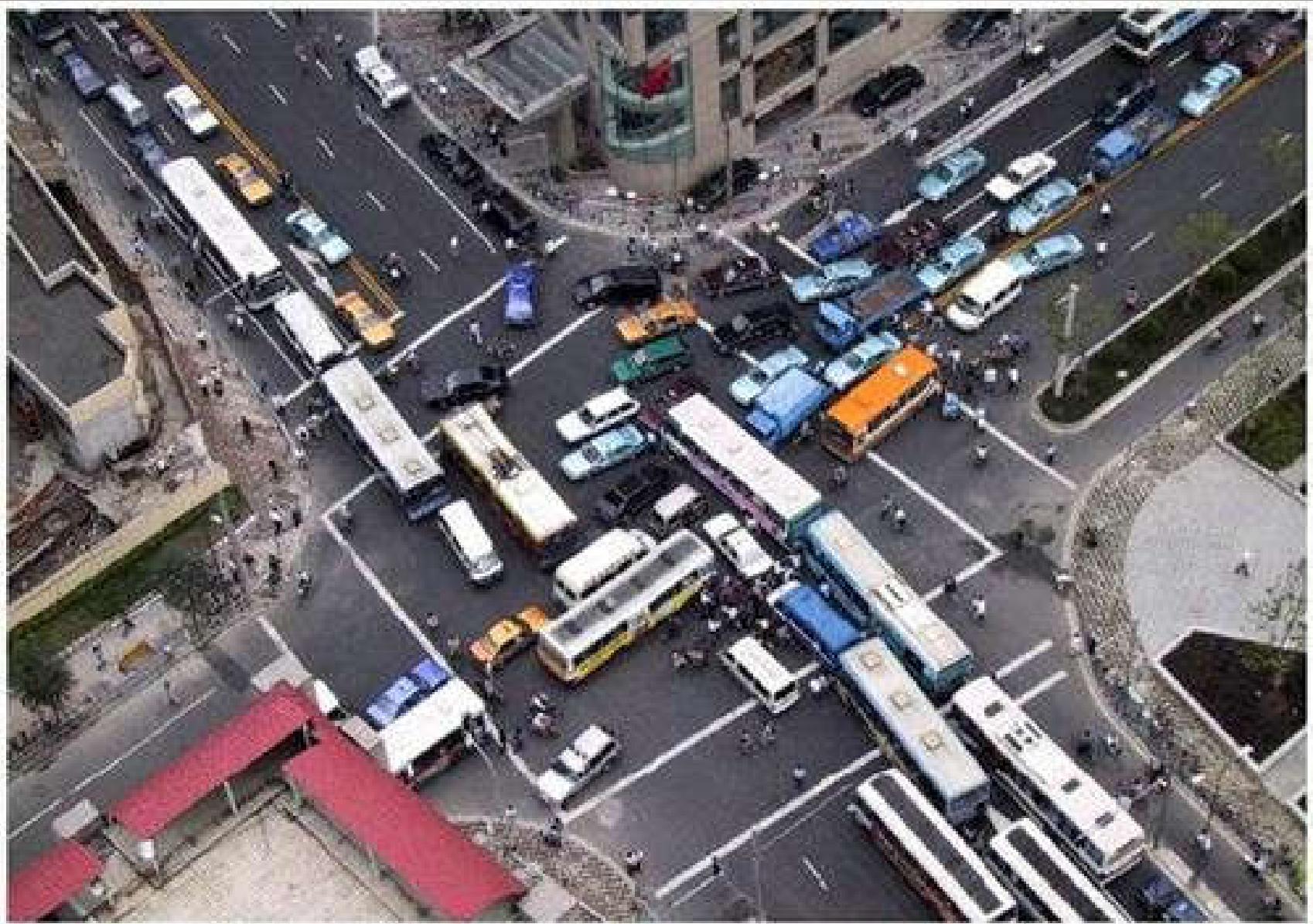
### Consumer

```
while(1){
    M.Get(&Item)
    {Sử dụng Msg}
    ...
}
```

## Chương 2 Quản lý tiến trình

- ① Tiến trình
- ② Luồng (Thread)
- ③ Điều phối CPU
- ④ Tài nguyên găng và điều độ tiến trình
- ⑤ Bẽ tắc và xử lý bẽ tắc





## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- Các phương pháp xử lý bẽ tắc
- Phòng ngừa bẽ tắc
- Phòng tránh bẽ tắc
- Nhận biết và khắc phục

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

##### Khái niệm bẽ tắc

- Hệ thống gồm nhiều TT hoạt động đồng thời cùng sử dụng tài nguyên (TN)
  - TN có **nhiều loại** (VD: CPU, bộ nhớ,...).
  - Mỗi loại TN có **nhiều đơn vị** (VD: 2 CPU, 5 máy in..)
- Mỗi TT thường gồm **dãy** liên tục **các thao tác**
  - **Đòi hỏi tài nguyên**: Nếu TN không có sẵn (đang được s/dụng bởi TT khác)  $\Rightarrow$  TT yêu cầu phải đợi
  - **Sử dụng TN** theo yêu cầu (in ấn, đọc dữ liệu...)
  - **Giải phóng TN** được cấp
- Khi các TT dùng chung ít nhất **2 TN**, hệ thống có thể gặp "**nguy hiểm**"

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

##### Khái niệm bẽ tắc

- Xét ví dụ: Hệ thống có 2 TT  $P_1$  &  $P_2$ 
  - 2 TT  $P_1$  &  $P_2$  dùng chung 2 TN  $R_1$  &  $R_2$
  - $R_1$  được điều độ bởi đèn báo  $S_1$  ( $S_1 \leftarrow 1$ )
  - $R_2$  được điều độ bởi đèn báo  $S_2$  ( $S_2 \leftarrow 1$ )
  - Đoạn mã cho  $P_1$  và  $P_2$

```
P(S1)
P(S2)
{ Sử dụng R1&R2 }
V(S1)
V(S2)
```

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

*Process P1*

P( $S_1$ )  
P( $S_2$ )  
{ Sử dụng  $R_1 \& R_2$  }  
V( $S_1$ )  
V( $S_2$ )

*Process P1*

*Process P2*

*Process P2*

P( $S_1$ )  
P( $S_2$ )  
{ Sử dụng  $R_1 \& R_2$  }  
V( $S_1$ )  
V( $S_2$ )



$S1 = 1$



$S2 = 1$

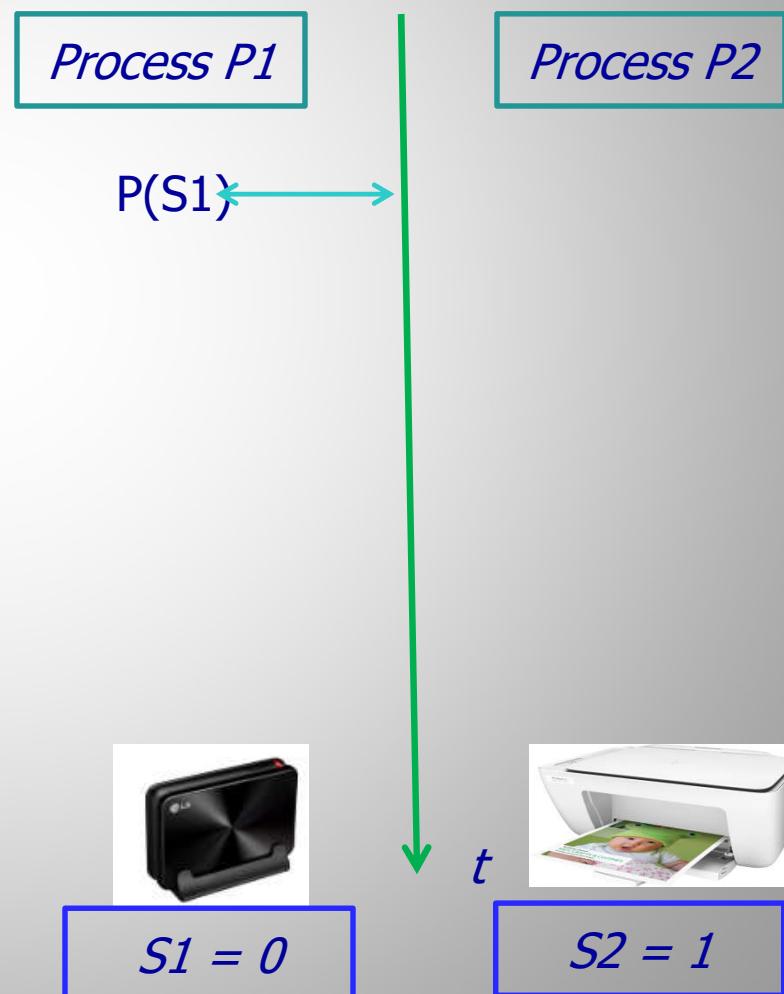
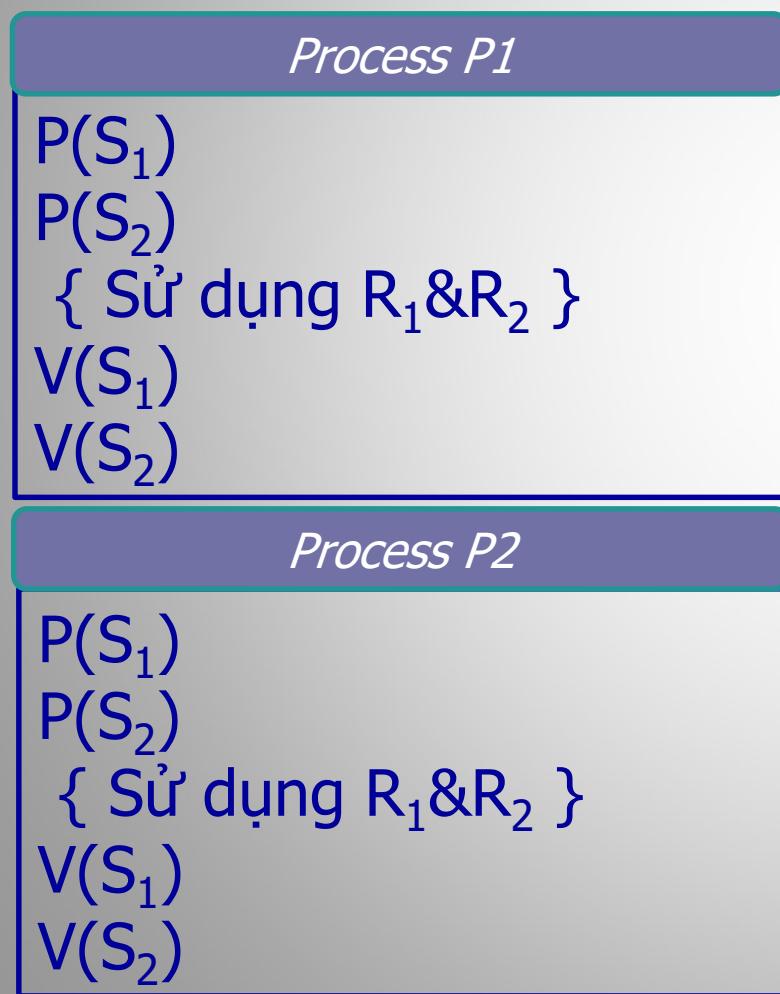
$t$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

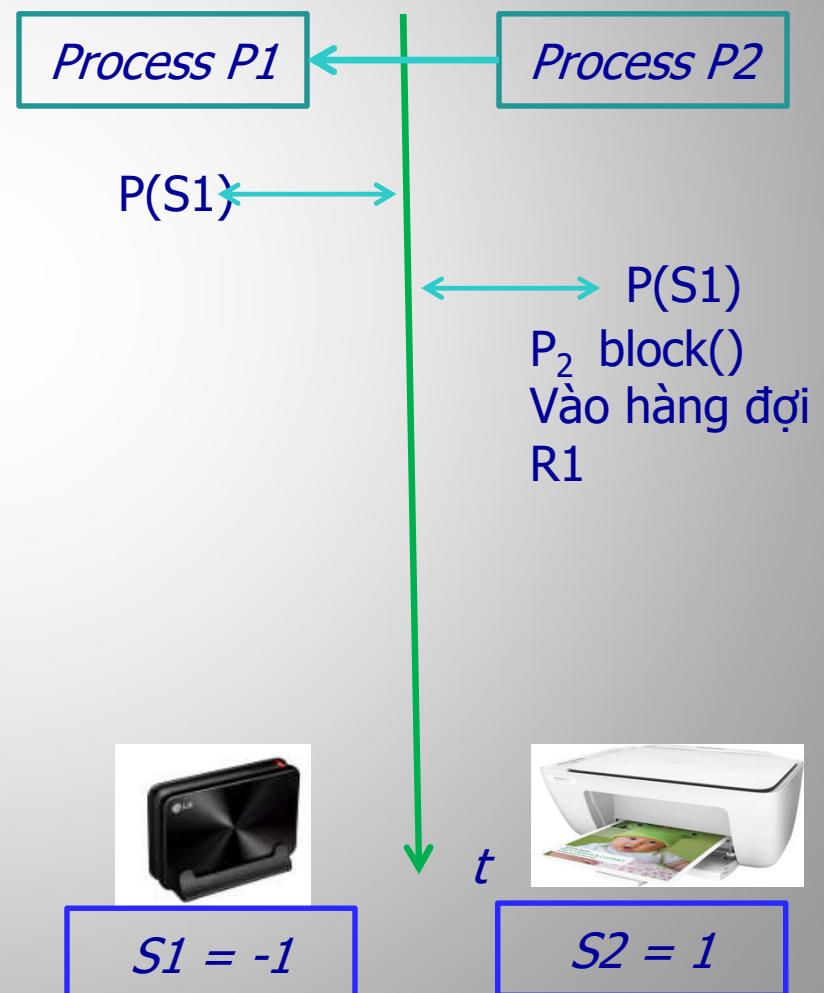
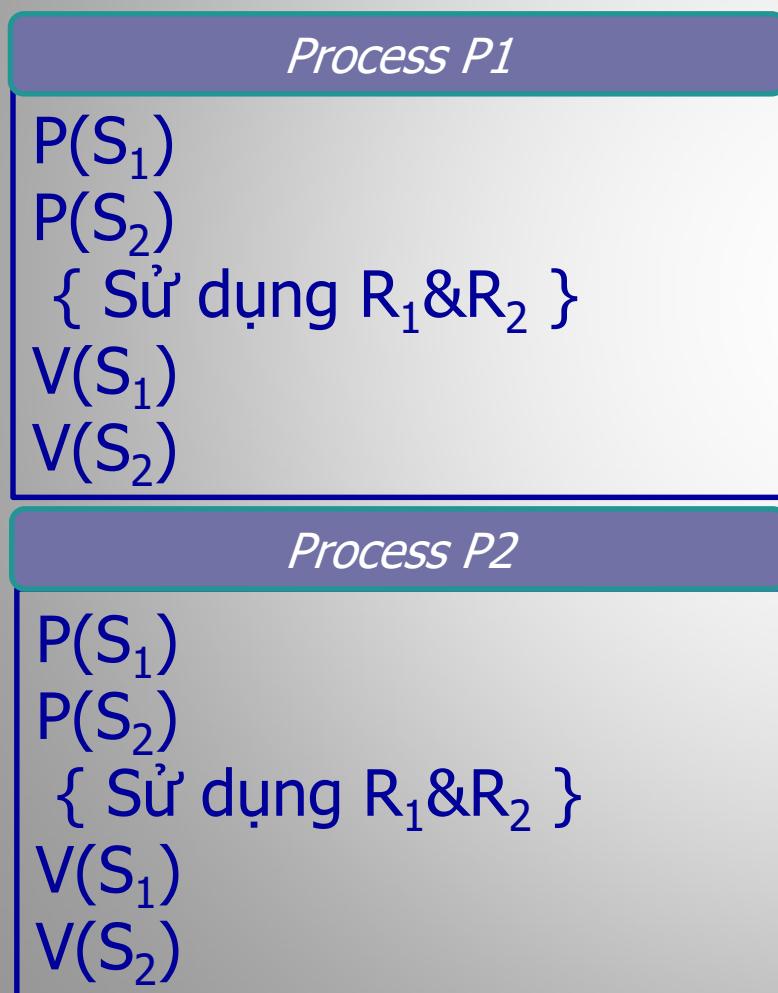


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

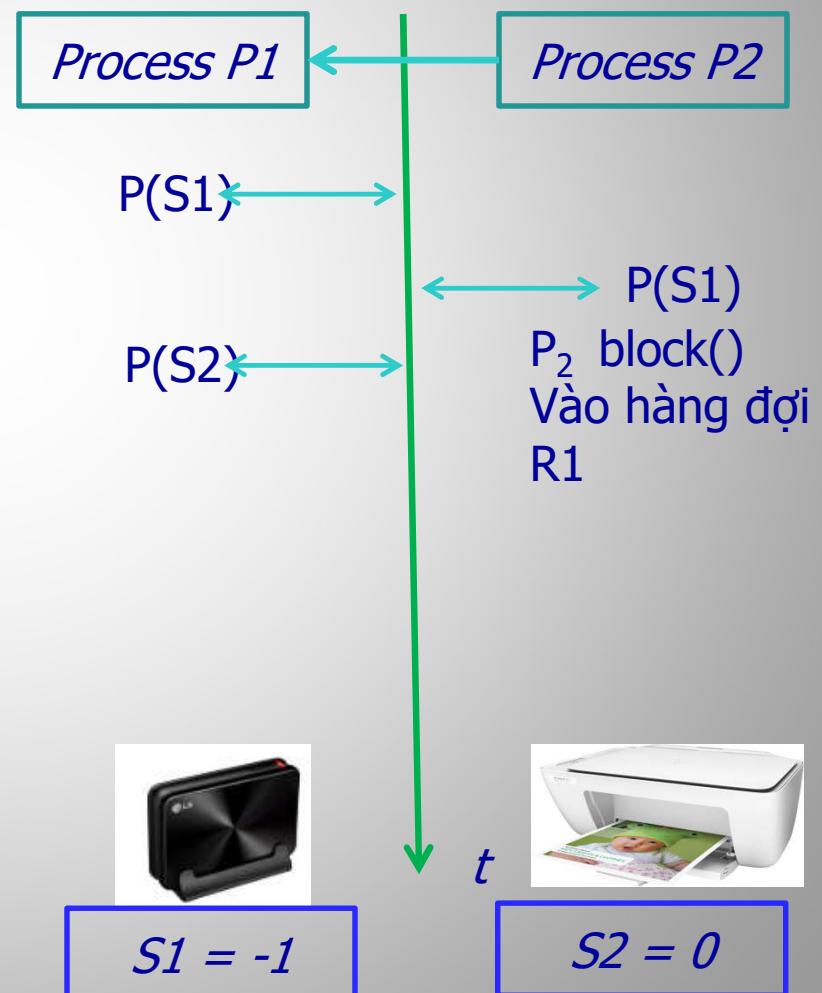
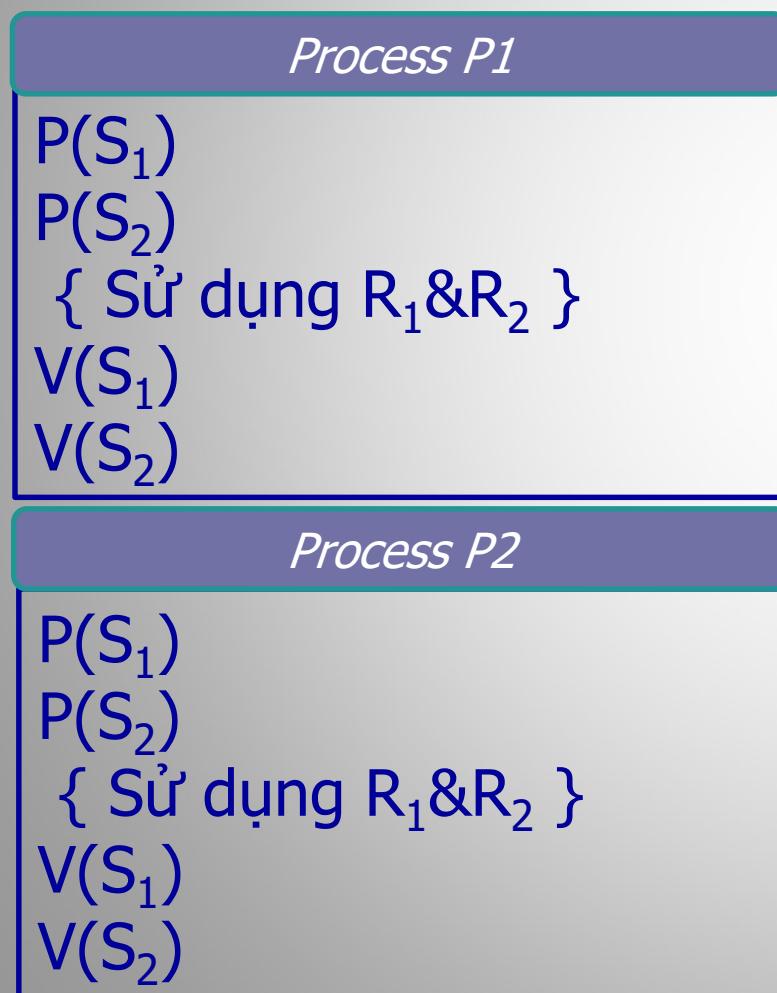


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

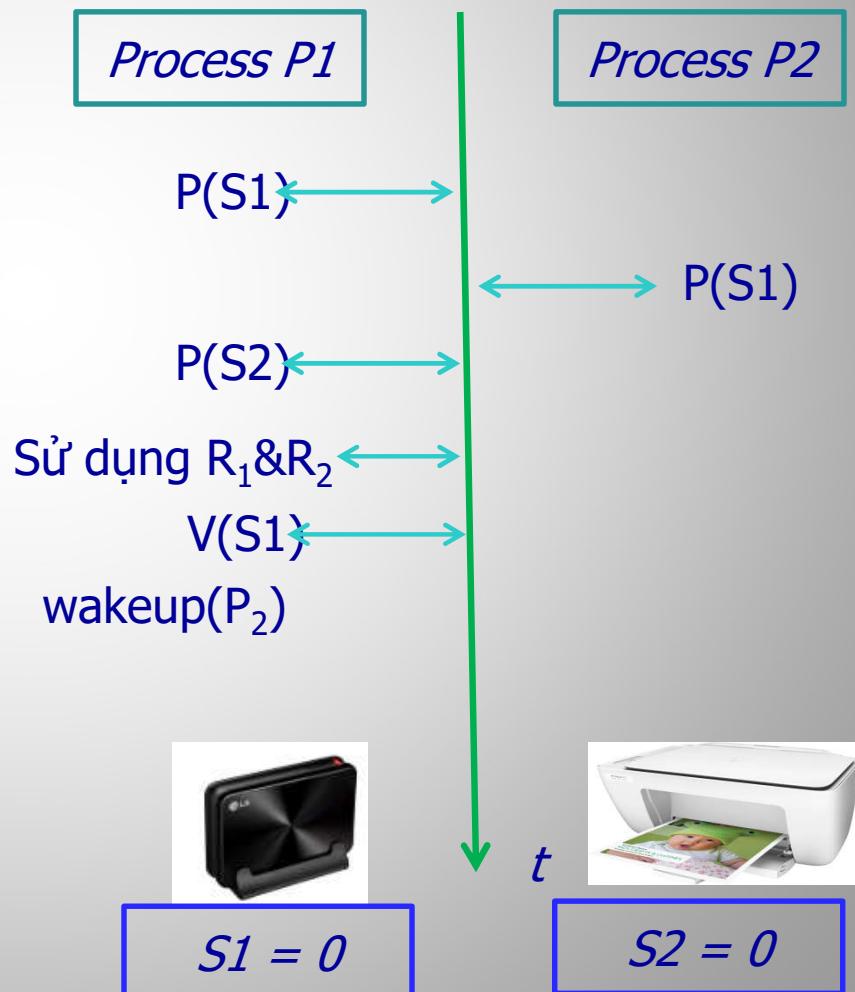
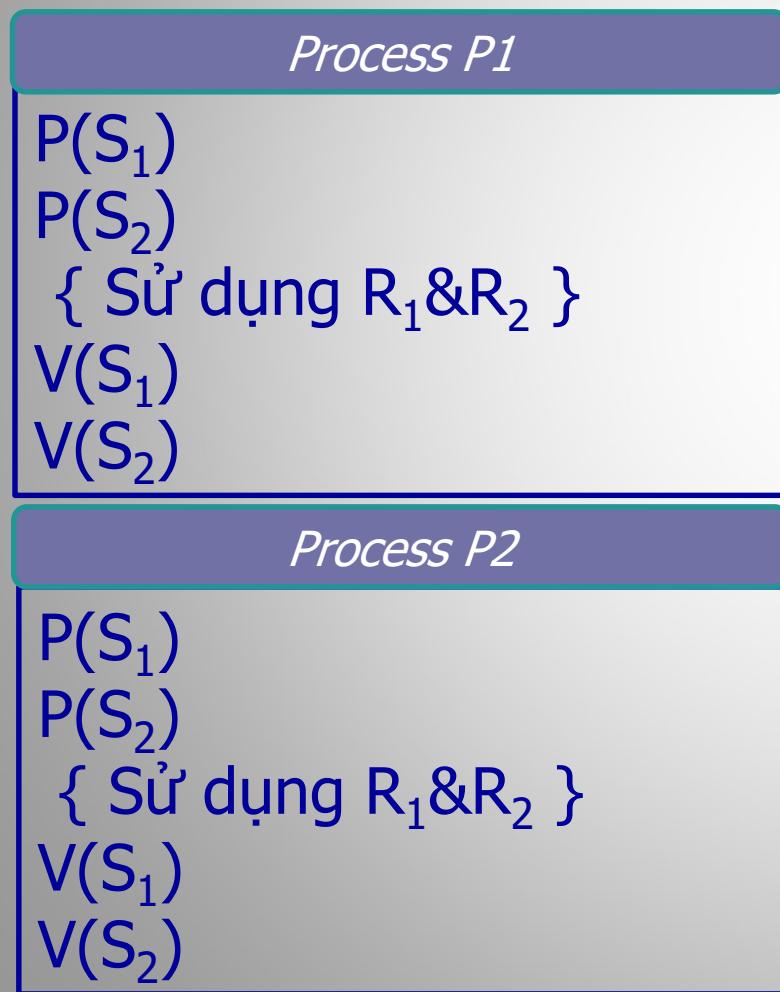


## Chương 2 Quản lý tiến trình

### 5. Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

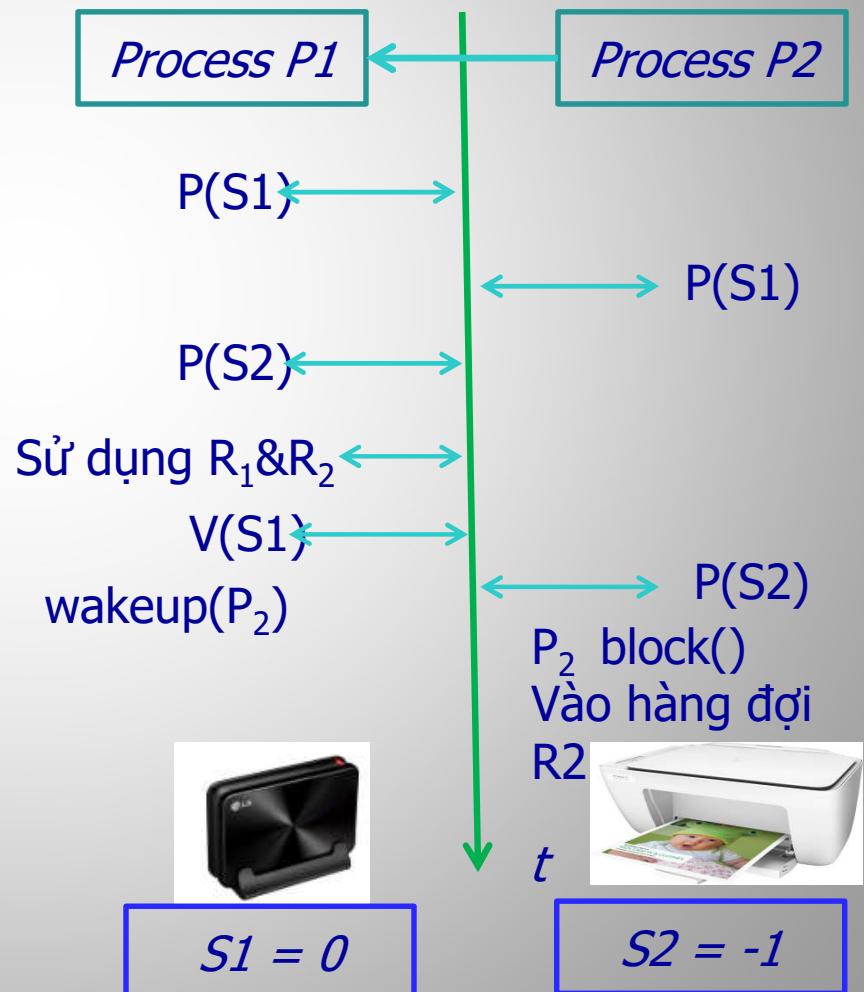
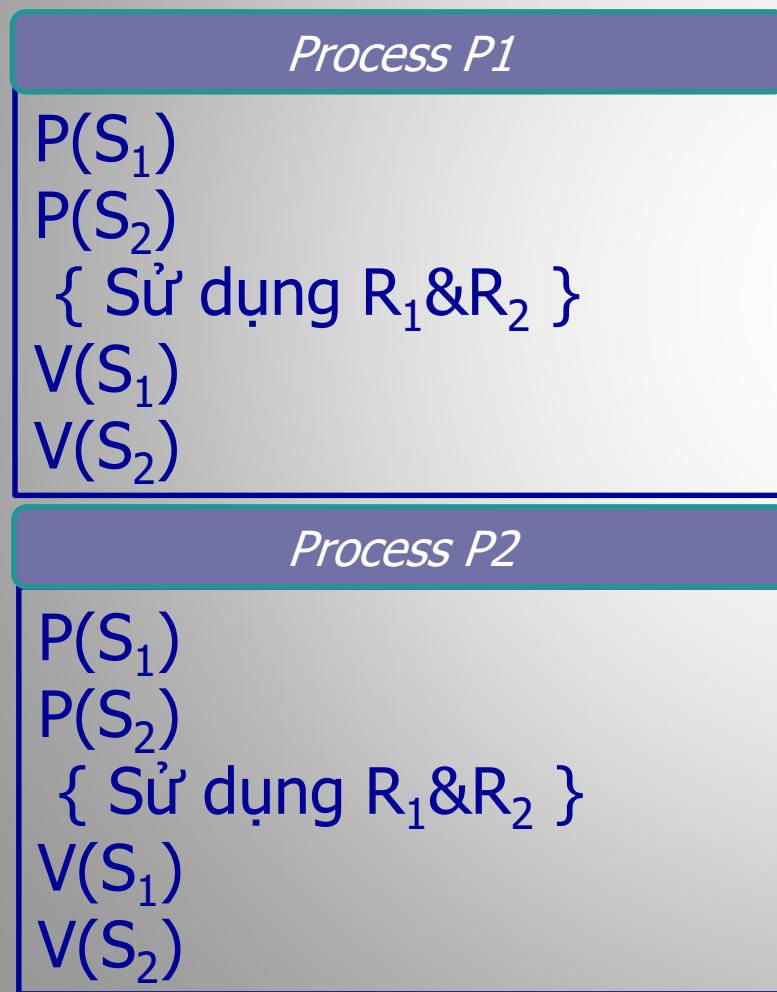


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

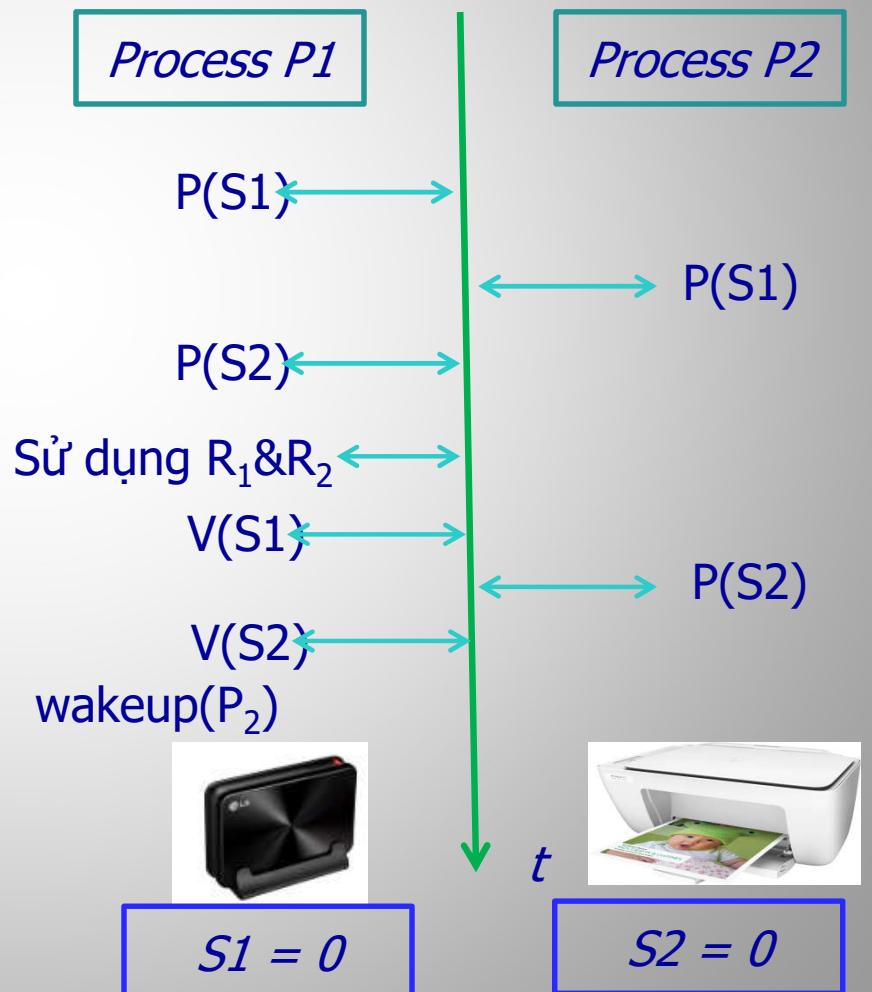
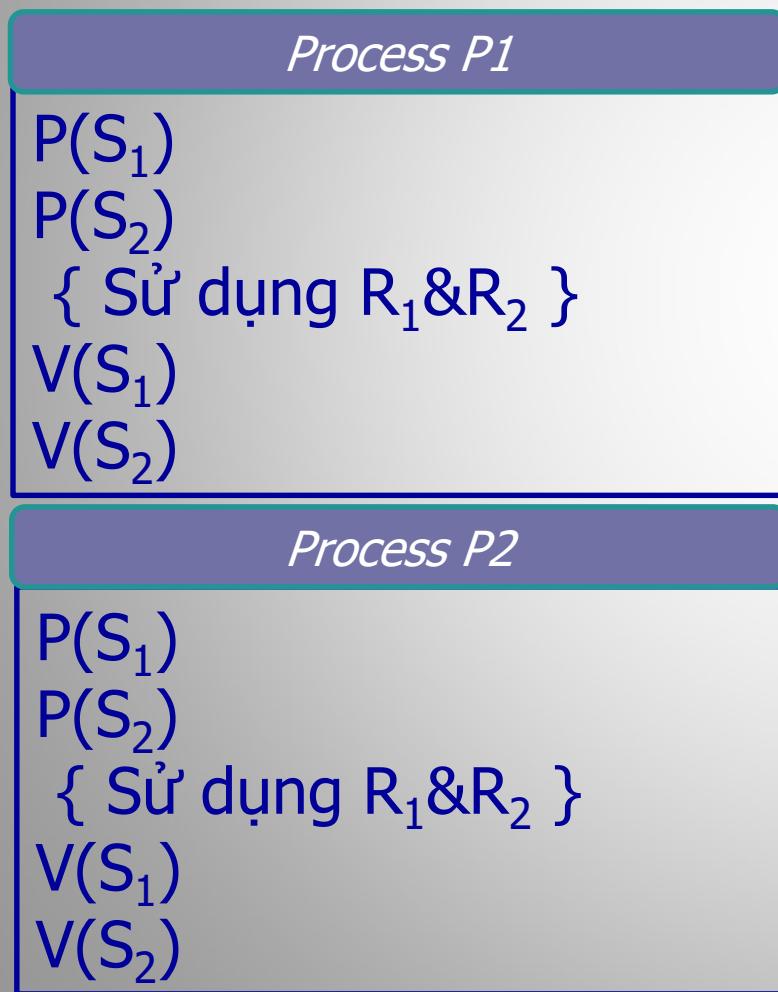


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

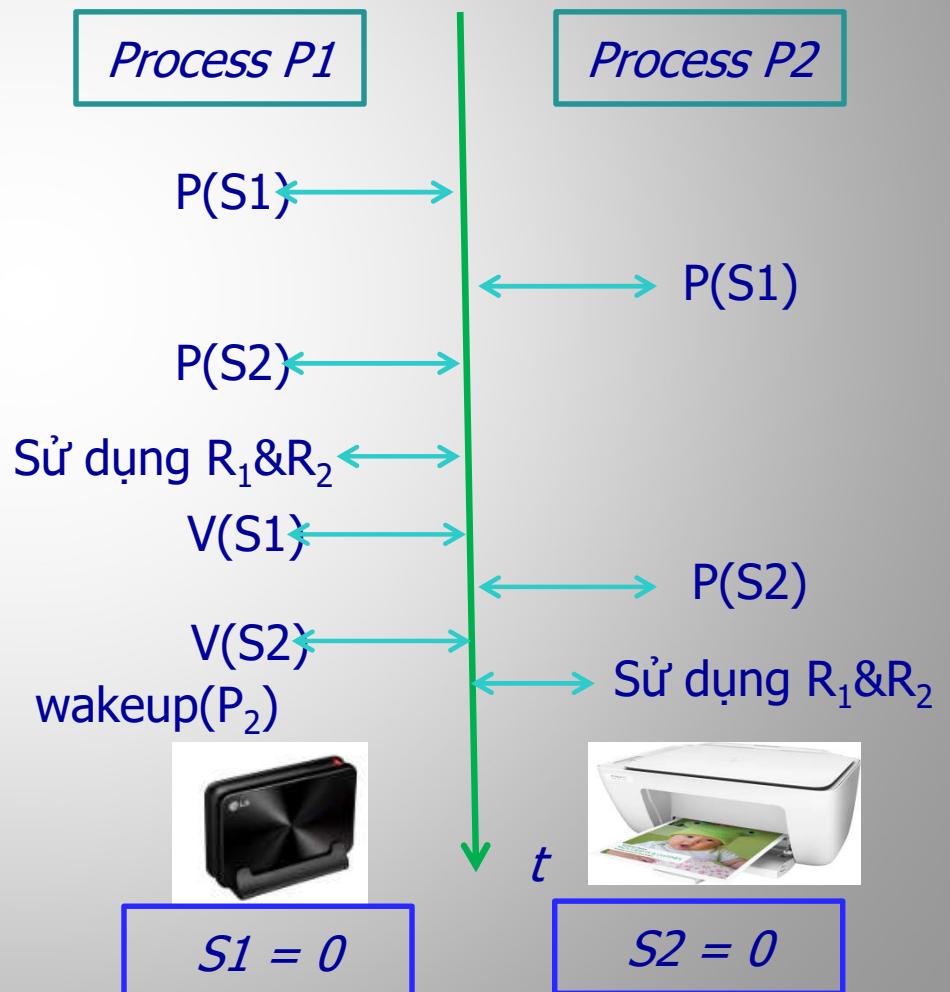
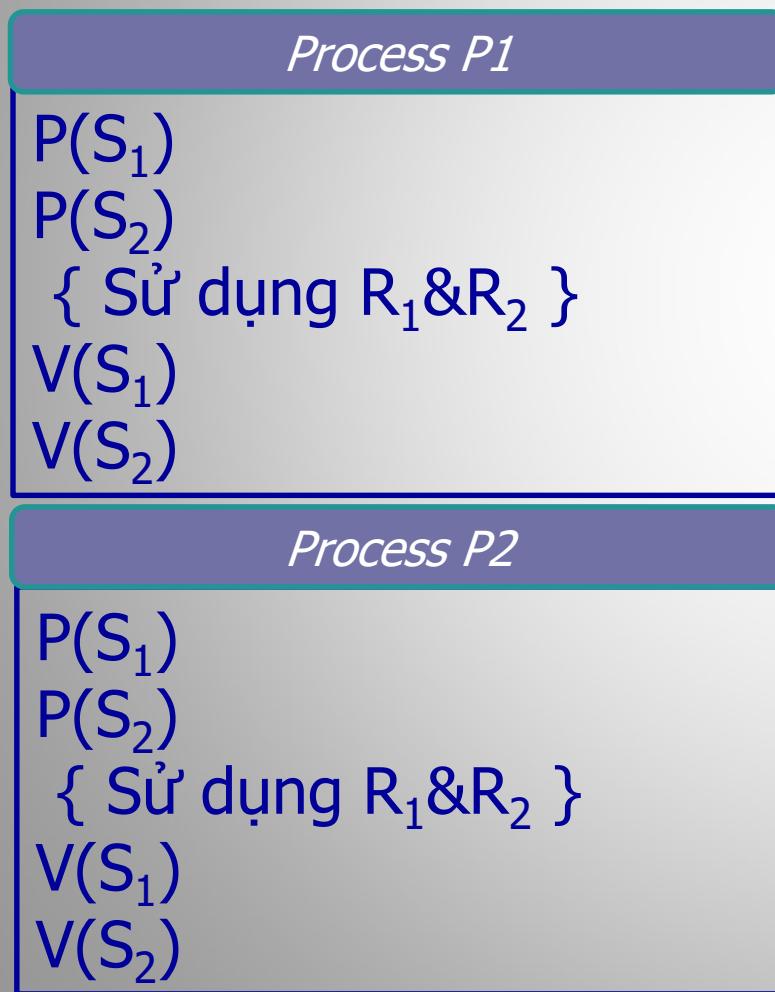


## Chương 2 Quản lý tiến trình

### 5. Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ



## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

*Process P1*

P( $S_1$ )  
P( $S_2$ )  
{ Sử dụng  $R_1 \& R_2$  }  
V( $S_1$ )  
V( $S_2$ )

*Process P1*

*Process P2*

*Process P2*

P( $S_2$ )  
P( $S_1$ )  
{ Sử dụng  $R_1 \& R_2$  }  
V( $S_1$ )  
V( $S_2$ )



$S1 = 1$



$S2 = 1$

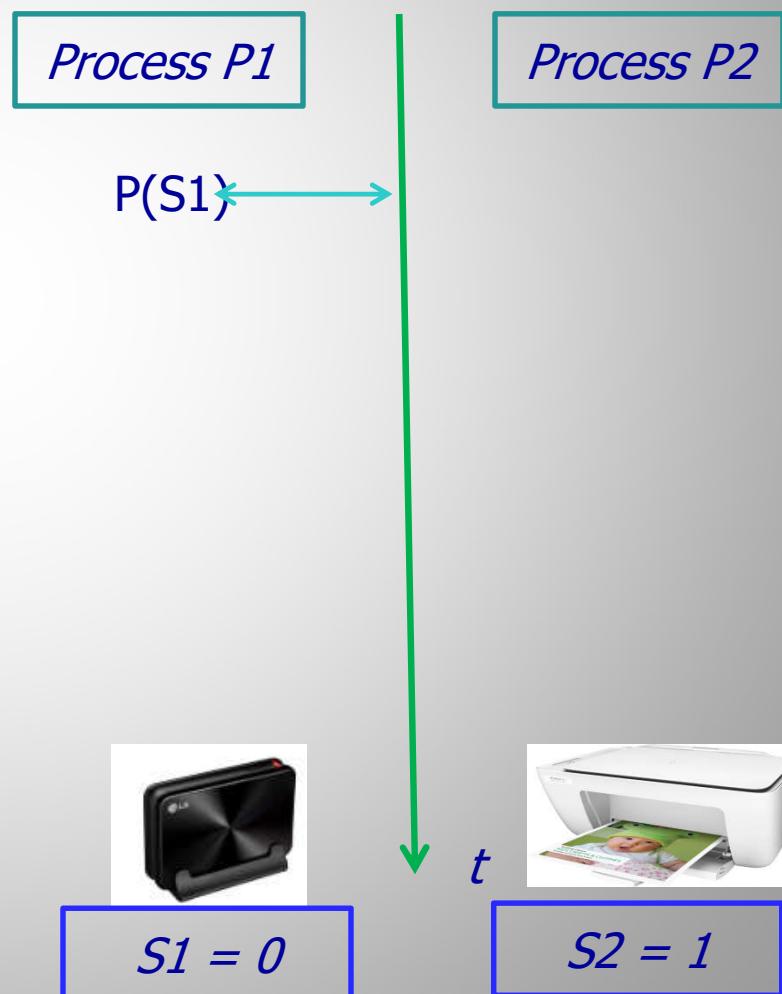
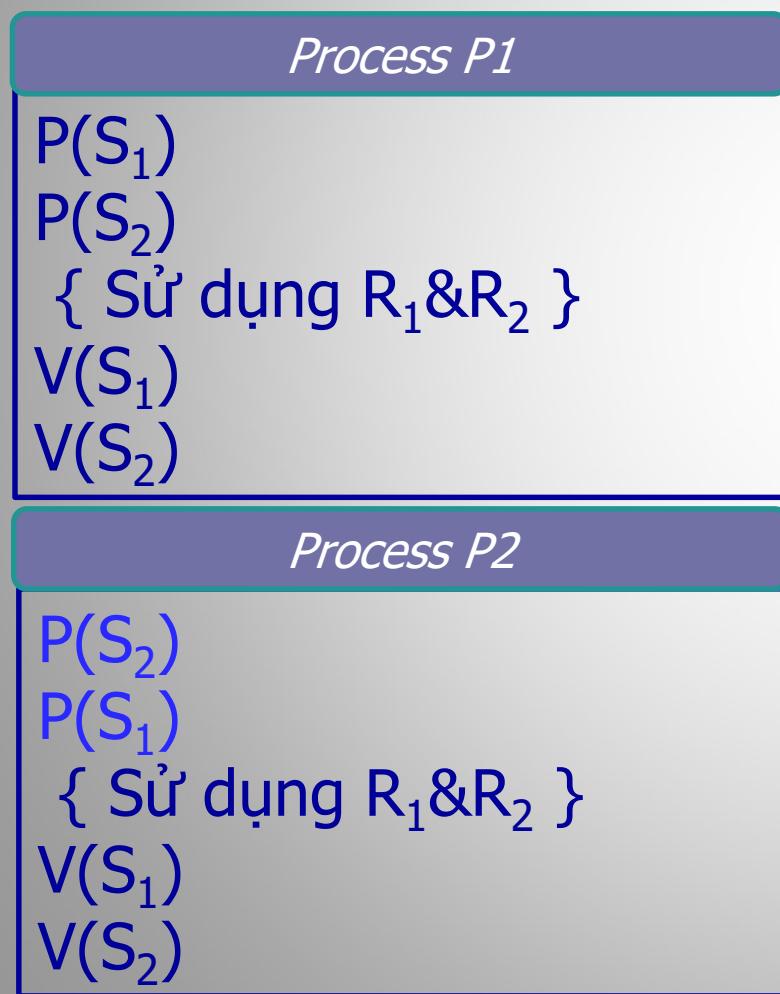
$t$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

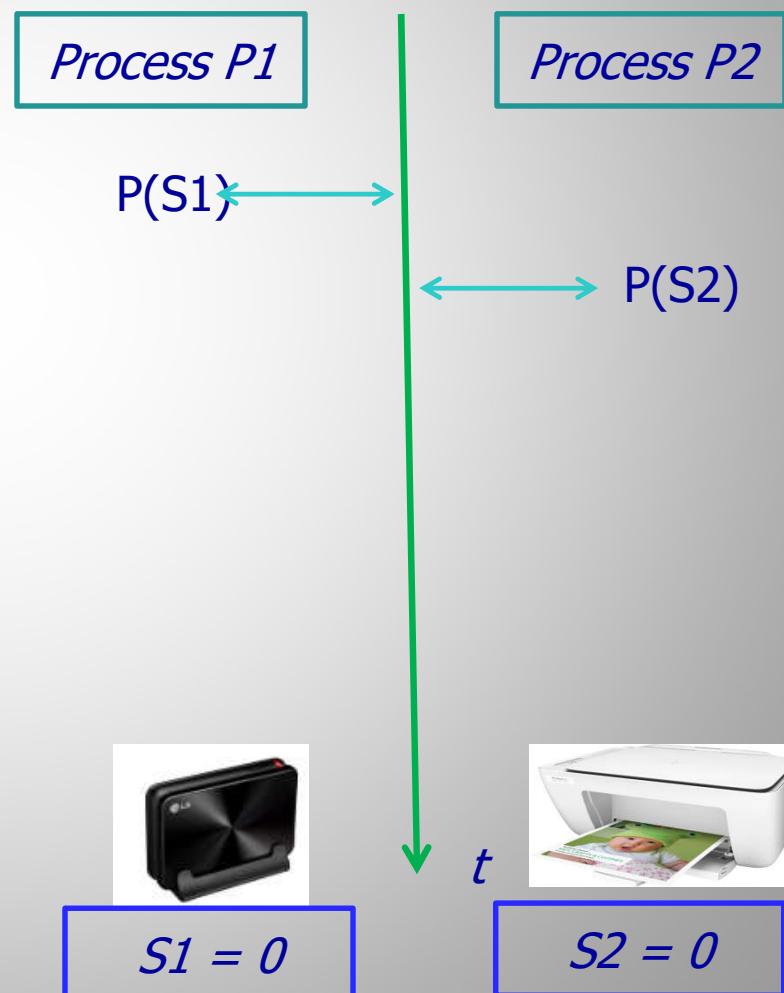
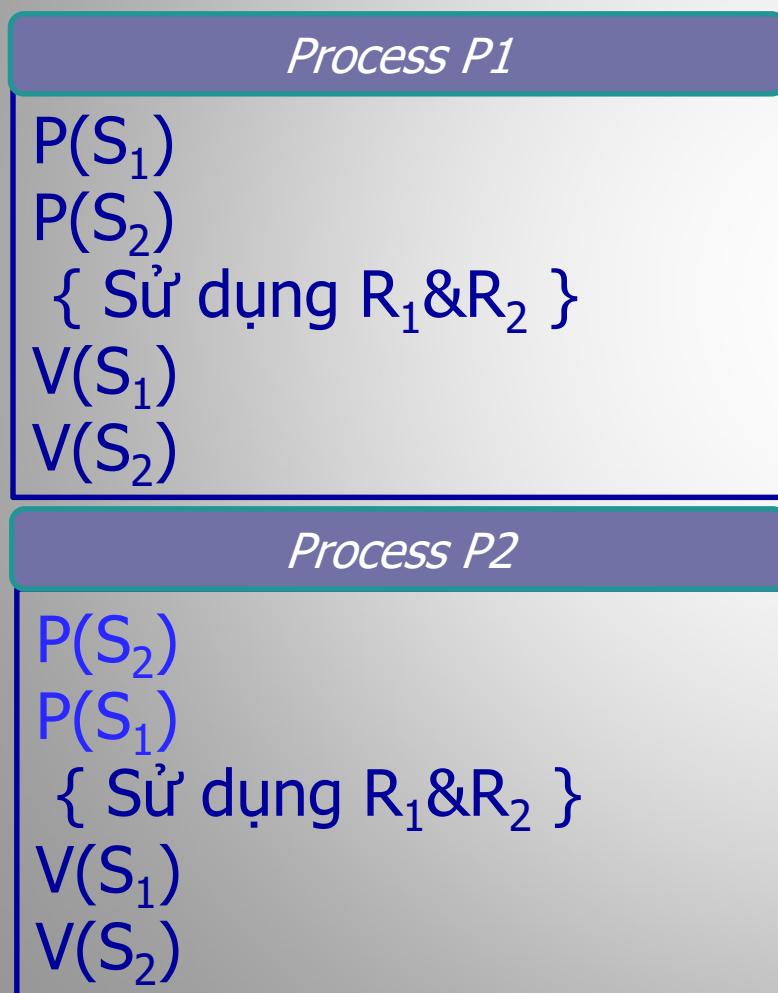


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

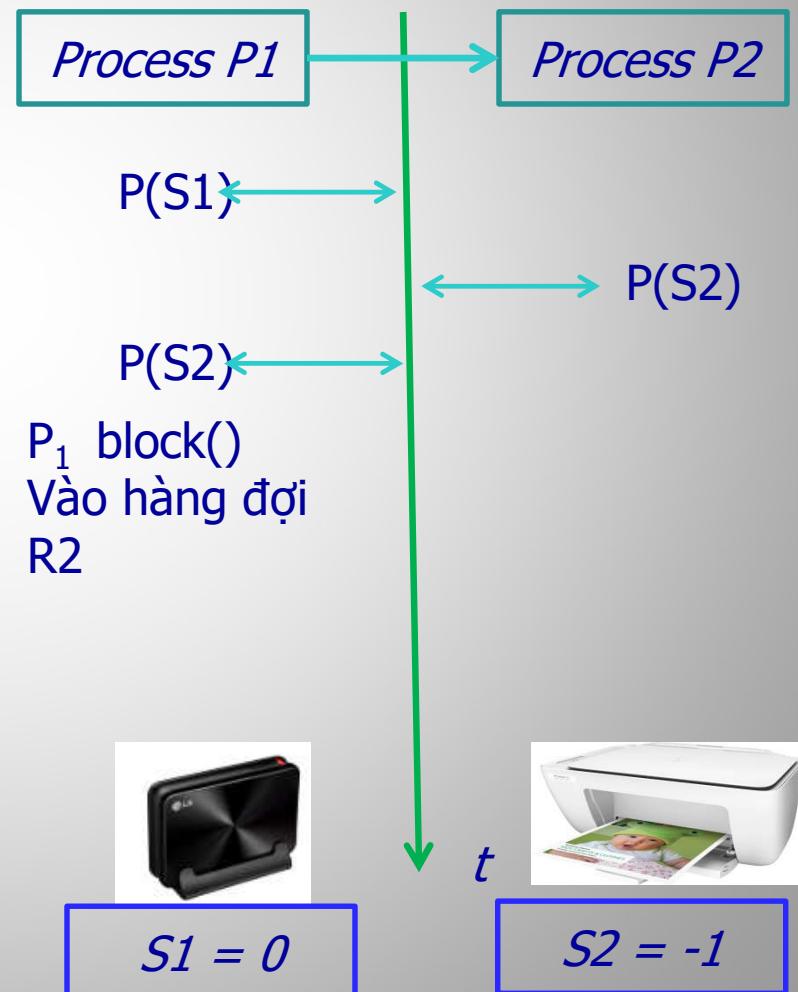
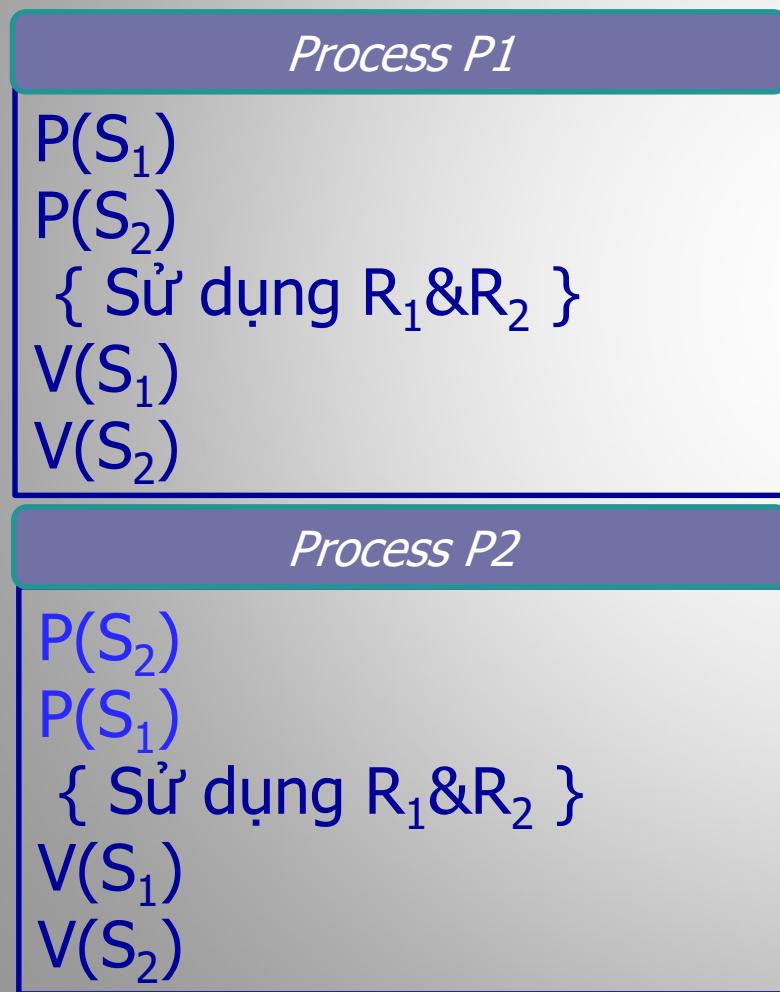


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

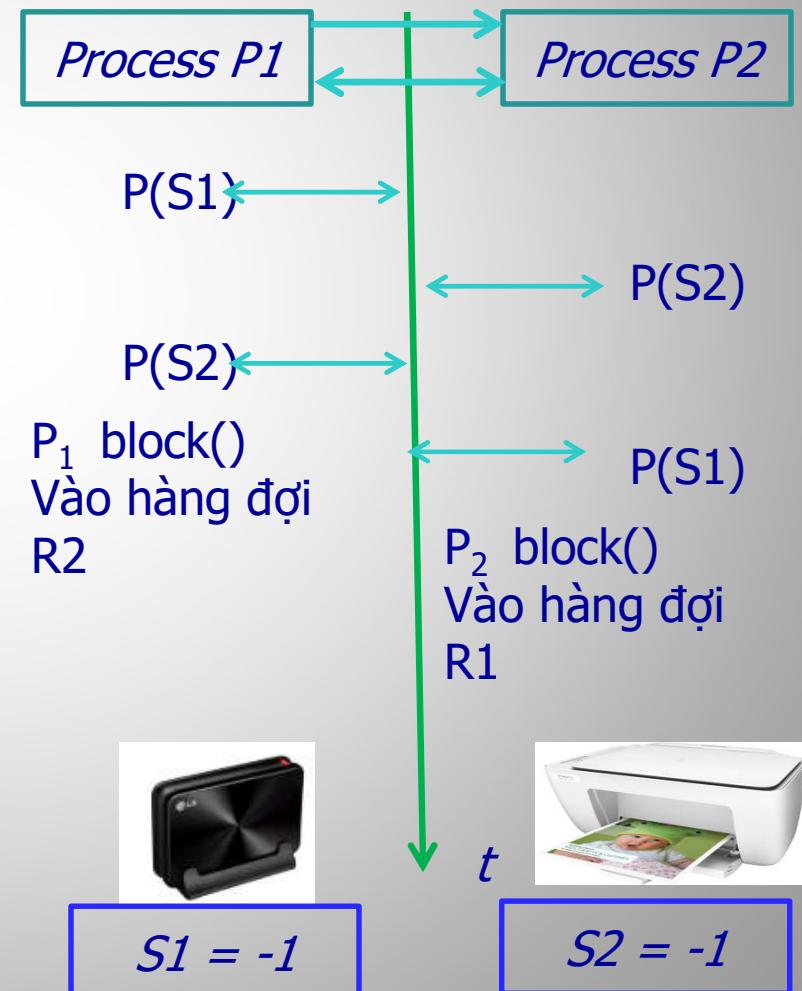
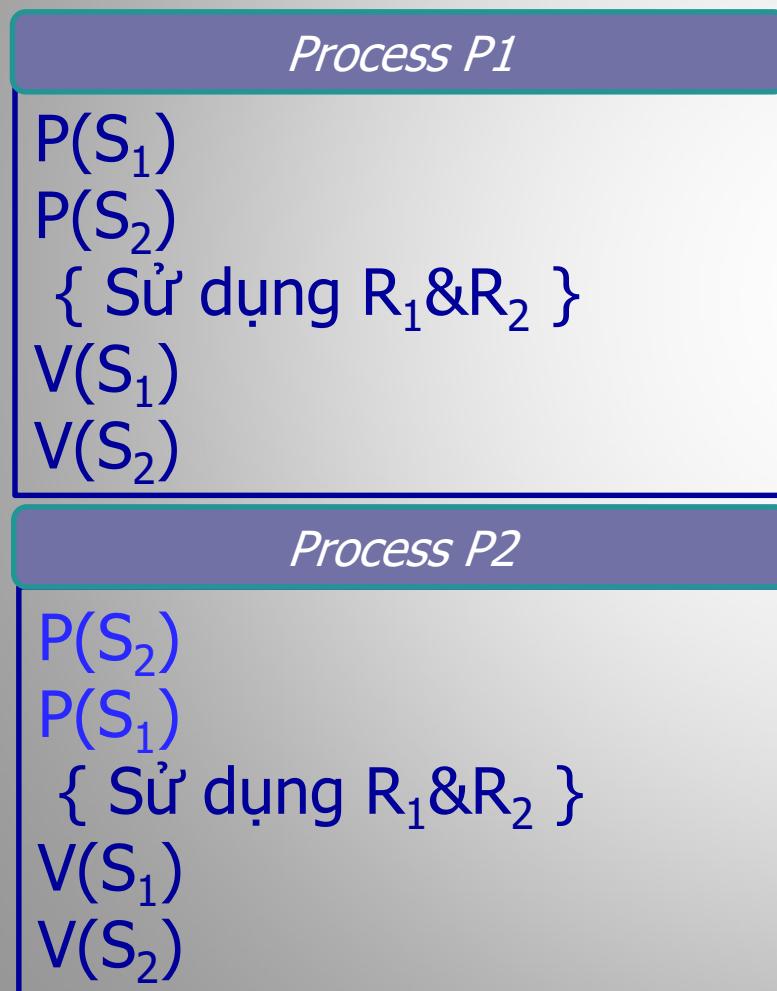


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ

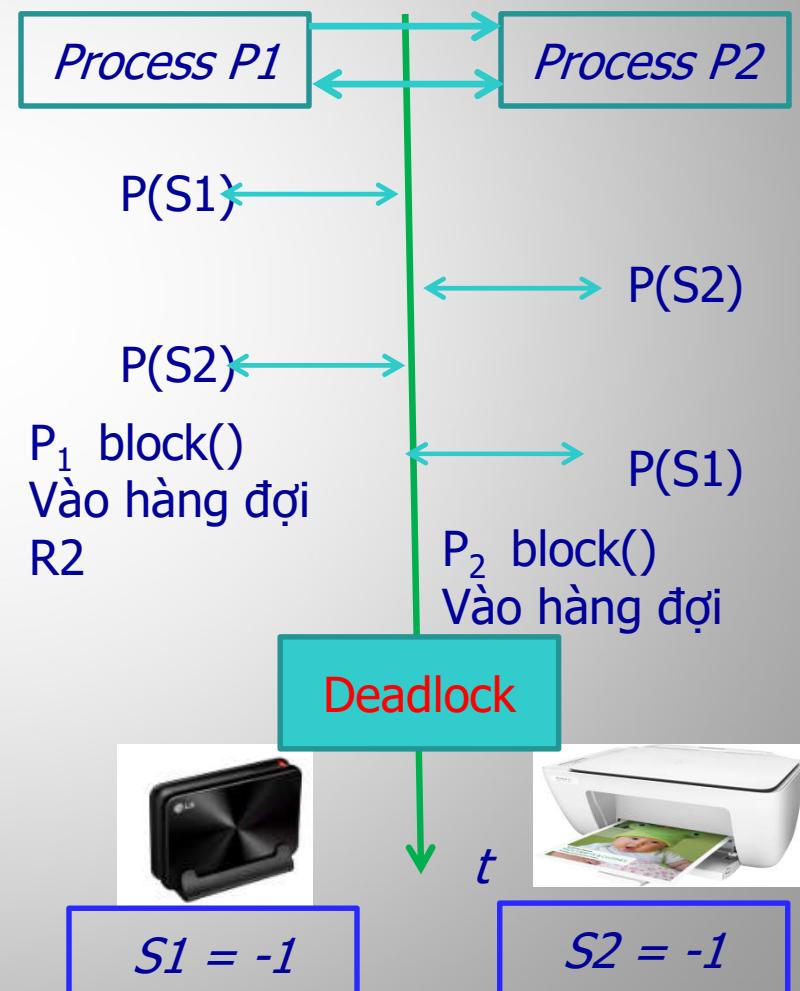
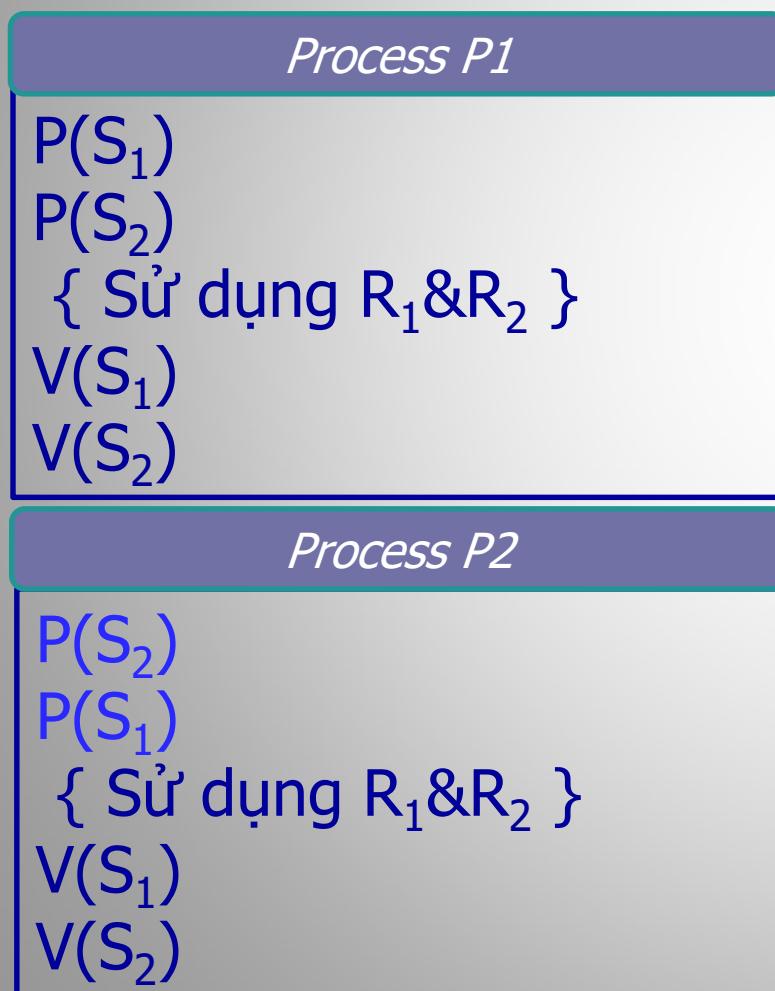


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

Ví dụ



## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

##### Định nghĩa

Bẽ tắc là tình trạng

- Hai hay nhiều tiến trình cùng chờ đợi một sự kiện nào đó xảy ra
- Nếu không có sự tác động gì từ bên ngoài, thì sự chờ đợi đó là vô hạn

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.1. Khái niệm bẽ tắc(Deadlock)

- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- Các phương pháp xử lý bẽ tắc
- Phòng ngừa bẽ tắc
- Phòng tránh bẽ tắc
- Nhận biết và khắc phục

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

##### Điều kiện cản

Cần có 4 điều kiện sau, không được thiếu điều kiện nào

- **Tồn tại tài nguyên găng**
  - TN được sử dụng theo mô hình **không phân chia** được
  - Chỉ có **1 TT** dùng TN tại **1 thời điểm**
  - TT khác cũng yêu cầu TN  $\Rightarrow$  yêu cầu phải được **hoãn lại** tới khi TN được giải phóng
- **Chờ đợi trước khi vào đoạn găng**
  - TT **không** được vào đoạn găng phải **xếp hàng chờ đợi**.
  - Trong khi chờ đợi vẫn **chiếm giữ** các TN được cung cấp
- **Không có hệ thống phân phối lại tài nguyên găng**
  - TN **không thể** được trưng dụng
  - TN được **giải phóng** chỉ bởi TT **đang chiếm giữ** khi đã hoàn thành nhiệm vụ
- **Chờ đợi vòng tròn**
  - Tồn tại tập các TT  $\{P_0, P_1, \dots, P_n\}$  đang đợi nhau theo kiểu:  $P_0 \rightarrow R_1 \rightarrow P_1; P_1 \rightarrow R_2 \rightarrow P_2; \dots; P_{n-1} \rightarrow R_n \rightarrow P_n; P_n \rightarrow R_0 \rightarrow P_0$
  - Chờ đợi vòng tròn tạo ra chu trình **không kết thúc**

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

Ví dụ: Bài toán bữa ăn tối của triết gia

- Tài nguyên găng
- Chờ đợi trước khi vào đoạn găng
- Trưng dụng tài nguyên găng
- Chờ đợi vòng tròn



## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

### Đồ thị cung cấp tài nguyên (Resource Allocation Graph) I

- Dùng để mô hình hóa tình trạng bẽ tắc trong hệ thống
- Là đồ thị định hướng gồm tập đỉnh  $V$  và tập cung  $E$
- Tập đỉnh  $V$  được chia thành 2 kiểu đỉnh
  - $P = \{P_1, P_2, \dots, P_n\}$  Tập chứa tất cả các TT trong hệ thống
  - $R = \{R_1, R_2, \dots, R_m\}$  Tập chứa tất cả các kiểu TN trong hệ thống
- Tập các cung  $E$  gồm 2 loại
  - Cung yêu cầu: đi từ TT  $P_i$  tới TN  $R_j$ :  
 $P_i \rightarrow R_j$
  - Cung sử dụng: Đi từ TN  $R_j$  tới TT  $P_i$ :  
 $R_j \rightarrow P_i$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

### Đồ thị cung cấp tài nguyên (Resource Allocation Graph) II

Khi 1 TT  $P_i$  yêu cầu TN  $R_j$

- Cung yêu cầu  $P_i \rightarrow R_j$  được chèn vào đồ thị
- Nếu yêu cầu được thỏa mãn, cung yêu cầu chuyển thành cung sử dụng  $R_j \rightarrow P_i$
- Khi TT  $P_i$  giải phóng TN  $R_j$ , cung sử dụng  $R_j \rightarrow P_i$  bị xóa khỏi đồ thị

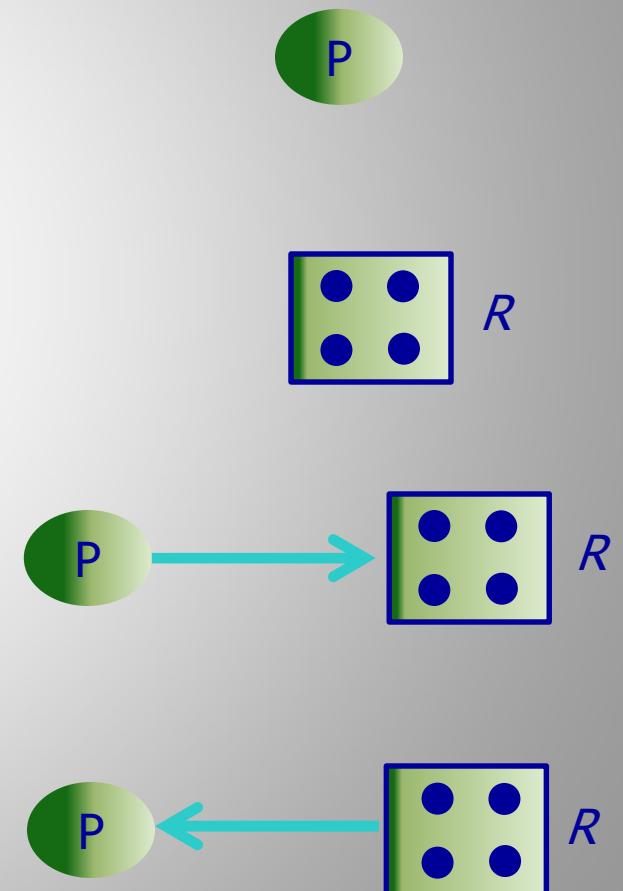
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Biểu diễn đồ trong đồ thị

- Đỉnh kiểu TT
- Đỉnh kiểu TN
  - Mỗi **đơn vị** của kiểu TN được biểu thị bằng 1 dấu chấm trong hình chữ nhật
- Cung yêu cầu đi từ đỉnh TT → đỉnh TN
- Cung sử dụng xuất phát từ dấu chấm bên trong đỉnh TN → đỉnh TT



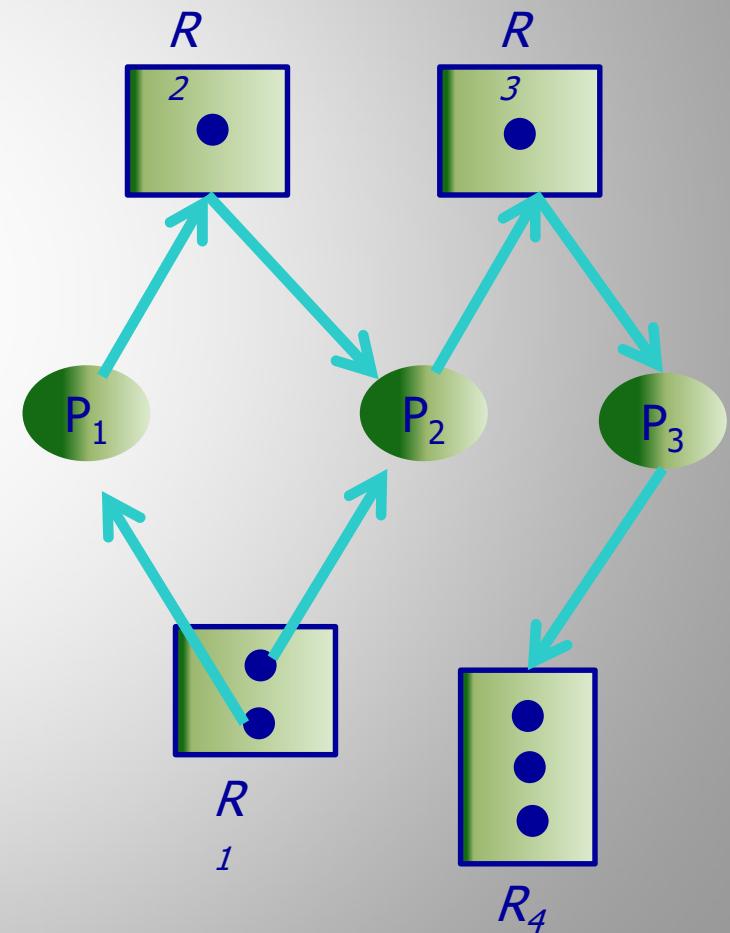
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Ví dụ

- Trạng thái hệ thống
  - 3 TT  $P_1, P_2, P_3$
  - 4 tài nguyên  $R_1, R_2, R_3, R_4$
- $P_3$  yêu cầu tài nguyên  $R_4$ 
  - Xuất hiện cung yêu cầu  $P_3 \rightarrow R_4$



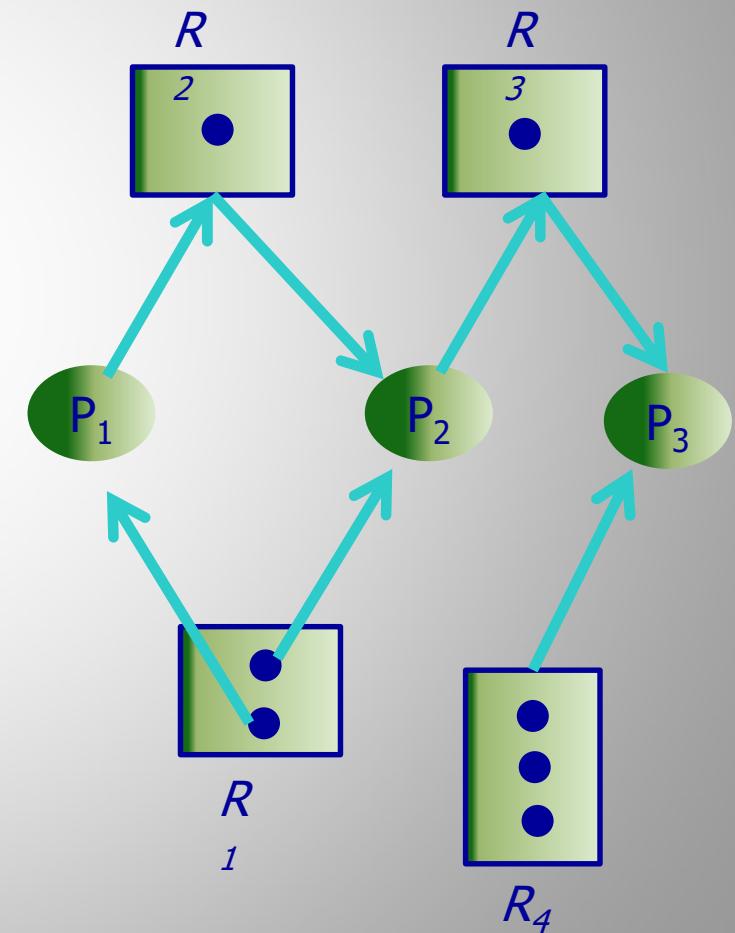
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Ví dụ

- Trạng thái hệ thống
  - 3 TT  $P_1, P_2, P_3$
  - 4 tài nguyên  $R_1, R_2, R_3, R_4$
- $P_3$  yêu cầu tài nguyên  $R_4$ 
  - Xuất hiện cung yêu cầu  $P_3 \rightarrow R_4$
  - Cung yêu cầu  $P_3 \rightarrow R_4$  chuyển thành cung sử dụng  $R_4 \rightarrow P_3$



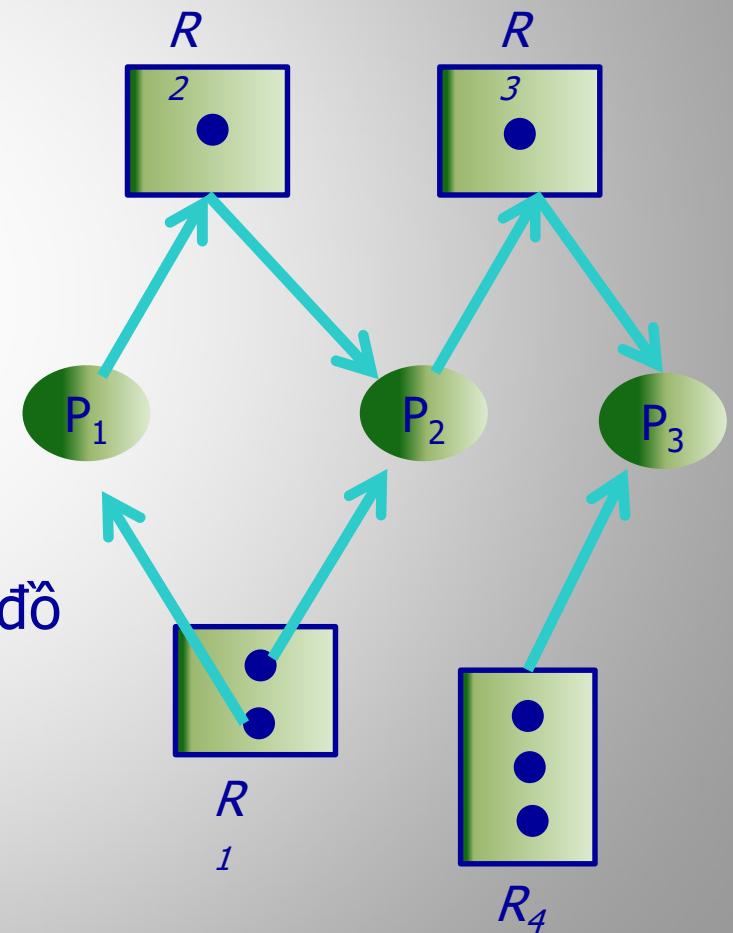
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Ví dụ

- Trạng thái hệ thống
  - 3 TT  $P_1, P_2, P_3$
  - 4 tài nguyên  $R_1, R_2, R_3, R_4$
- $P_3$  yêu cầu tài nguyên  $R_4$ 
  - Xuất hiện cung yêu cầu  $P_3 \rightarrow R_4$
  - Cung yêu cầu  $P_3 \rightarrow R_4$  chuyển thành cung sử dụng  $R_4 \rightarrow P_3$
- $P_3$  Giải phóng tài nguyên  $R_4$ 
  - Cung sử dụng  $R_4 \rightarrow P_3$  bị xóa khỏi đồ thị



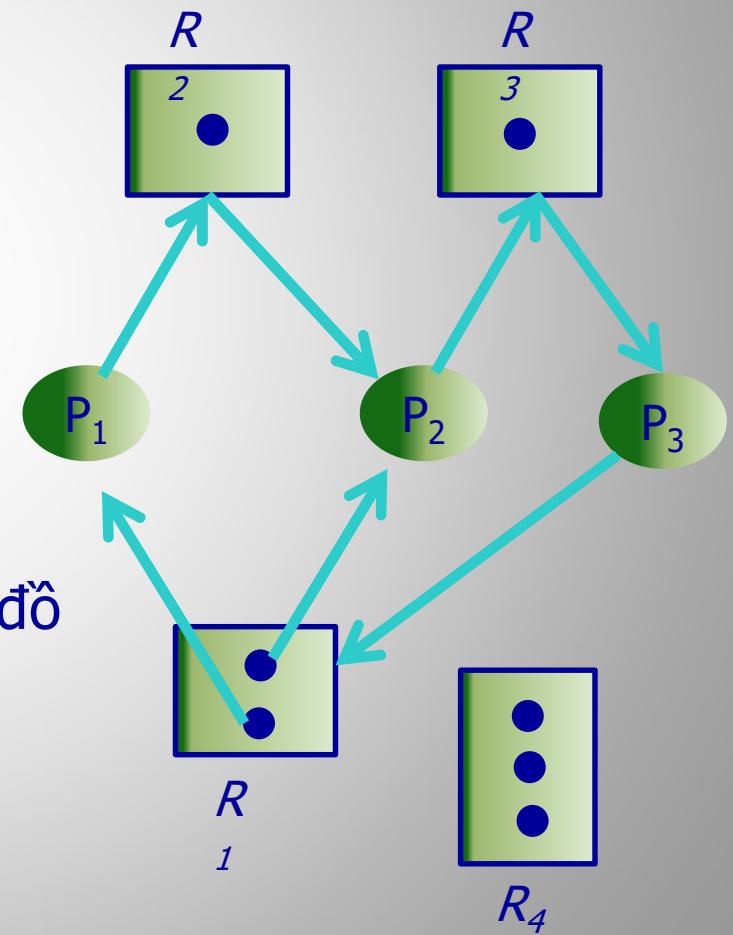
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Ví dụ

- Trạng thái hệ thống
  - 3 TT  $P_1, P_2, P_3$
  - 4 tài nguyên  $R_1, R_2, R_3, R_4$
- $P_3$  yêu cầu tài nguyên  $R_4$ 
  - Xuất hiện cung yêu cầu  $P_3 \rightarrow R_4$
  - Cung yêu cầu  $P_3 \rightarrow R_4$  chuyển thành cung sử dụng  $R_4 \rightarrow P_3$
- $P_3$  Giải phóng tài nguyên  $R_4$ 
  - Cung sử dụng  $R_4 \rightarrow P_3$  bị xóa khỏi đồ thị
- $P_3$  yêu cầu tài nguyên  $R_1$ 
  - Xuất hiện cung yêu cầu  $P_3 \rightarrow R_1$
  - Trên đồ thị xuất hiện chu trình
  - Hệ thống bẽ tắc



## Chương 2 Quản lý tiến trình

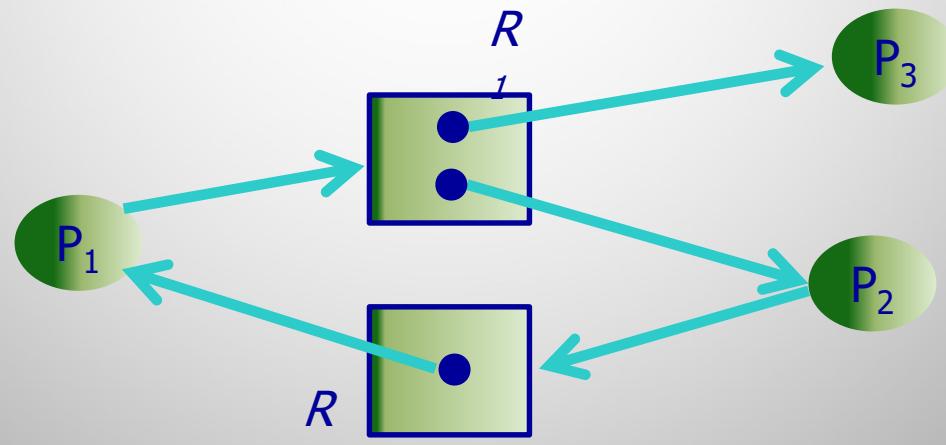
### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.2. Điều kiện xảy ra bẽ tắc

#### Đồ thị cung cấp tài nguyên: Lập luận cơ bản

Chu trình trên đồ thị và tình trạng bẽ tắc có liên quan ?

- Đồ thị không chứa chu trình, không bẽ tắc
- Nếu đồ thị chứa đựng chu trình
  - Nếu TN chỉ có 1 đơn vị  $\Rightarrow$  Bẽ tắc
  - Nếu TN có nhiều hơn 1 đơn vị: có khả năng bẽ tắc



Đồ thị có chu trình nhưng hệ thống không bẽ tắc

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.3 Các phương pháp xử lý bẽ tắc

- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- **Các phương pháp xử lý bẽ tắc**
  - Phòng ngừa bẽ tắc
  - Phòng tránh bẽ tắc
  - Nhận biết và khắc phục

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.3. Các phương pháp xử lý bẽ tắc

#### Phương pháp

- Phòng ngừa
  - Áp dụng các biện pháp để đảm bảo hệ thống không bao giờ rơi vào tình trạng bẽ tắc
  - Tốn kém
  - Áp dụng cho hệ thống hay xảy ra bẽ tắc và tổn thất do bẽ tắc gây ra lớn
- Phòng tránh
  - Kiểm tra từng yêu cầu TN của TT và không chấp nhận yêu cầu nếu việc cung cấp TN có khả năng dẫn đến tình trạng bẽ tắc
  - Thường yêu cầu các thông tin phụ trợ
  - Áp dụng cho hệ thống ít xảy ra bẽ tắc nhưng tổn hại lớn
- Nhận biết và khắc phục
  - Cho phép hệ thống hoạt động bình thường ⇒ có thể rơi vào tình trạng bẽ tắc
  - Định kỳ kiểm tra xem bẽ tắc có đang xảy ra không
  - Nếu đang bẽ tắc, áp dụng các biện pháp loại bỏ bẽ tắc
  - Áp dụng cho hệ thống ít xảy ra bẽ tắc và thiệt hại không lớn

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- Các phương pháp xử lý bẽ tắc
- **Phòng ngừa bẽ tắc**
  - Phòng tránh bẽ tắc
  - Nhận biết và khắc phục

Chương 2 Quản lý tiến trình

5.Bẽ tắc và xử lí bẽ tắc

5.4 Phòng ngừa bẽ tắc

### Nguyên tắc

Tác động vào 1 trong 4 điều kiện cần của bẽ tắc để nó không xảy ra

Tài nguyên găng

Chờ đợi trước khi vào đoạn găng

Trưng dụng tài nguyên găng

Chờ đợi vòng tròn

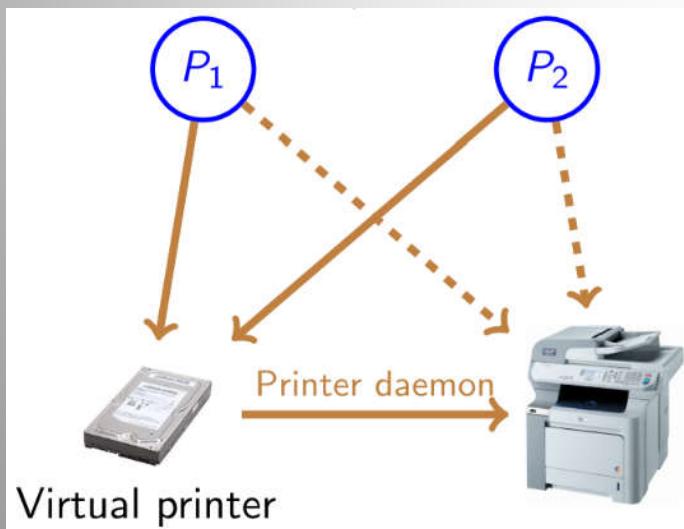
## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

### Điều kiện tài nguyên găng

- Giảm bớt mức độ găng của hệ thống
  - TN **phân chia được** (file chỉ đọc): Sử dụng đồng thời
  - TN **không phân chia được**: Sử dụng không đồng thời
- Kỹ thuật SPOOL(*Simultaneous peripheral operation on-line*)
  - Không phân phối TN khi không thực sự cần thiết
  - Chỉ một số ít TT có khả năng yêu cầu TN



- Chỉ *printer daemon* mới làm việc với máy in ⇒ Bẽ tắc cho TN máy in bị hủy bỏ
- Không phải TN nào cũng dùng kỹ thuật SPOOL được

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện chờ đợi trước khi vào đoạn găng

**Nguyên tắc:** Đảm bảo 1 TT xin TN chỉ khi không sở hữu bất kỳ TN nào khác

- Cung cấp trước
  - TT xin **toàn bộ** TN ngay từ đầu và chỉ thực hiện khi đã có **đầy đủ** TN
  - **Hiệu quả sử dụng TN thấp**
    - TT chỉ sử dụng TN ở giai đoạn cuối?
    - Tổng số TN đòi hỏi vượt quá khả năng của hệ thống?
- Giải phóng TN
  - TT giải phóng **tất cả** TN trước khi xin (xin lại) TN mới
  - Nhận xét
    - Tốc độ thực hiện TT chậm
    - Phải đảm bảo dữ liệu được giữ trong TN tạm giải phóng không bị mất

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện chờ đợi trước khi vào đoạn găng: Ví dụ

- TT gồm 2 giai đoạn
  - Sao chép dữ liệu từ băng từ sang một file trên đĩa từ
  - Sắp xếp dữ liệu trong file và đưa ra máy in



- Phương pháp cung cấp trước
  - Xin cả băng từ, file trên đĩa và máy in
  - **Lặng phí** máy in giai đoạn đầu, băng từ giai đoạn cuối

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện chờ đợi trước khi vào đoạn găng: Ví dụ

- TT gồm 2 giai đoạn
  - Sao chép dữ liệu từ băng từ sang một file trên đĩa từ
  - Sắp xếp dữ liệu trong file và đưa ra máy in



- Phương pháp cung cấp trước
  - Xin cả băng từ, file trên đĩa và máy in
  - **Lặng phí** máy in giai đoạn đầu, băng từ giai đoạn cuối
- Phương pháp giải phóng tài nguyên
  - Xin băng từ và file trên đĩa cho giai đoạn 1

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện chờ đợi trước khi vào đoạn găng: Ví dụ

- TT gồm 2 giai đoạn
  - Sao chép dữ liệu từ băng từ sang một file trên đĩa từ
  - Sắp xếp dữ liệu trong file và đưa ra máy in



- Phương pháp cung cấp trước
  - Xin cả băng từ, file trên đĩa và máy in
  - **Lãng phí** máy in giai đoạn đầu, băng từ giai đoạn cuối
- Phương pháp giải phóng TN
  - Xin băng từ và file trên đĩa cho giai đoạn 1
  - Giải phóng băng từ và file trên đĩa
  - Xin file trên đĩa và máy in cho giai đoạn 2

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện trưng dụng tài nguyên găng

**Nguyên tắc:** cho phép trưng dụng TN khi cần thiết

- TT  $P_i$  xin tài nguyên  $R_j$ 
  - $R_j$  sẵn có: Cung cấp  $R_j$  cho  $P_i$
  - $R_j$  không sẵn: ( $R_j$  bị chiếm bởi TT  $P_k$ )
    - $P_k$  đang đợi TN khác
      - Trưng dụng  $R_j$  từ  $P_k$  và cung cấp cho  $P_i$  theo yêu cầu
      - Thêm  $R_j$  vào danh sách các TN đang thiếu của  $P_k$
      - $P_k$  được thực hiện trở lại khi
        - Có được TN đang thiếu
        - Đòi lại được  $R_j$
    - $P_k$  đang thực hiện
      - $P_i$  phải đợi (*không giải phóng tài nguyên*)
      - Cho phép trưng dụng TN nhưng chỉ khi cần thiết

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

#### Điều kiện trưng dụng tài nguyên găng

**Nguyên tắc:** cho phép trưng dụng tài nguyên khi cần thiết

- Chỉ áp dụng cho các TN có thể lưu trữ và khôi phục trạng thái dễ dàng
  - (*CPU, không gian nhớ*)
  - Khó có thể áp dụng cho các TN như máy in
- 1 TT bị trưng dụng nhiều lần ?

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

##### Điều kiện chờ đợi vòng tròn

- Đặt ra một thứ tự toàn cục của tất cả các kiểu TN
  - $R = \{R_1, R_2, \dots, R_n\}$  Tập tất cả các kiểu TN
  - Xây dựng hàm trật tự  $f : R \rightarrow N$ 
    - Hàm  $f$  được xây dựng dựa trên trật tự sử dụng các TN
      - $f(\text{Băng từ}) = 1$
      - $f(\text{Đĩa từ}) = 5$
      - $f(\text{Máy in}) = 12$
- TT chỉ được yêu cầu TN theo trật tự tăng
  - TT chiếm giữ TN kiểu  $R_k$  chỉ được xin TN kiểu  $R_j$  thỏa mãn  $f(R_j) > f(R_k)$
  - TT yêu cầu tới TN  $R_k$  sẽ phải giải phóng tất cả TN  $R_i$  thỏa mãn điều kiện  $f(R_i) \geq f(R_k)$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.4 Phòng ngừa bẽ tắc

##### Điều kiện chờ đợi vòng tròn

- TT chỉ được yêu cầu TN theo trật tự tăng
  - TT chiếm giữ TN kiểu  $R_k$  chỉ được xin TN kiểu  $R_j$  thỏa mãn  $f(R_j) > f(R_k)$
  - TT yêu cầu tới TN  $R_k$  sẽ phải giải phóng tất cả TN  $R_i$  thỏa mãn điều kiện  $f(R_i) \geq f(R_k)$
- Chứng minh
  - Giả thiết bẽ tắc xảy ra giữa các TT  $\{P_1, P_2, \dots, P_m\}$ 
    - $R_1 \rightarrow P_1 \rightarrow R_2 \rightarrow P_2 \Rightarrow f(R_1) < f(R_2)$
    - $R_2 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \Rightarrow f(R_2) < f(R_3) \dots$
    - $R_m \rightarrow P_m \rightarrow R_1 \rightarrow P_1 \Rightarrow f(R_m) < f(R_1)$
  - $f(R_1) < f(R_2) < \dots < f(R_m) < f(R_1) \Rightarrow$  Vô lý

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

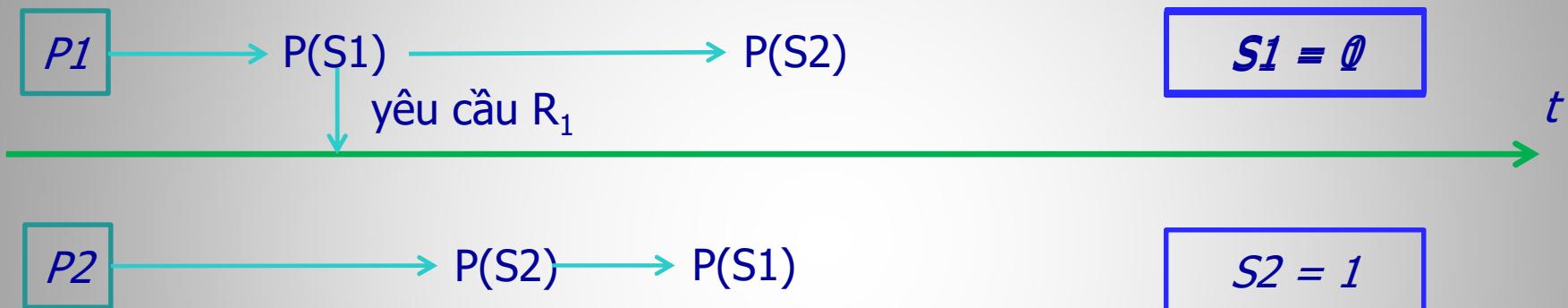
- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- Các phương pháp xử lý bẽ tắc
- Phòng ngừa bẽ tắc
- **Phòng tránh bẽ tắc**
- Nhận biết và khắc phục

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ

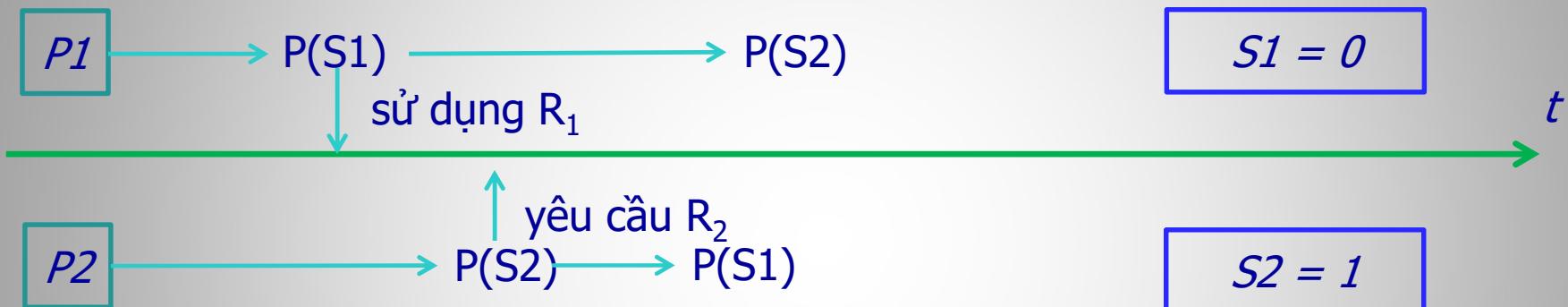


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ

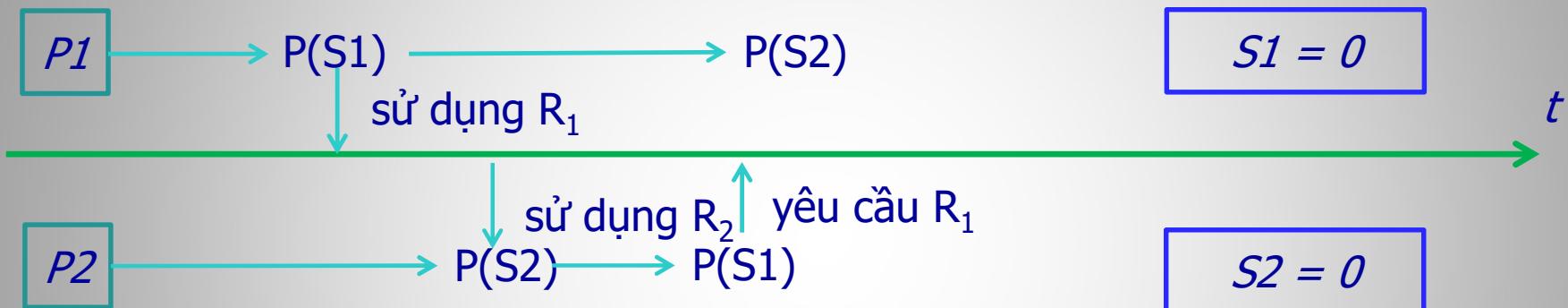


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ

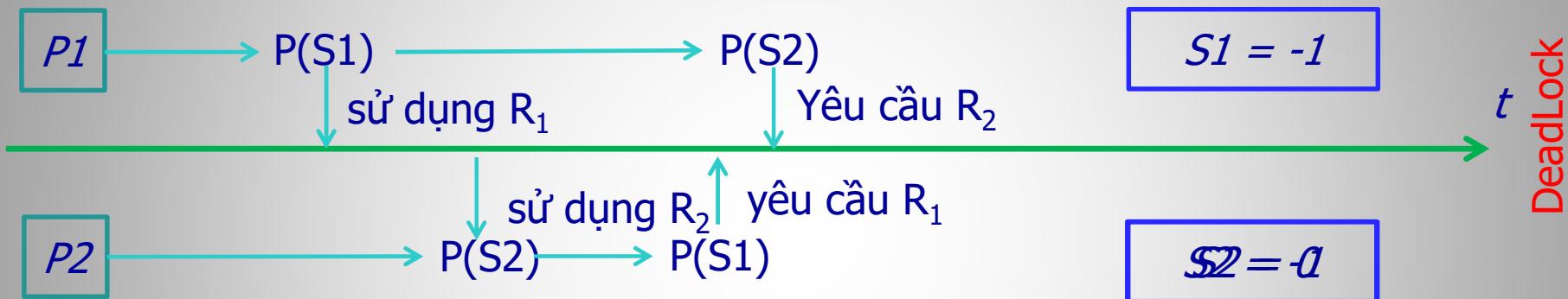


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ

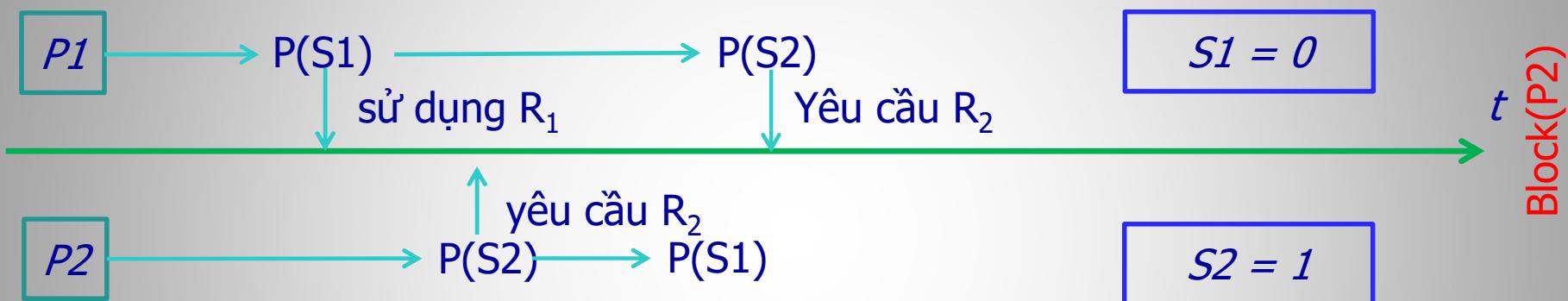


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ

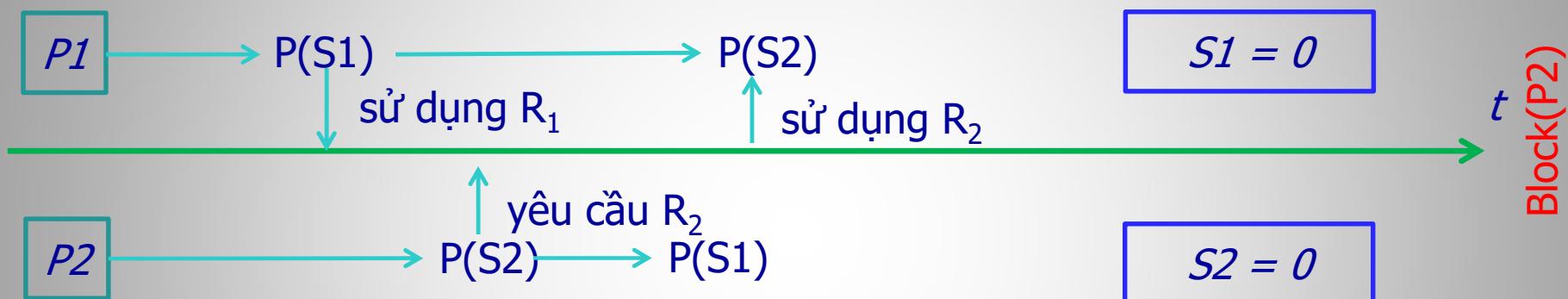


## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Ví dụ



Nhận xét:

Nếu biết được chuỗi yêu cầu/giải phóng TN của các TT, hệ thống có thể đưa ra được chiến lược phân phối TN (chấp thuận hay phải đợi) cho mọi yêu cầu để bẽ tắc không xảy ra.

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Nguyên tắc

- Phải biết trước các thông tin về TT và TN
    - TT phải khai báo lượng TN lớn nhất mỗi loại sẽ yêu cầu khi thực hiện
  - Quyết định dựa trên kết quả kiểm tra t/thái cung cấp TN (Resource-Allocation State) –T/thái hệ thống
    - T/thái cung cấp t/nguyên xác định bởi các thông số:
      - Số đơn vị TN
        - có sẵn trong hệ thống
        - đã được cấp cho mỗi TT
        - lớn nhất mỗi TT có thể yêu cầu
      - Nếu **hệ thống an toàn**, sẽ đáp ứng cho yêu cầu
  - Thực hiện kiểm tra mỗi khi nhận được yêu cầu TN
    - Mục đích: Đảm bảo t/thái hệ thống luôn an toàn
      - Thời điểm ban đầu (chưa c/cấp tài nguyên), hệ thống an toàn
      - Hệ thống chỉ cung cấp TN khi vẫn đảm bảo an toàn
- ⇒**Hệ thống chuyển từ t/thái an toàn này → t/thái an toàn khác**

Chương 2 Quản lý tiến trình

5.Bẽ tắc và xử lí bẽ tắc

5.5 Phòng tránh bẽ tắc

Trạng thái an toàn

Trạng thái của hệ thống là an toàn khi

- Có thể cung cấp TN cho từng TT (đến yêu cầu lớn nhất) theo 1 trật tự nào đấy mà không xảy ra bẽ tắc
- Tồn tại chuỗi an toàn của tất cả các TT

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Chuỗi an toàn

Chuỗi tiến trình  $P=\{P_1, P_2, \dots, P_n\}$  là an toàn nếu

- Với mỗi TT  $P_i$ , mọi yêu cầu TN trong tương lai đều có thể đáp ứng nhờ vào
  - Lượng TN hiện có trong hệ thống
  - TN đang chiếm giữ bởi tất cả các TT  $P_j (j < i)$

Trong chuỗi an toàn, khi  $P_i$  yêu cầu TN

- Nếu không thể đáp ứng ngay lập tức,  $P_i$  đợi cho tới khi  $P_j$  kết thúc ( $j < i$ )
- Khi  $P_j$  kết thúc và giải phóng TN,  $P_i$  sẽ nhận được TN cần thiết, thực hiện, giải phóng các TN đã được cung cấp và kết thúc
- Khi  $P_i$  kết thúc và giải phóng TN  $\Rightarrow P_{i+1}$  sẽ nhận được TN cần thiết và kết thúc được . . .
- Tất cả các TT trong chuỗi an toàn đều kết thúc được

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ minh họa

##### Xem xét hệ thống gồm

- 3 TT  $P_1, P_2, P_3$  và 1 TN R có 12 đơn vị
- $(P_1, P_2, P_3)$  có thể yêu cầu tối đa tới (10, 4, 9) đơn vị TN R
- Tại thời điểm  $t_0$ ,  $(P_1, P_2, P_3)$  đã được cấp (5, 2, 2) đơn vị TN R
- Tại thời điểm hiện tại ( $t_0$ ) hệ thống có an toàn?
  - Hệ thống đã cấp (5 + 2 + 2) đơn vị, vậy còn lại 3 đơn vị
  - $(P_1, P_2, P_3)$  còn có thể yêu cầu (5, 2, 7) đơn vị
  - Với 3 đơn vị hiện có, mọi yêu cầu của  $P_2$  đều đáp ứng được  $\Rightarrow P_2$  chắc chắn kết thúc được và sẽ giải phóng 2 đơn vị R
  - Với 3 + 2 đơn vị,  $P_1$  chắc chắn kết thúc, sẽ giải phóng 5 đơn vị
  - Với 3 + 2 + 5 đơn vị  $P_3$  chắc chắn kết thúc được
- Ở thời điểm  $t_0$   $P_1, P_2, P_3$  đều chắc chắn kết thúc  $\Rightarrow$  hệ thống an toàn với dãy an toàn  $(P_2, P_1, P_3)$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Ví dụ minh họa 2

Xem xét hệ thống gồm

- 3 TT  $P_1, P_2, P_3$  và 1 tài nguyên R có 12 đơn vị
- Các TT ( $P_1, P_2, P_3$ ) có thể yêu cầu tối đa tới (10, 4, 9) đơn vị TN R
- Tại thời điểm  $t_0$ , các TT ( $P_1, P_2, P_3$ ) đã được cấp (5, 2, 2) đơn vị TN R
- Tại thời điểm  $t_1$ , TT  $P_3$  yêu cầu và được cấp 1 đơn vị TN R. Hệ thống có an toàn?
  - Với 2 đơn vị hiện có, mọi yêu cầu của  $P_2$  đều đáp ứng được  $\Rightarrow P_2$  chắc chắn kết thúc, giải phóng 2 đơn vị R
  - Khi  $P_2$  kết thúc số TN sẵn có trong hệ thống là 4
  - Với 4 đơn vị TN,  $P_1$  và  $P_3$  đều có thể phải đợi khi xin thêm 5 đơn vị TN
  - Vậy hệ thống không an toàn với dãy ( $P_1, P_3$ )
- **Nhận xét:** Tại thời điểm  $t_1$  nếu TT  $P_3$  phải đợi khi yêu cầu thêm 1 đơn vị TN, bẽ tắc sẽ được loại trừ

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Ví dụ minh họa

- Nhận xét
  - Hệ thống an toàn ⇒ Các TT đều có thể kết thúc được  
⇒ không xảy ra bẽ tắc
  - Hệ thống không an toàn ⇒ Có khả năng xảy ra bẽ tắc
- Phương pháp
  - Không để hệ thống rơi vào tình trạng không an toàn
  - Kiểm tra mọi yêu cầu TN
    - Nếu hệ thống vẫn an toàn khi cung cấp ⇒ Cung cấp
    - Nếu hệ thống không an toàn khi cung cấp ⇒ Phải đợi
- Thuật toán
  - Thuật toán dựa vào đồ thị cung cấp tài nguyên
  - Thuật toán người quản lý nhà băng

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Thuật toán dựa vào đồ thị cung cấp tài nguyên

- Sử dụng khi mỗi kiểu TN chỉ có 1 đơn vị
  - Có chu trình, sẽ có bẽ tắc
- Thêm vào đồ thị loại cung mới: cung đòi hỏi  $P_i \rightarrow R_j$ 
  - Cùng hướng với cung yêu cầu, thể hiện trong đồ thị -- >
  - Cho biết  $P_i$  có thể yêu cầu  $R_j$  trong tương lai
- TT khi tham gia hệ thống, phải thêm tất cả các cung đòi hỏi tương ứng vào đồ thị
  - Khi  $P_i$  yêu cầu  $R_j$ , cung đòi hỏi  $P_i \rightarrow R_j$  chuyển thành cung yêu cầu  $P_i \rightarrow R_j$
  - Khi  $P_i$  giải phóng  $R_j$ , cung sử dụng  $R_j \rightarrow P_i$  chuyển thành cung đòi hỏi  $P_i \rightarrow R_j$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Thuật toán dựa vào đồ thị cung cấp tài nguyên

Thuật toán:

Yêu cầu tài nguyên  $R_j$  của TT  $P_i$  được thỏa mãn chỉ khi việc chuyển cung yêu cầu  $P_i \rightarrow R_j$  thành cung sử dụng  $R_j \rightarrow P_i$  không tạo chu trình trên đồ thị.

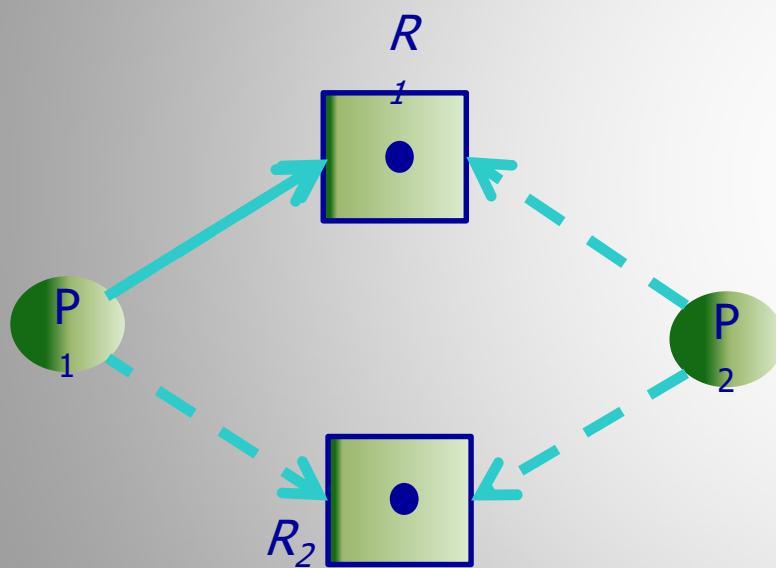
- Không chu trình: Hệ thống an toàn
- Có chu trình: Việc cung cấp tài nguyên đẩy hệ thống vào tình trạng không an toàn

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ



Hệ thống: 2 TT  $P_1, P_2$  và 2 tài nguyên  $R_1, R_2$ , mỗi loại 1 đơn vị

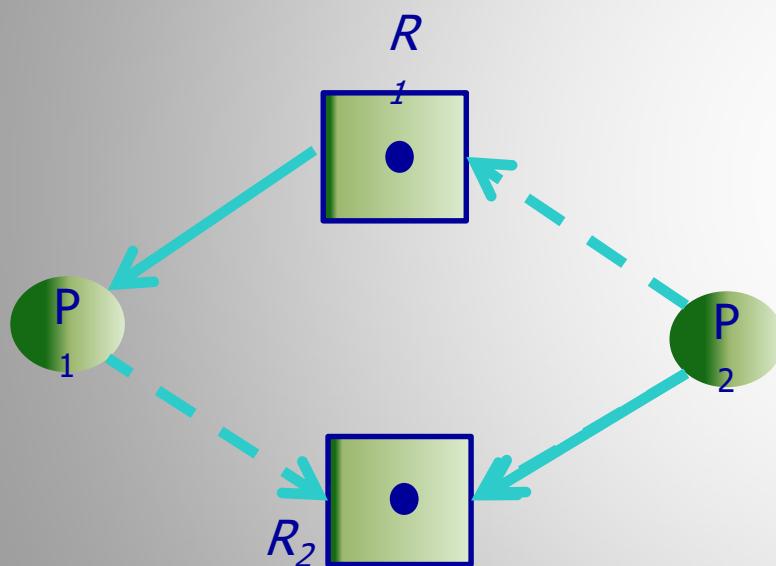
- $P_1$  có thể xin  $R_1, R_2$  trong tương lai
- $P_2$  có thể xin  $R_1, R_2$  trong tương lai
- $P_1$  yêu cầu tài nguyên  $R_1$ 
  - Cung đòi hỏi trở thành cung yêu cầu

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ



Hệ thống: 2 TT  $P_1, P_2$  và 2 tài nguyên  $R_1, R_2$ , mỗi loại 1 đơn vị

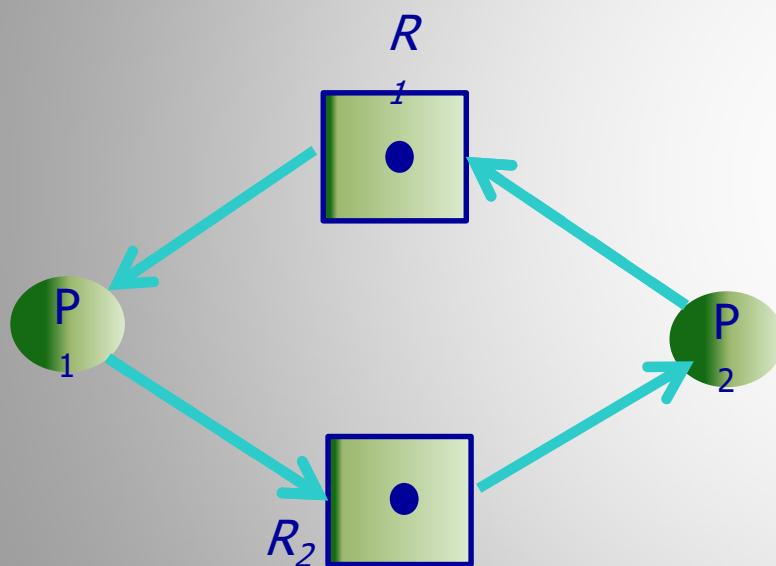
- $P_1$  có thể xin  $R_1, R_2$  trong tương lai
- $P_2$  có thể xin  $R_1, R_2$  trong tương lai
- $P_1$  yêu cầu tài nguyên  $R_1$ 
  - Cung đòi hỏi trở thành cung yêu cầu
- Yêu cầu của  $P_1$  được đáp ứng
  - Cung yêu cầu thành cung sử dụng
- $P_2$  yêu cầu tài nguyên  $R_2 \Rightarrow$  cung đòi hỏi trở thành cung yêu cầu  $P_2 \rightarrow R_2$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ



Hệ thống: 2 TT  $P_1, P_2$  và 2 tài nguyên  $R_1, R_2$ , mỗi loại 1 đơn vị

- $P_1$  có thể xin  $R_1, R_2$  trong tương lai
- $P_2$  có thể xin  $R_1, R_2$  trong tương lai
- $P_1$  yêu cầu tài nguyên  $R_1$ 
  - Cung đòi hỏi trở thành cung yêu cầu
- Yêu cầu của  $P_1$  được đáp ứng
  - Cung yêu cầu thành cung sử dụng
- $P_2$  yêu cầu tài nguyên  $R_2 \Rightarrow$  cung đòi hỏi trở thành cung yêu cầu  $P_2 \rightarrow R_2$ 
  - Nếu đáp ứng
    - ⇒ Cung yêu cầu thành cung sử dụng
    - ⇒ Khi  $P_1$  yêu cầu  $R_2 \Rightarrow P_1$  phải đợi
    - ⇒ Khi  $P_2$  yêu cầu  $R_1 \Rightarrow P_2$  phải đợi

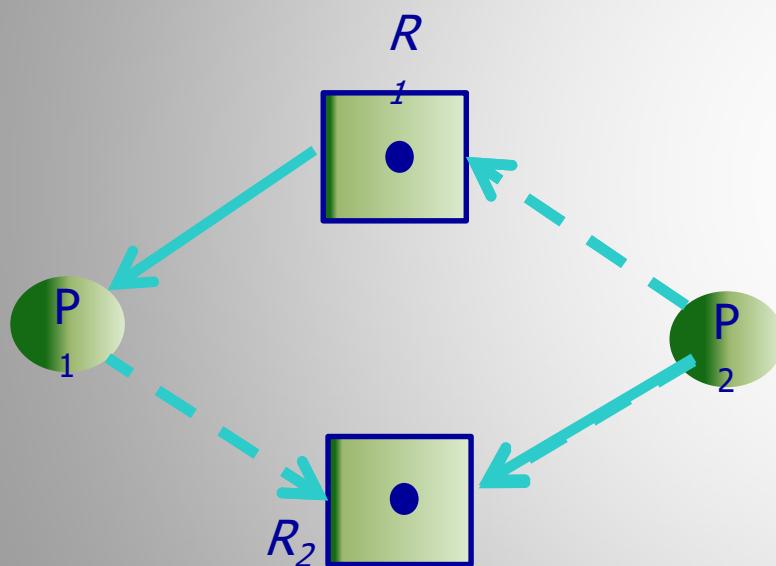
**Hệ thống bẽ tắc**

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ



Hệ thống: 2 TT  $P_1, P_2$  và 2 tài nguyên  $R_1, R_2$ , mỗi loại 1 đơn vị

- $P_1$  có thể xin  $R_1, R_2$  trong tương lai
  - $P_2$  có thể xin  $R_1, R_2$  trong tương lai
  - $P_1$  yêu cầu tài nguyên  $R_1$ 
    - Cung đòi hỏi trở thành cung yêu cầu
  - Yêu cầu của  $P_1$  được đáp ứng
    - Cung yêu cầu thành cung sử dụng
  - $P_2$  yêu cầu tài nguyên  $R_2 \Rightarrow$  cung đòi hỏi trở thành cung yêu cầu  $P_2 \rightarrow R_2$ 
    - Nếu đáp ứng
      - ⇒ Cung yêu cầu thành cung sử dụng
      - ⇒ Khi  $P_1$  yêu cầu  $R_2 \Rightarrow P_1$  phải đợi
      - ⇒ Khi  $P_2$  yêu cầu  $R_1 \Rightarrow P_2$  phải đợi
- Hệ thống bẽ tắc**
- Yêu cầu của  $P_2$  không được đáp ứng

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Thuật toán người quản lý nhà băng: Giới thiệu

- Thích hợp cho các hệ thống gồm các kiểu TN có nhiều đơn vị
- 1 TT mới xuất hiện trong hệ thống cần khai báo số đơn vị lớn nhất của mỗi kiểu TN sẽ sử dụng
  - Không được vượt quá tổng số TN của hệ thống
- Khi 1 TT yêu cầu TN, hệ thống kiểm tra liệu đáp ứng cho yêu cầu hệ thống có còn an toàn không
  - Nếu hệ thống vẫn an toàn  $\Rightarrow$  Cung cấp TN cho yêu cầu
  - Nếu hệ thống không an toàn  $\Rightarrow$  TT phải đợi
- Thuật toán cẩn
  - Các cấu trúc dữ liệu biểu diễn t/thái phân phôi TN
  - Thuật toán kiểm tra tình trạng an toàn của hệ thống
  - Thuật toán yêu cầu TN

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Các cấu trúc dữ liệu I

Hệ thống

n số TT trong hệ thống

m số kiểu TN trong hệ thống

Các cấu trúc dữ liệu

Available Vector chiều dài m cho biết số đơn vị TN sẵn có trong hệ thống. (**Available[3] = 8**  $\Rightarrow ?$ )

Max Ma trận n \* m cho biết số lượng lớn nhất mỗi kiểu TN của từng TT . (**Max[2,3] = 5**  $\Rightarrow ?$ )

Allocation Ma trận n \* m cho biết số lượng mỗi kiểu TN đã cấp cho TT. (**Allocation[2,3] = 2**  $\Rightarrow ?$ )

Need Ma trận n \* m chỉ ra số lượng mỗi kiểu TN còn cần đến của từng TT. **Need[2,3] = 3**  $\Rightarrow ?$ )

**Need[i][j] = Max[i][j] - Allocation[i][j]**

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

## Các cấu trúc dữ liệu I

### Quy ước

- $X, Y$  là các vector độ dài  $n$ 
  - $X \leq Y \Leftrightarrow X[i] \leq Y[i] \forall i = 1, 2, \dots, n$
- Các dòng của ma trận *Max, Need, Allocation* được xử lý như các vector
- Thuật toán tính toán trên các vector
- Các cấu trúc cục bộ
  - Work vector độ dài  $m$  cho biết mỗi TN còn bao nhiêu
  - Finish vector độ dài  $n$ , kiểu logic cho biết TT có chắc chắn
  - kết thúc không

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Thuật toán kiểm tra An toàn

```
BOOL Safe(Current Resource-Allocation State){  
    Work←Available  
    for (i : 1 → n) Finish[i]←false  
    flag← true  
    While(flag){  
        flag←false  
        for (i : 1 → n)  
            if(Finish[i]=false AND Need[i] ≤Work){  
                Finish[i]← true  
                Work ← Work+Allocation[i]  
                flag← true  
            } //endif  
    } //endwhile  
    for (i : 1 → n) if (Finish[i]=false) return false  
    return true;  
} //End function
```

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ minh họa

- Xét hệ thống gồm 5 TT  $P_0, P_1, P_2, P_3, P_4$  và 3 TN  $R_0, R_1, R_2$ 
  - TN  $R_0$  có 10 đơn vị,  $R_1$  có 5 đơn vị,  $R_2$  có 7 đơn vị
- Yêu cầu TN lớn nhất và lượng TN đã cấp của mỗi TT

	<b>R0</b>	<b>R1</b>	<b>R2</b>
$P_0$	7	5	3
$P_1$	3	2	2
$P_2$	9	0	2
$P_3$	2	2	2
$P_4$	4	3	3

- Hệ thống có an toàn?

	<b>R0</b>	<b>R1</b>	<b>R2</b>
$P_0$	0	1	0
$P_1$	2	0	0
$P_2$	3	0	2
$P_3$	2	1	1
$P_4$	0	0	2

Allocation

- TT  $P_1$  yêu cầu thêm 1 đơn vị  $R_0$  và 2 đơn vị  $R_2$ ?
- TT  $P_4$  yêu cầu thêm 3 đơn vị  $R_0$  và 3 đơn vị  $R_1$ ?
- TT  $P_0$  yêu cầu thêm 2 đơn vị  $R_1$ . Cung cấp?

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ minh họa : Kiểm tra tính an toàn

- Số tài nguyên còn sẵn trong hệ thống  $Available(R_0, R_1, R_2) = (3, 3, 2)$
- Yêu cầu còn lại của mỗi TT ( $Need = Max - Allocation$ )

	R <sub>0</sub>	R <sub>1</sub>	R <sub>2</sub>
P <sub>0</sub>	7	5	3
P <sub>1</sub>	3	2	2
P <sub>2</sub>	9	0	2
P <sub>3</sub>	2	2	2
P <sub>4</sub>	4	3	3
Max			

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1
Need			

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa : Kiểm tra tính an toàn

- Số tài nguyên còn sẵn trong hệ thống  $Available(R_0, R_1, R_2) = (3, 3, 2)$

	$R_0$	$R_1$	$R_2$
$P_0$	7	5	3
$P_1$	3	2	2
$P_2$	9	0	2
$P_3$	2	2	2
$P_4$	4	3	3
Max			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2
Allocation			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1
Need			

Tiến trình	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$
Finish	F	T	T	F	F
Work	$(3, 3, 2)$				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa : Kiểm tra tính an toàn

- Số tài nguyên còn sẵn trong hệ thống  $Available(R_0, R_1, R_2) = (3, 3, 2)$

	$R_0$	$R_1$	$R_2$
$P_0$	7	5	3
$P_1$	3	2	2
$P_2$	9	0	2
$P_3$	2	2	2
$P_4$	4	3	3
Max			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2
Allocation			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1
Need			

Tiến trình	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$
Finish	T	T	T	T	T
Work	(10,5,7)	(7,5,5)	(7,4,5)		

Hệ thống an toàn  
(P1, P3, P4, P0, P2)

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

### Thuật toán yêu cầu tài nguyên

- Request[i] Vector yêu cầu tài nguyên của TT  $P_i$ 
  - Request[3,2] = 2: TT  $P_3$  yêu cầu 2 đơn vị tài nguyên  $R_2$
- Khi  $P_i$  yêu cầu TN, hệ thống thực hiện
  - ① if(Request[i]>Need[i])  
**Error**(Yêu cầu vượt quá khai báo TN)
  - ② if(Request[i]>Available)  
**Block**(Không đủ TN, TT phải đợi)
  - ③ Thiết lập trạng thái phân phối TN mới cho hệ thống
    - Available = Available - Request[i]
    - Allocation[i] = Allocation[i] + Request[i]
    - Need[i] = Need[i] - Request[i]
  - ④ Phân phối TN dựa trên kết quả kiểm tra tính an toàn của trạng thái phân phối TN mới  
**if**(Safe(New Resource Allocation State))  
    Phân phối cho  $P_i$  theo yêu cầu  
**else**  
    TT  $P_i$  phải đợi  
    Khôi phục lại trạng thái cũ (Available, Allocation,Need)

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- $\text{Request}[1] \leq \text{Available} ((1, 0, 2) \leq (3, 3, 2)) \Rightarrow \text{Có thể cung cấp}$
- Nếu cung cấp :  $\text{Available} = (2, 3, 0)$

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	2
P3	2	1	1
P4	0	0	2
Allocation			



	R0	R1	R2
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2
Allocation at t1			

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- Request[1] ≤ Available  $((1, 0, 2) \leq (3, 3, 2)) \Rightarrow$  Có thể cung cấp
- Nếu cung cấp : Available =  $(2, 3, 0)$

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	7	4	3
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1
Need			



	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1
Need at t1			

	$R_0$	$R_1$	$R_2$
$P_0$	7	5	3
$P_1$	3	2	2
$P_2$	9	0	2
$P_3$	2	2	2
$P_4$	4	3	3
Max			

## Chương 2 Quản lí tiến trình

## 5. Bé tắc và xử lí bé tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	R0	R1	R2		R0	R1	R2
P0	0	1	0	P0	7	4	3
P1	3	0	2	P1	0	2	0
P2	3	0	2	P2	6	0	0
P3	2	1	1	P3	0	1	1
P4	0	0	2	P4	4	3	1
Allocation				Need			
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>		
Finish	F	F	F	F	F		
Work			(2,3,0)				

## Chương 2 Quản lí tiến trình

## 5. Bé tắc và xử lí bé tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa : P<sub>1</sub> yêu cầu (1, 0, 2)

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	<b>R0</b>	<b>R1</b>	<b>R2</b>		<b>R0</b>	<b>R1</b>	<b>R2</b>	
P0	0	1	0		P0	7	4	3
P1	3	0	2		P1	0	2	0
P2	3	0	2		P2	6	0	0
P3	2	1	1		P3	0	1	1
P4	0	0	2		P4	4	3	1
Allocation				Need				
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>			
Finish	F		F	F	F			
Work				(2,3,0)				

## Chương 2 Quản lí tiến trình

## 5. Bé tắc và xử lí bé tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	R0	R1	R2		R0	R1	R2
P0	0	1	0	P0	7	4	3
P1	3	0	2	P1	0	2	0
P2	3	0	2	P2	6	0	0
P3	2	1	1	P3	0	1	1
P4	0	0	2	P4	4	3	1
Allocation				Need			
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>		
Finish	F	T	F	F	F		
Work				(5,3,2)			

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- $\text{Request}[1] \leq \text{Available} ((1, 0, 2) \leq (3, 3, 2)) \Rightarrow \text{Có thể cung cấp}$
- Nếu cung cấp :  $\text{Available} = (2, 3, 0)$

	R0	R1	R2
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

	R0	R1	R2
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Allocation	Need				
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	F	T	F	F	F
Work	(5,3,2)				

## Chương 2 Quản lí tiến trình

## 5. Bé tắc và xử lí bé tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	R0	R1	R2		R0	R1	R2
P0	0	1	0	P0	7	4	3
P1	3	0	2	P1	0	2	0
P2	3	0	2	P2	6	0	0
P3	2	1	1	P3	0	1	1
P4	0	0	2	P4	4	3	1
Allocation				Need			
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>		
Finish	F	T	F	T	F		
Work						(7,4,3)	

## Chương 2 Quản lí tiến trình

## 5. Bẽ tắc và xử lí bẽ tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	<b>R0</b>	<b>R1</b>	<b>R2</b>		<b>R0</b>	<b>R1</b>	<b>R2</b>	
P0	0	1	0		P0	7	4	3
P1	3	0	2		P1	0	2	0
P2	3	0	2		P2	6	0	0
P3	2	1	1		P3	0	1	1
P4	0	0	2		P4	4	3	1
Allocation				Need				
Tiến trình	$P_0$	$P_1$	$P_2$	$P_3$	$P_4$			
Finish	F	T	F	T	T			
Work				(7,4,5)				

## Chương 2 Quản lí tiến trình

## 5. Bé tắc và xử lí bé tắc

## 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu (1, 0, 2)

- Request[1] ≤ Available ( $(1, 0, 2) \leq (3, 3, 2)$ )  $\Rightarrow$  Có thể cung cấp
  - Nếu cung cấp : Available = ( 2 , 3 , 0 )

	<b>R0</b>	<b>R1</b>	<b>R2</b>		<b>R0</b>	<b>R1</b>	<b>R2</b>	
P0	0	1	0		P0	7	4	3
P1	3	0	2		P1	0	2	0
P2	3	0	2		P2	6	0	0
P3	2	1	1		P3	0	1	1
P4	0	0	2		P4	4	3	1
Allocation				Need				
Tiến trình	P <sub>0</sub>		P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>		
Finish	T	T		F	T	T		
Work				(7,5,5)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

Ví dụ minh họa :  $P_1$  yêu cầu  $(1, 0, 2)$

- $\text{Request}[1] \leq \text{Available} ((1, 0, 2) \leq (3, 3, 2)) \Rightarrow \text{Có thể cung cấp}$
- Nếu cung cấp :  $\text{Available} = (2, 3, 0)$

	R0	R1	R2
P0	0	1	0
P1	3	0	2
P2	3	0	2
P3	2	1	1
P4	0	0	2

	R0	R1	R2
P0	7	4	3
P1	0	2	0
P2	6	0	0
P3	0	1	1
P4	4	3	1

Allocation	Need				
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	T	T	T	T
Work	(10,5,7)				

Yêu cầu được chấp nhận

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.5 Phòng tránh bẽ tắc

#### Ví dụ minh họa : (tiếp tục)

- TT P<sub>4</sub> yêu cầu thêm 3 đơn vị R<sub>0</sub> và 3 đơn vị R<sub>2</sub>
  - Request[4] = (3, 0, 3)
  - Available = (2, 3, 0)

⇒ Không đủ tài nguyên, P<sub>4</sub> phải đợi
  
- TT P<sub>0</sub> yêu cầu thêm 2 đơn vị R<sub>1</sub>
  - Request[0] ≤ Available ((0, 2, 0) ≤ (2, 3, 0)) ⇒ Có thể cung cấp
  - Nếu cung cấp : Available = (2, 1, 0)
  - Thực hiện thuật toán an toàn
    - ⇒ Tất cả các TT đều có thể không kết thúc
    - ⇒ Nếu chấp nhận, hệ thống rơi vào trạng thái không an toàn

⇒ Đủ tài nguyên nhưng không cung cấp. P<sub>0</sub> phải đợi

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.6 Nhận biết và khắc phục

- Khái niệm bẽ tắc
- Điều kiện xảy ra bẽ tắc
- Các phương pháp xử lý bẽ tắc
- Phòng ngừa bẽ tắc
- Phòng tránh bẽ tắc
- **Nhận biết và khắc phục**

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Giới thiệu

- Nguyên tắc
  - Không áp dụng các biện pháp phòng ngừa hoặc phòng tránh, để cho bẽ tắc xảy ra
  - Định kỳ kiểm tra xem bẽ tắc có đang xảy ra không. Nếu có tìm cách khắc phục
  - Để thực hiện, hệ thống phải cung cấp
    - Thuật toán xác định hệ thống đang bẽ tắc không
    - Biện pháp kỹ thuật chữa bẽ tắc
- Nhận biết bẽ tắc
  - Thuật toán dựa trên đồ thị cung cấp TN
  - Thuật toán chỉ ra bẽ tắc tổng quát
- Khắc phục bẽ tắc
  - Kết thúc TT
  - Trưng dụng tài nguyên

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

##### Thuận toán dựa trên đồ thị cung cấp tài nguyên

I

- Áp dụng khi mỗi TN trong hệ thống có 1 đơn vị
- K/tracker hệ thống có bẽ tắc bằng cách kiểm tra chu trình trên đồ thị
  - Nếu trên đồ thị có chu trình, hệ thống đang bẽ tắc
- Định kỳ gọi tới các thuật toán kiểm tra chu trình trên đồ thị
  - Thuật toán đòi hỏi  $n^2$  thao tác ( $n$ : số đỉnh của đồ thị)

## Thuận toán dựa trên đồ thị cung cấp tài nguyên

II

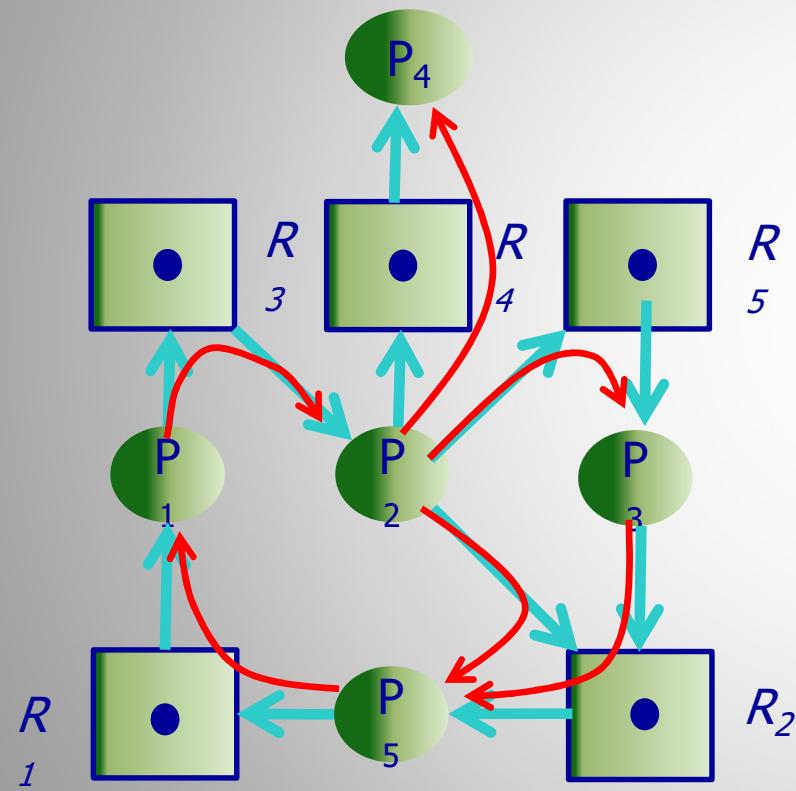
- Sử dụng đồ thị chờ đợi - phiên bản thu gọn của đồ thị cung cấp tài nguyên
  - Chỉ có các đỉnh dạng TT
  - Cung chờ đợi  $P_i \rightarrow P_j$ : TT  $P_i$  đang đợi TT  $P_j$  giải phóng tài nguyên  $P_i$  cần
  - Cung chờ đợi  $P_i \rightarrow P_j$  tồn tại trên đồ thị đợi khi và chỉ khi trên đồ thị phân phối tài nguyên tương ứng tồn tại đồng thời cung yêu cầu  $P_i \rightarrow R$  và cung sử dụng  $R \rightarrow P_j$

## Chương 2 Quản lý tiến trình

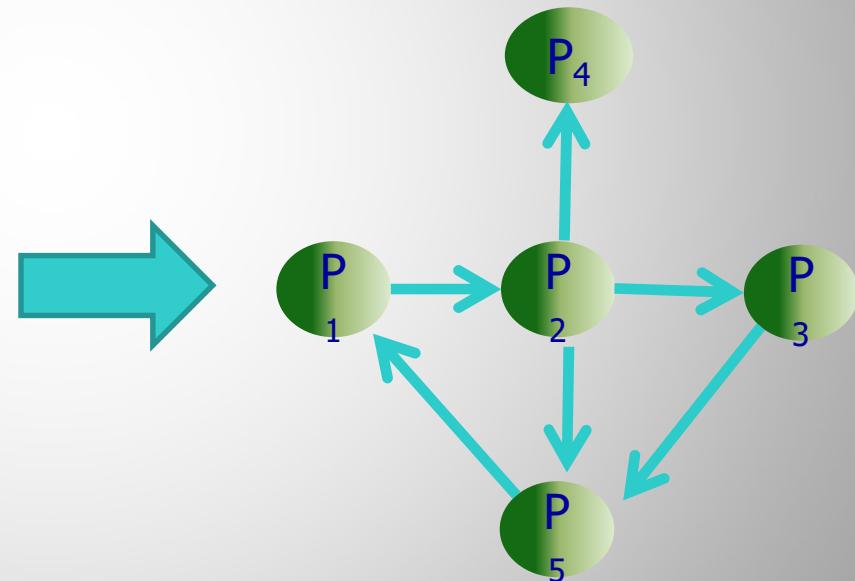
### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Đồ thị chờ đợi: Ví dụ



#### Đồ thị chờ đợi



## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Thuật toán chỉ ra bẽ tắc tổng quát : Giới thiệu

- Áp dụng: hệ thống có các **kiểu tài nguyên** gồm **nhiều đơn vị**
- Thuật toán: tương tự thuật toán **người quản lý nhà băng**
- Các cấu trúc dữ liệu
  - Available Vector độ dài m: Tài nguyên sẵn có
  - Allocation Ma trận n \* m: Tài nguyên đã cấp
  - Request Ma trận n \* m: Tài nguyên yêu cầu
- Các cấu trúc cục bộ
  - Work Vector độ dài m cho biết TN hiện đang có
  - Finish Vector độ dài n cho biết TT có thể kết thúc không
- **Các qui ước**
  - Quan hệ  $\leq$  giữa các Vector
  - Xử lý các **dòng** ma trận n \* m như các vector

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Thuật toán chỉ ra bẽ tắc tổng quát

```
BOOL Deadlock(Current Resource-Allocation State){  
    Work←Available  
    for (i : 1 → n)  
        if(Allocation[i]≠0) Finish[i]←false  
        else Finish[i] ← true;      //Allocation= 0 không nằm trong chu trình đợi  
        flag← true  
        while(flag){  
            flag←false  
            for (i : 1 → n)  
                if (Finish[i]=false AND Request[i] ≤Work){  
                    Finish[i]← true  
                    Work ← Work+Allocation[i]  
                    flag← true  
                } //endif  
            } //endwhile  
            for (i : 1 → n) if (Finish[i]=false) return true;  
            return false;          //Finish[i] = false, tiến trình Pi đang bị bẽ tắc  
} //End function
```

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

- 5 tiến trình  $P_0, P_1, P_2, P_3, P_4$ ; 3 tài nguyên  $R_0, R_1, R_2$ 
  - Tài nguyên  $R_0$  có 7 đơn vị,  $R_1$  có 2 đơn vị,  $R_2$  có 6 đơn vị
- Trạng thái cung cấp tài nguyên tại thời điểm  $t_0$

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Allocation

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2

Request



Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	F	F	F	F	F
Work	(0,0,0)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	F	F	F
Work	(0,1,0)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	T	F	F
Work	(3,1,3)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	T	T	F
Work	(5,2,4)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	T	T	F
Work	(5,2,4)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	T	T	T	F
Work	(7,2,4)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	0	0
P1	2	0	2
P2	0	0	0
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	T	T	T	T
Work	(7,2,6)				

Hệ thống không bẽ tắc  
 $(P_0, P_2, P_3, P_1, P_4)$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

- tại thời điểm  $t_1$ :  $P_2$  yêu cầu thêm 1 đơn vị tài nguyên  $R_2$
- Trạng thái cung cấp tài nguyên

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	6	0	2
Request			

- Tài nguyên hiện có  $(R_0, R_1, R_2) = (0, 0, 0)$

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2
Allocation			
Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>
Finish	F	F	F
Work	(0,0,0)		

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	6	0	2
Request			

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Allocation

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	6	0	2

Request

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	F	F	F
Work	(0,1,0)				

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

	R0	R1	R2
P0	0	1	0
P1	2	0	0
P2	3	0	3
P3	2	1	1
P4	0	0	2

Allocation

Tiến trình	P <sub>0</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>
Finish	T	F	F	F	F
Work	(0,1,0)				

	R0	R1	R2
P0	0	0	0
P1	2	0	2
P2	0	0	1
P3	1	0	0
P4	6	0	2

Request

P<sub>0</sub> có thể kết thúc nhưng hệ thống đang bẽ tắc.

Các tiến trình đang chờ đợi lẫn nhau (P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>)

Chương 2 Quản lý tiến trình

5.Bê tắc và xử lí bê tắc

5.6 Nhận biết và khắc phục

Khắc phục bê tắc: Phương pháp kết thúc tiến trình

**Nguyên tắc:** Hủy bỏ các TT đang trong tình trạng bê tắc và lấy lại tài nguyên đã cấp cho TT bị hủy bỏ

- Hủy bỏ tất cả các TT
- Hủy bỏ lần lượt TT cho tới khi bê tắc không xảy ra

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Khắc phục bẽ tắc: Phương pháp kết thúc tiến trình

- Hủy bỏ tất cả các TT
  - Nhanh chóng loại bỏ bẽ tắc
  - Quá tốn kém
    - Các TT bị hủy bỏ có thể gần kết thúc
- Hủy bỏ lần lượt TT cho tới khi bẽ tắc không xảy ra
  - Sau khi hủy bỏ, phải kiểm tra xem bẽ tắc còn tồn tại không
    - Thuật toán kiểm tra bẽ tắc có độ phức tạp  $m * n^2$
  - Cần chỉ ra thứ tự TT bị hủy bỏ để phá vỡ bẽ tắc
    - Độ ưu tiên.
    - Thời gian tồn tại, thực hiện
    - Tài nguyên đang chiếm giữ, còn cần để kết thúc
    - . . .

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

Khắc phục bẽ tắc: Phương pháp kết thúc tiến trình

##### ● Văn đề hủy bỏ TT

- TT đang cập nhật file ⇒ File không hoàn chỉnh
- TT sử dụng máy in ⇒ Reset trạng thái máy in

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

**Khắc phục bẽ tắc:** Phương pháp trưng dụng tài nguyên

**Nguyên tắc:** Trưng dụng liên tục một vài tài nguyên từ một số TT đang bẽ tắc cho các TT khác đến khi bẽ tắc được hủy bỏ

Các vấn đề cần quan tâm

- ① Lựa chọn nạn nhân (victim)
- ② Quay lui (Rollback)
- ③ Đói tài nguyên (Starvation)

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

### Khắc phục bẽ tắc: Phương pháp trưng dụng tài nguyên

#### ① Lựa chọn nạn nhân (victim)

- Tài nguyên nào và tiến trình nào ?
- Trật tự trưng dụng để chi phí nhỏ nhất
- Lượng tài nguyên nắm giữ, thời gian sử dụng. . .

#### ② Quay lui (Rollback)

- Quay lui tới một trạng thái an toàn trước đó và bắt đầu lại
- Yêu cầu lưu giữ thông tin trạng thái của TT đang thực hiện

#### ③ Đói tài nguyên (Starvation)

- 1 TT bị trưng dụng quá nhiều lần ⇒ chờ đợi vô hạn
- Giải pháp: ghi lại số lần bị trưng dụng

## Chương 2 Quản lý tiến trình

### 5.Bê tắc và xử lý bê tắc

#### 5.6 Nhận biết và khắc phục

Cách xử lý bê tắc khác ?



## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lý bẽ tắc

#### Tổng kết

- Bẽ tắc là tình trạng 2 hay nhiều TT cùng chờ đợi độc lập 1 sự kiện chỉ có thể xảy ra bởi sự hoạt động của các TT đang đợi
- Bẽ tắc xảy ra khi hội đủ 4 điều kiện
  - Tồn tại tài nguyên găng
  - Phải chờ đợi trước khi vào đoạn găng
  - Không tồn tại hệ thống phân phõi lại tài nguyên
  - Tồn tại hiện tượng chờ đợi vòng tròn
- Để xử lý bẽ tắc có 3 lớp thuật toán
  - Phòng ngừa bẽ tắc
    - Tác động vào các điều kiện xảy ra bẽ tắc
  - Dự báo và phòng tránh
    - Ngăn ngừa hệ thống rơi vào tình trạng có thể dẫn đến bẽ tắc
  - Nhận biết và khắc phục
    - Cho phép bẽ tắc xảy ra, chỉ ra bẽ tắc và khắc phục sau

## Chương 2 Quản lý tiến trình

### 5.Bẽ tắc và xử lí bẽ tắc

#### 5.6 Nhận biết và khắc phục

#### Ví dụ minh họa

- 5 tiến trình  $P_0, P_1, P_2, P_3, P_4$ ; 3 tài nguyên  $R_0, R_1, R_2$ 
  - Tài nguyên  $R_0$  có 6 đơn vị,  $R_1$  có 4 đơn vị,  $R_2$  có 7 đơn vị
- Trạng thái cung cấp tài nguyên tại thời điểm  $t_0$

	<b>R0</b>	<b>R1</b>	<b>R2</b>
<b>P0</b>	0	1	1
<b>P1</b>	1	0	0
<b>P2</b>	3	0	2
<b>P3</b>	2	1	1
<b>P4</b>	0	2	2
Allocation			

	<b>R0</b>	<b>R1</b>	<b>R2</b>
<b>P0</b>	0	0	0
<b>P1</b>	2	1	2
<b>P2</b>	0	0	2
<b>P3</b>	1	0	0
<b>P4</b>	6	0	2
Request			