

Chương 2

Tiến trình và Luồng

2.1 Tiến trình - Processes

2.2 Luồng - Threads

2.3 Lập lịch - Scheduling

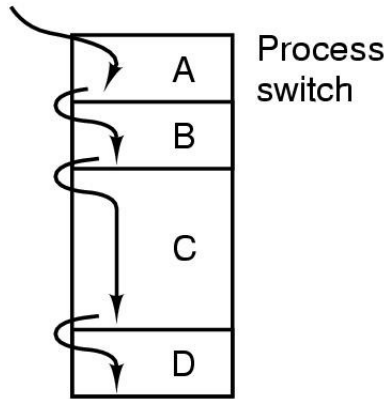
2.4 Giao tiếp liên Tiến trình - Interprocess communication

2.1 Tiến trình

Tiến trình

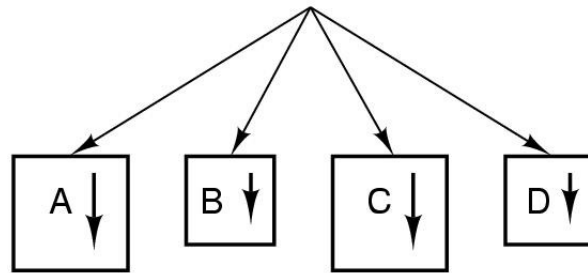
Mô hình Tiến trình

One program counter

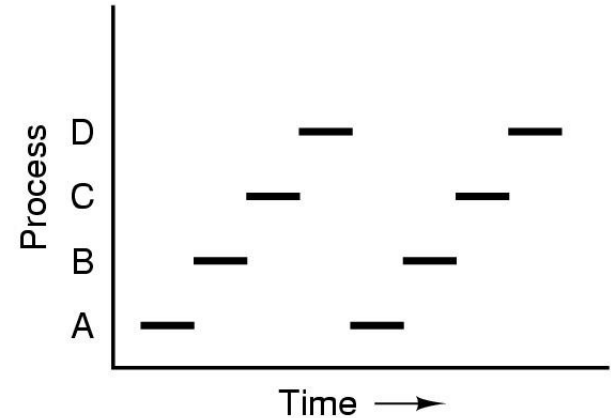


(a)

Four program counters



(b)



(c)

- (a) Đa nhiệm với 4 chương trình
- (b) Mô hình 4 Tiến trình độc lập, tuần tự
- (c) Mỗi thời điểm chỉ có 1 Tiến trình thực thi

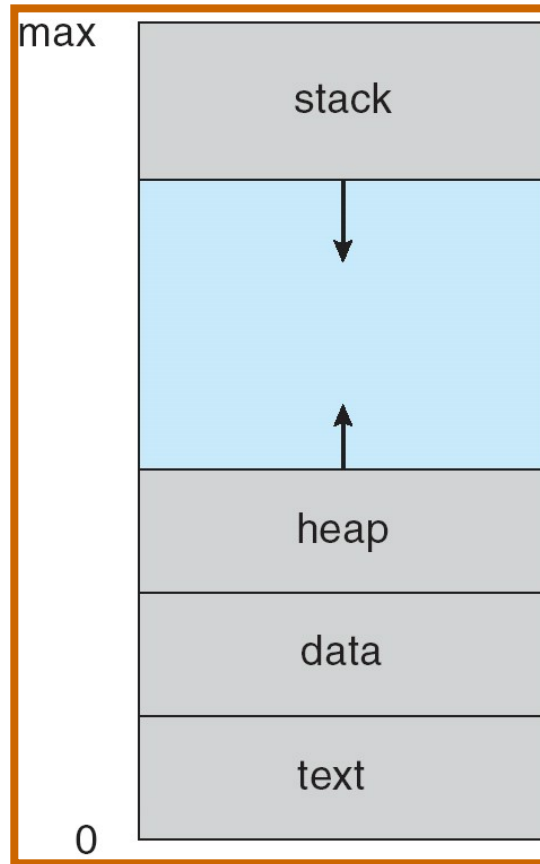
Tiến trình

Khái niệm Tiến trình

- Một Hệ điều hành thực hiện một loạt các chương trình:
 - Hệ thống xử lý lô (batch system) – các công việc
 - Hệ thống chia sẻ thời gian (time-shared systems) – các chương trình của người dùng (user programs) hay các tác vụ (tasks)
- Tiến trình – một chương trình đang thực hiện; thực hiện tiến trình cần phải được tiến hành một cách tuần tự
- Tiến trình cần cấp phát một tập tài nguyên:
 - Không gian địa chỉ (text segment, data segment)
 - CPU (virtual)
 - bộ đếm chương trình (program counter)
 - các thanh ghi (registers)
 - ngăn xếp (stack)
 - Các tài nguyên khác (open files, child processes...)

Tiến trình

Tiến trình trong Bộ nhớ



Tiến trình

Khởi tạo Tiến trình (1)

Các sự kiện làm cho Tiến trình khởi tạo

1. Khởi động hệ điều hành (System initialization)
2. Thực thi của Tiến trình có gọi đến lời gọi hệ thống
3. User yêu cầu tạo ra một tiến trình mới
4. Khởi động một công việc xử lý lô (batch job)

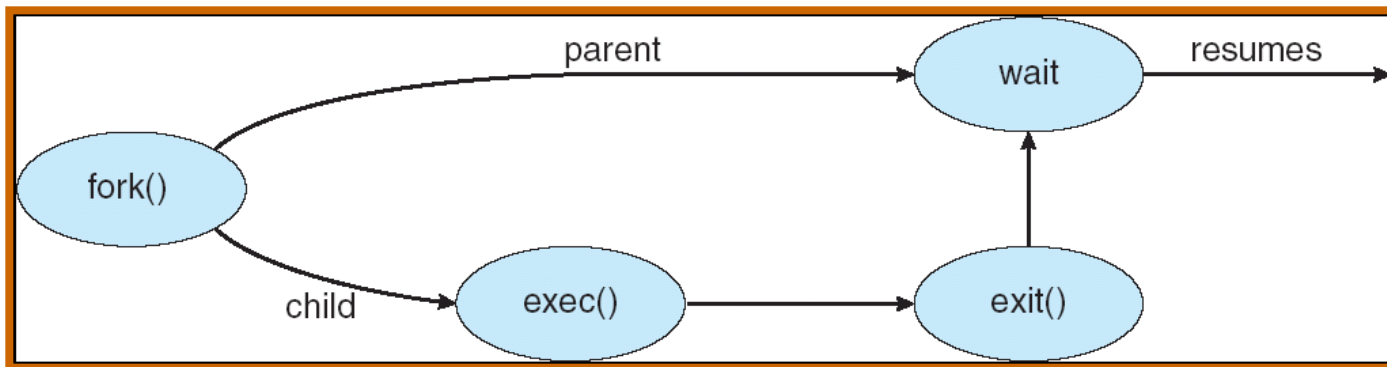
Tiến trình

Khởi tạo Tiến trình (2)

- Không gian địa chỉ
 - Tiến trình con là bản sao của tiến trình cha
 - Ở đó tiến trình con tải chương trình vào
- Ví dụ: Trong UNIX
 - Lời gọi hệ thống **fork** tạo ra tiến trình con
 - Lời gọi hệ thống **exec** được gọi sau **fork** để thay thế không gian bộ nhớ của tiến trình để nạp chương trình mới

Tiến trình

Khởi tạo Tiến trình (3) : Ví dụ



Tiến trình

Kết thúc Tiến trình

Các điều kiện để kết thúc Tiến trình

1. Kết thúc bình thường (tự nguyện)
2. Kết thúc do lỗi (tự nguyện)
3. Kết thúc do lỗi định mệnh (không tự nguyện)
4. Bị hủy (kill) bởi tiến trình khác (không tự nguyện)

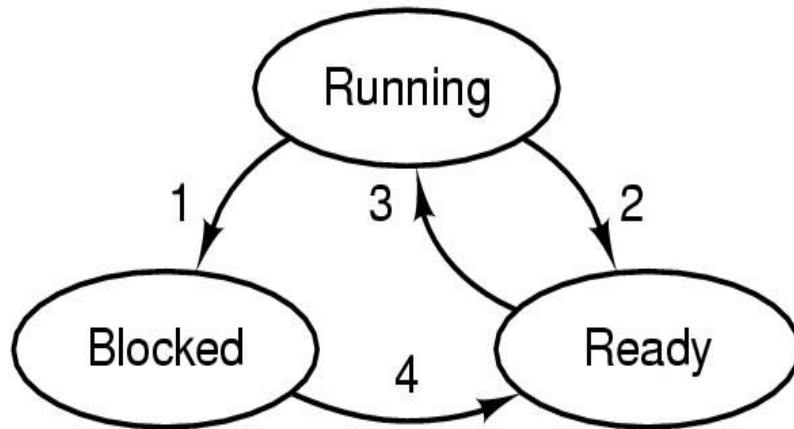
Tiến trình

Phân cấp Tiến trình

- Tiến trình cha tạo ra tiến trình con, tiến trình con tạo ra tiến trình con khác
- Dạng của phân cấp
 - UNIX gọi là nhóm tiến trình "process group"
- Windows không có khái niệm phân cấp tiến trình
 - tất cả các tiến trình được tạo ngang bằng nhau

Tiến trình

Trạng thái Tiến trình (1)

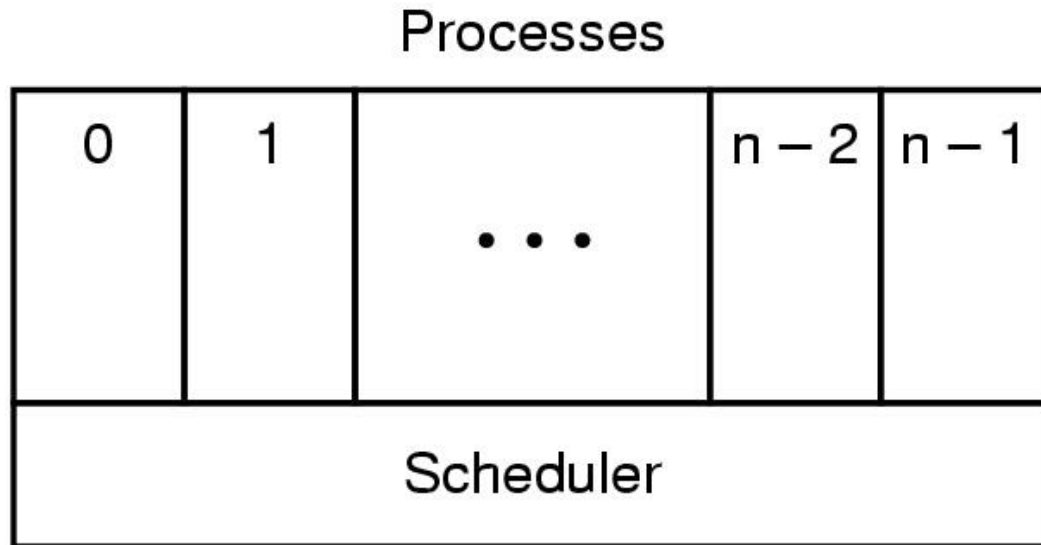


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

- Các trạng thái của Tiến trình
 - đang chạy (running)
 - bị chặn lại (blocked)
 - sẵn sàng (ready)
- Chuyển đổi giữa các trạng thái như hình

Tiến trình

Trạng thái Tiến trình (2)



- Mức thấp nhất của cấu trúc tiến trình của HĐH
 - điều khiển ngắt (handles interrupts), lập lịch (scheduling)
- Bên trên là các tiến trình tuần tự

Tiến trình

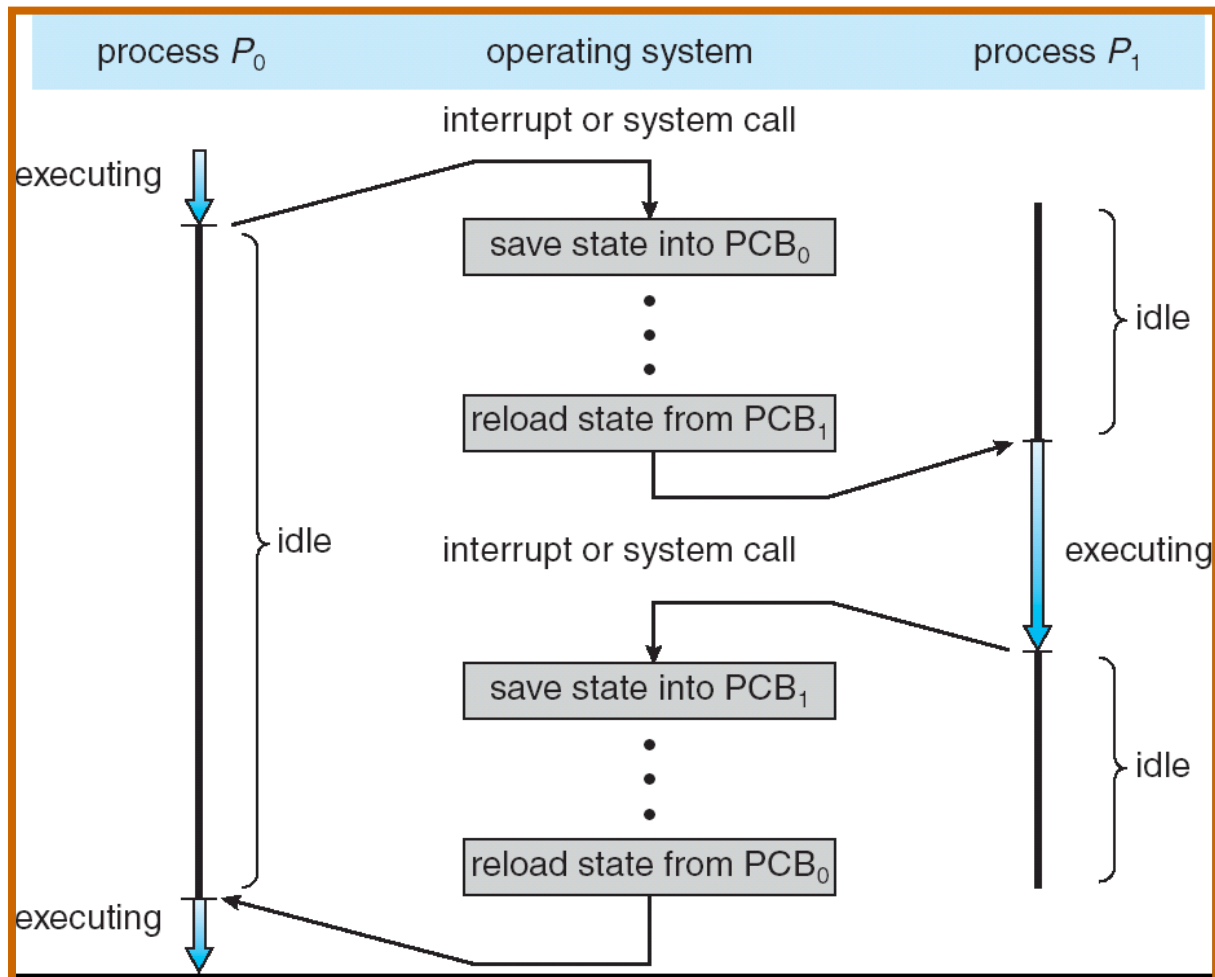
Khởi điều khiển Tiến trình

(PCB - Process Control Block)



Tiến trình

Chuyển đổi ngữ cảnh



Tiến trình

Thực hiện Tiến trình (1)

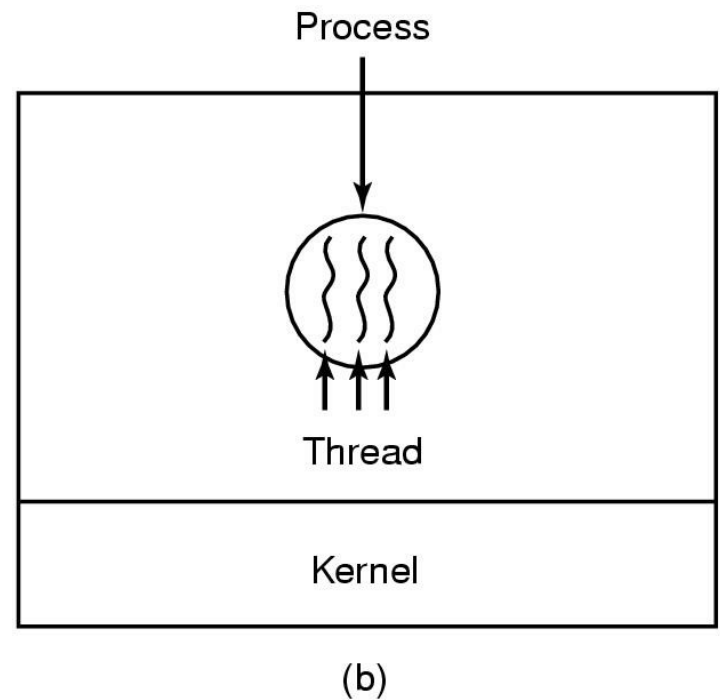
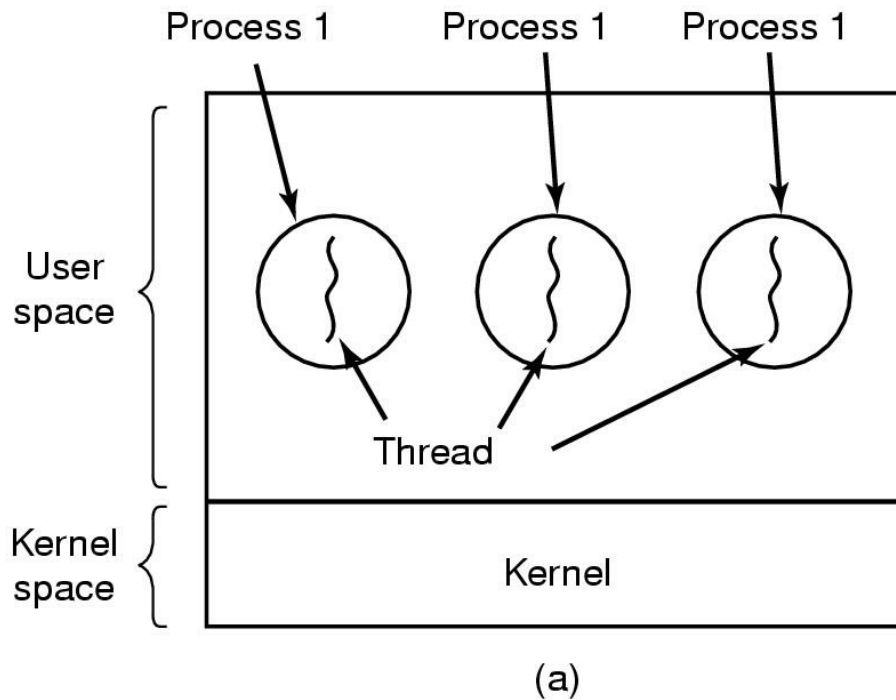
Process management	Memory management	File management
Registers Program counter Program status word Stack pointer Process state Priority Scheduling parameters Process ID Parent process Process group Signals Time when process started CPU time used Children's CPU time Time of next alarm	Pointer to text segment Pointer to data segment Pointer to stack segment	Root directory Working directory File descriptors User ID Group ID

Các trường của một entry của Bảng Tiến trình

2.2 Luồng - Threads

Luồng

Mô hình Luồng



(a) Ba Tiến trình, mỗi tiến trình một Luồng

(b) Một Tiến trình với 3 Luồng

Luồng

Một tiến trình với đơn luồng

- Một tiến trình:
 - Không gian địa chỉ (text section, data section)
 - Đơn luồng xử lý
 - program counter
 - registers
 - Stack
 - Các tài nguyên khác (open files, child processes...)

Luồng

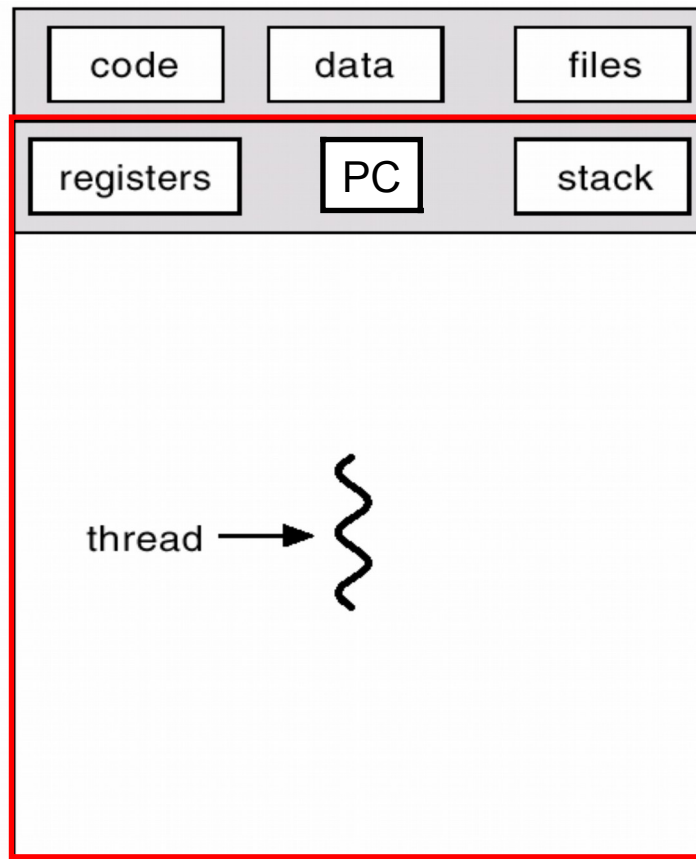
Tiến trình với đa luồng - Multiple threads

Đa luồng xử lý trong cùng một môi trường của tiến trình

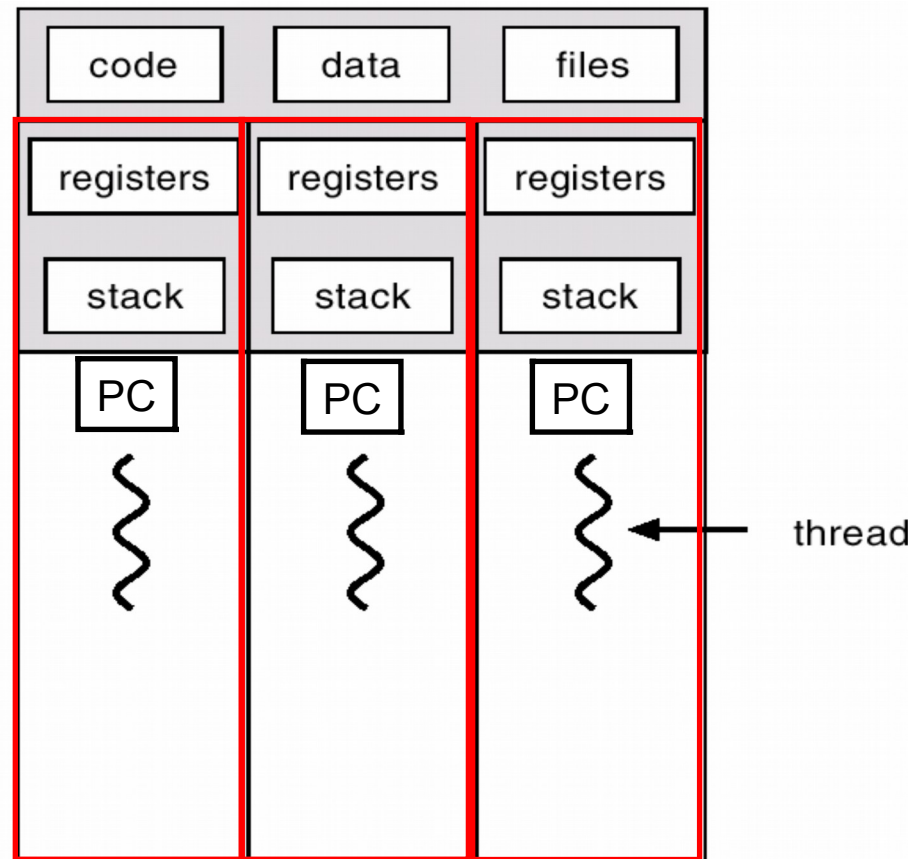
- Không gian địa chỉ (text section, data section)
- Đa luồng xử lý, mỗi luồng có một tập:
 - program counter
 - registers
 - stack
- Các tài nguyên khác (open files, child processes...)

Luồng

Tiến trình đơn luồng và đa luồng (Single and Multithreaded Processes)



single-threaded



multithreaded

Luồng

Các Item chia sẻ và các Item riêng tư

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

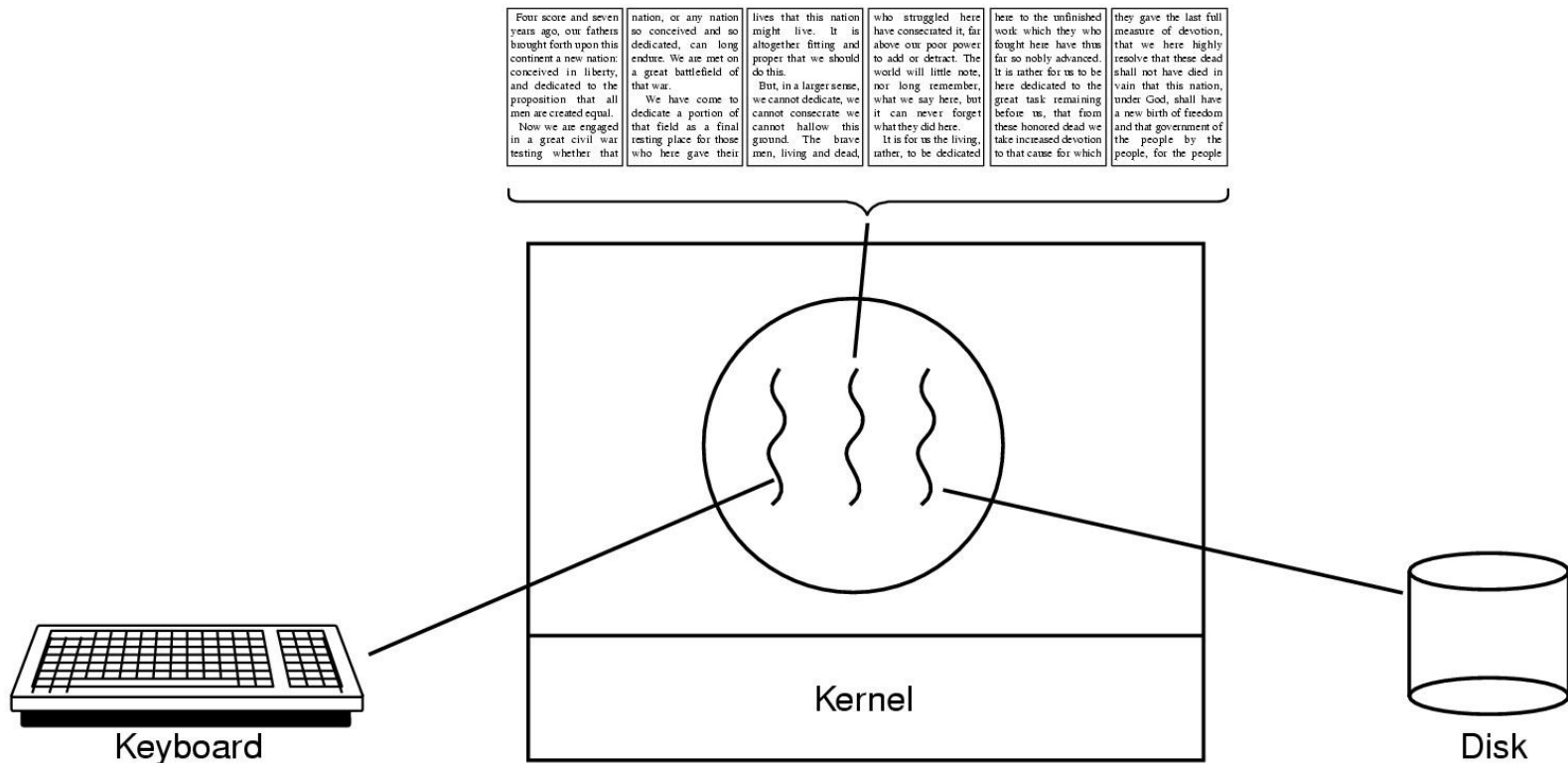
- Các Item chia sẻ các luồng trong một tiến trình
- Các Item riêng trong mỗi tiến trình

Luồng Ích lợi

- Khả năng hồi đáp
- Chia sẻ tài nguyên
- Lợi ích kinh tế
- Phù hợp với các kiến trúc đa bộ xử lý

Luồng

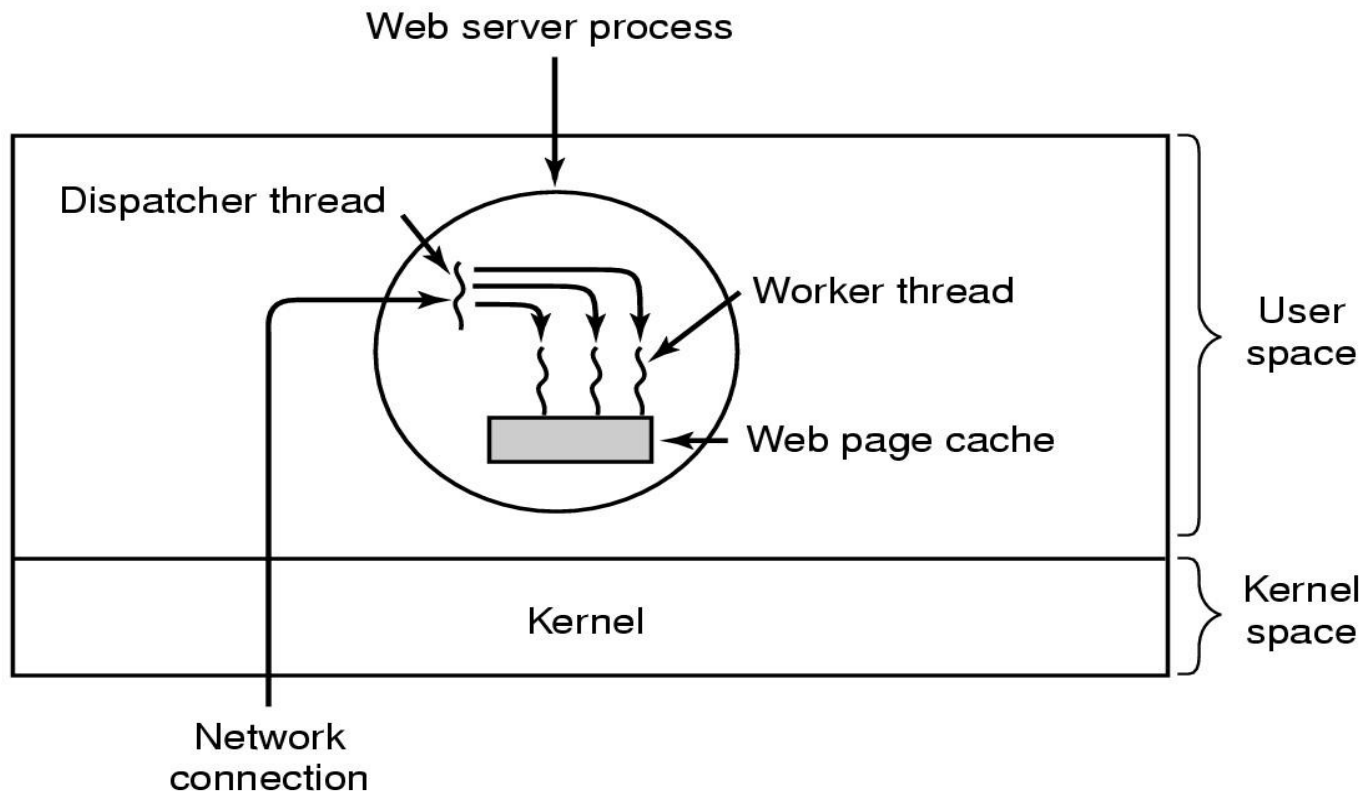
Cách dùng luồng (1)



A word processor with three threads

Luồng

Cách dùng luồng (2)



A multithreaded Web server

Luồng

Cách dùng luồng (3)

```
while (TRUE) {  
    get_next_request(&buf);  
    handoff_work(&buf);  
}
```

(a)

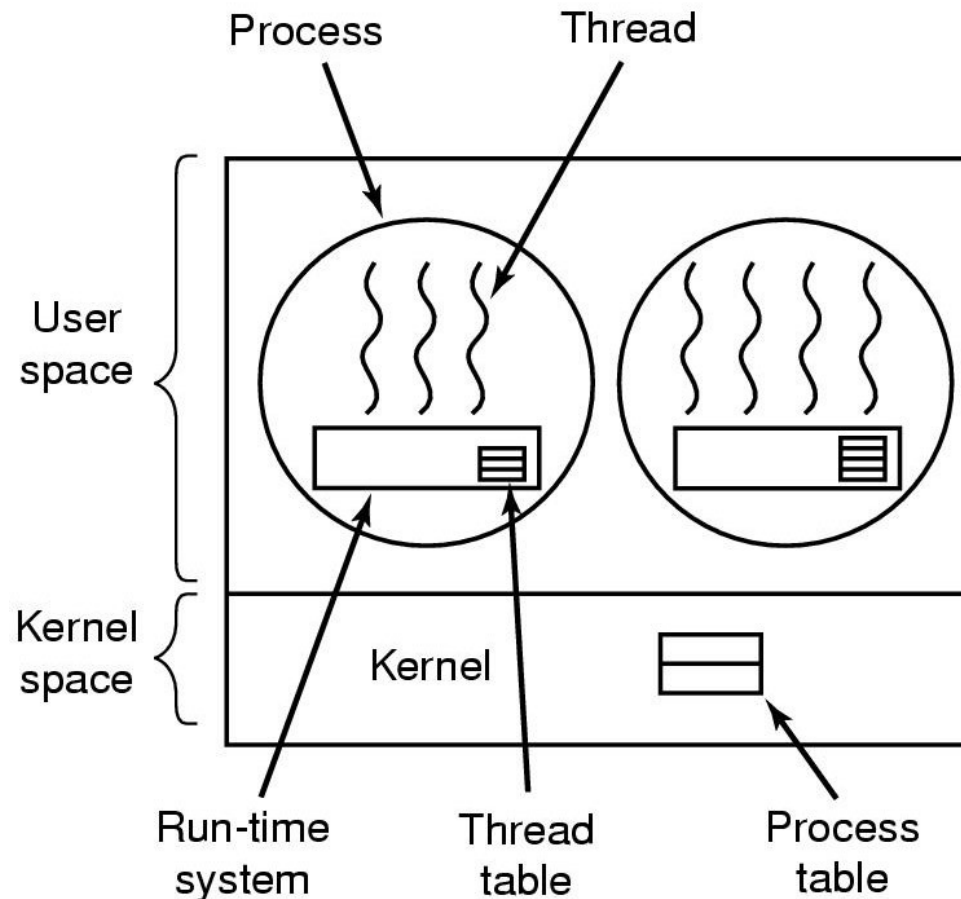
```
while (TRUE) {  
    wait_for_work(&buf)  
    look_for_page_in_cache(&buf, &page);  
    if (page_not_in_cache(&page)  
        read_page_from_disk(&buf, &page);  
    return_page(&page);  
}
```

(b)

- Rough outline of code for previous slide
 - (a) Dispatcher thread
 - (b) Worker thread

Luồng

Thực hiện luồng trong không gian User (1)



Một gói các luồng mức user

Luồng

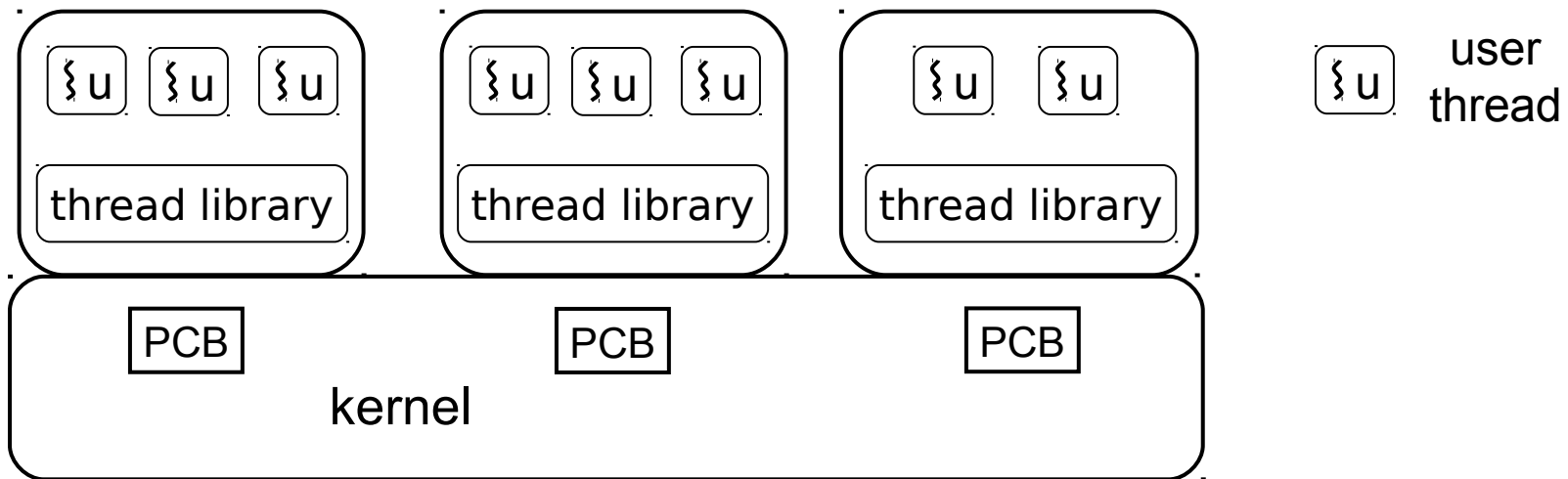
Thực hiện luồng trong không gian User (2)

- Thư viện luồng, (run-time system) trong không gian user
 - `thread_create`
 - `thread_exit`
 - `thread_wait`
 - `thread_yield` (tự nguyện từ bỏ CPU)
- Khối điều khiển luồng (Thread control block-TCB) (Thread Table Entry) lưu trạng thái của luồng user (program counter, registers, stack)
- Kernel không biết sự có hiện diện của luồng user

Luồng

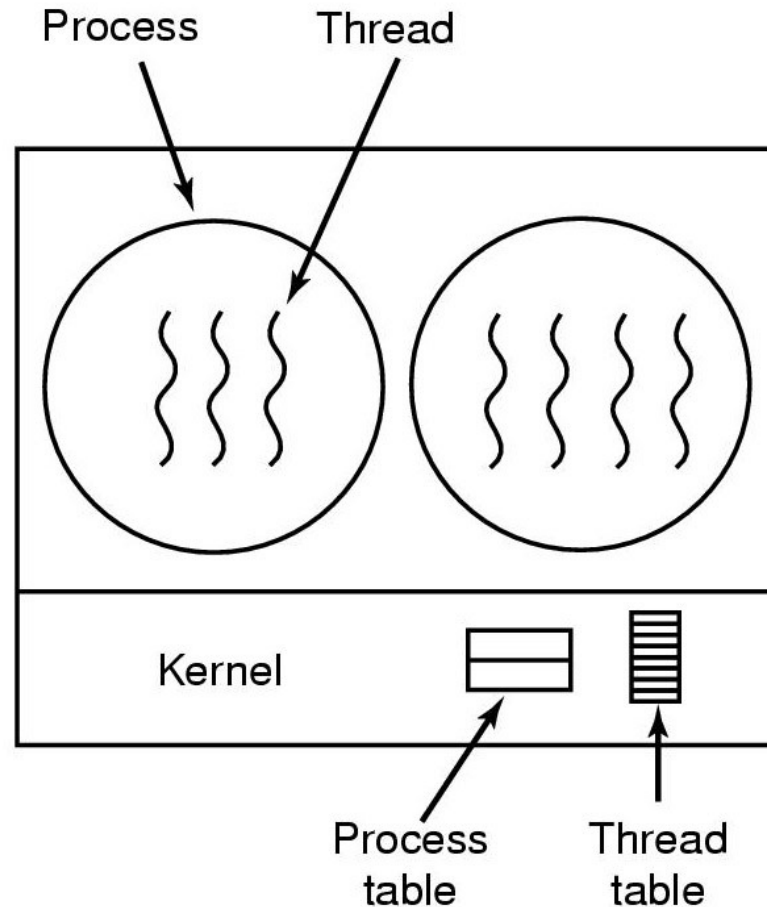
Thực hiện luồng trong không gian User (3)

- Hệ điều hành truyền thống chỉ cung cấp một “kernel thread” và được trình bày bằng PCB cho mỗi tiến trình.
 - Vấn đề blocking*: Nếu một luồng user bị chặn -> luồng kernel cũng bị chặn, -> tất cả các luồng khác trong tiến trình cũng bị chặn.



Luồng

Thực hiện luồng trong Kernel (1)



Một gói các luồng được quản lý bởi kernel

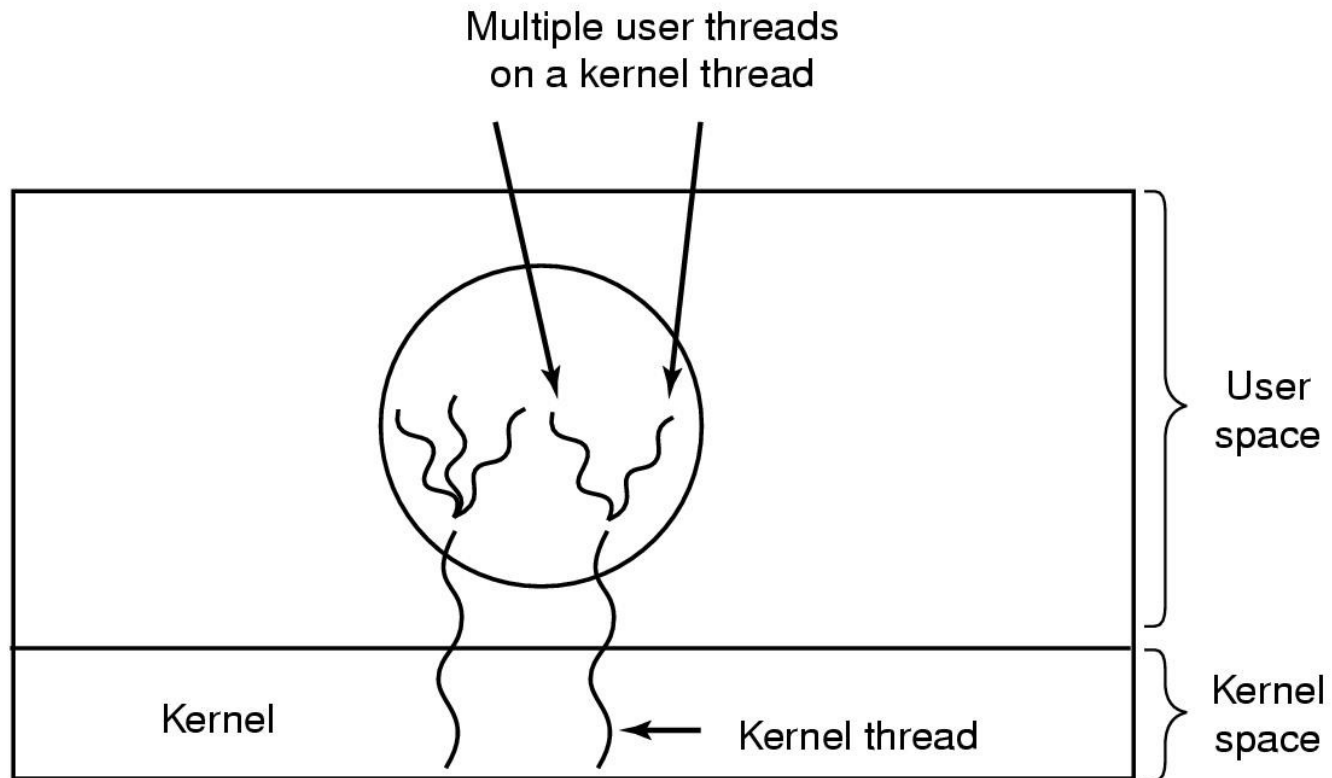
Luồng

Thực hiện luồng trong Kernel (2)

- Đa luồng được cung cấp trực tiếp bởi Hệ điều hành:
 - Kernel quản lý các tiến trình và luồng
 - CPU lập lịch cho luồng thực hiện trong kernel
- Ích lợi của đa luồng trong kernel
 - Tốt cho các kiến trúc đa bộ xử lý (multiprocessor architecture)
 - Nếu một luồng bị chặn thì đó không phải là nguyên nhân dẫn đến luồng khác bị chặn.
- Bất lợi của đa luồng trong kernel
 - Khởi tạo và quản lý luồng sẽ chậm hơn

Luồng

Thực hiện Lai (Hybrid)



Ghép các luồng mức user vào các luồng mức kernel

2.3 Lập lịch-Scheduling

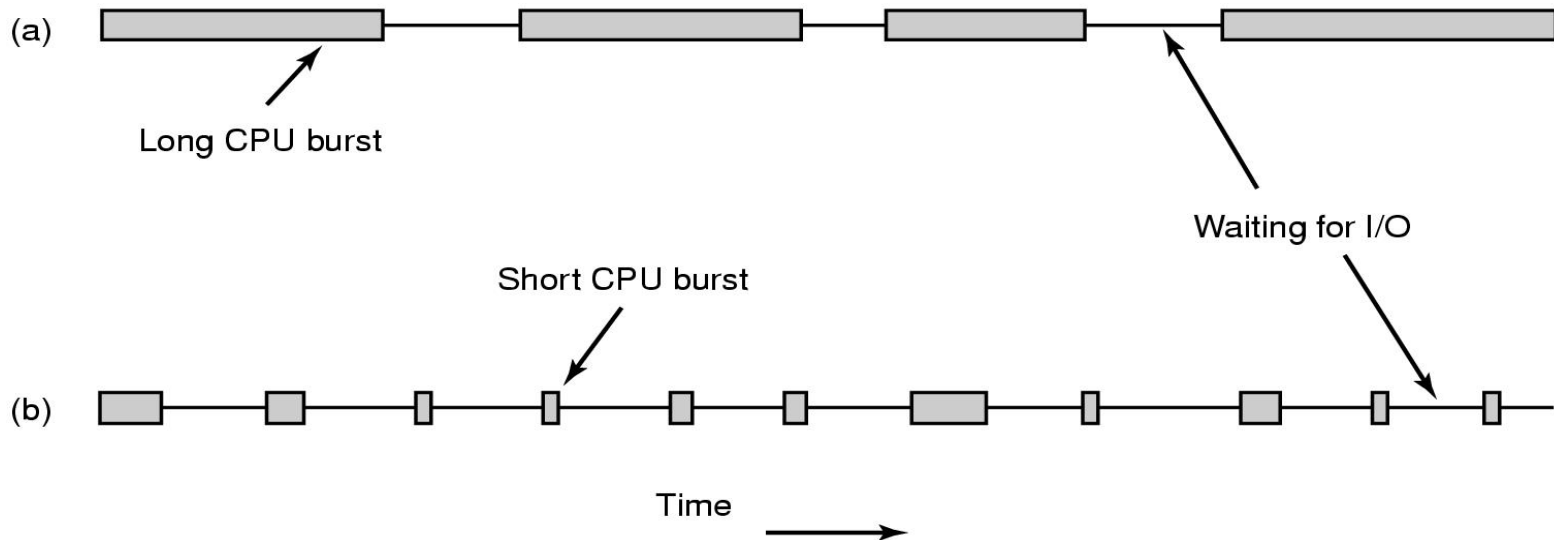
Lập lịch

Giới thiệu về lập lịch (1)

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait
- CPU burst distribution

Lập lịch

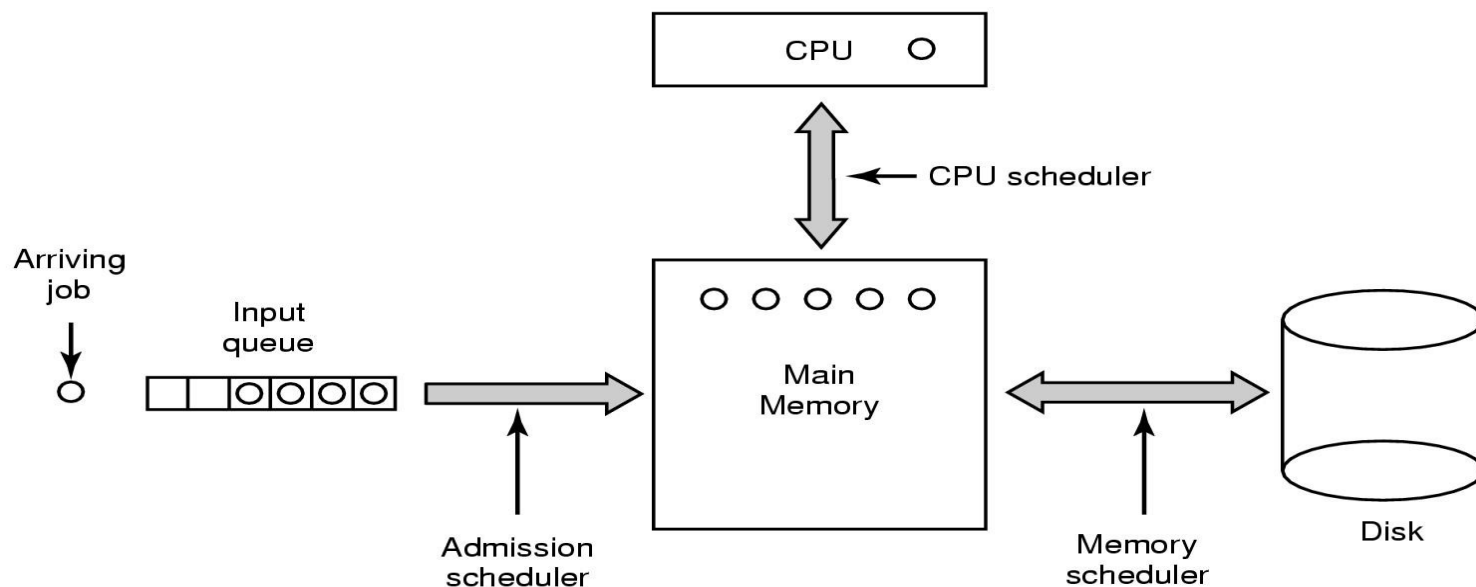
Giới thiệu về lập lịch (2)



- Bursts of CPU usage alternate with periods of I/O wait
 - (a) a CPU-bound process
 - (b) an I/O-bound process

Lập lịch

Giới thiệu về lập lịch (3)



Ba mức lập lịch

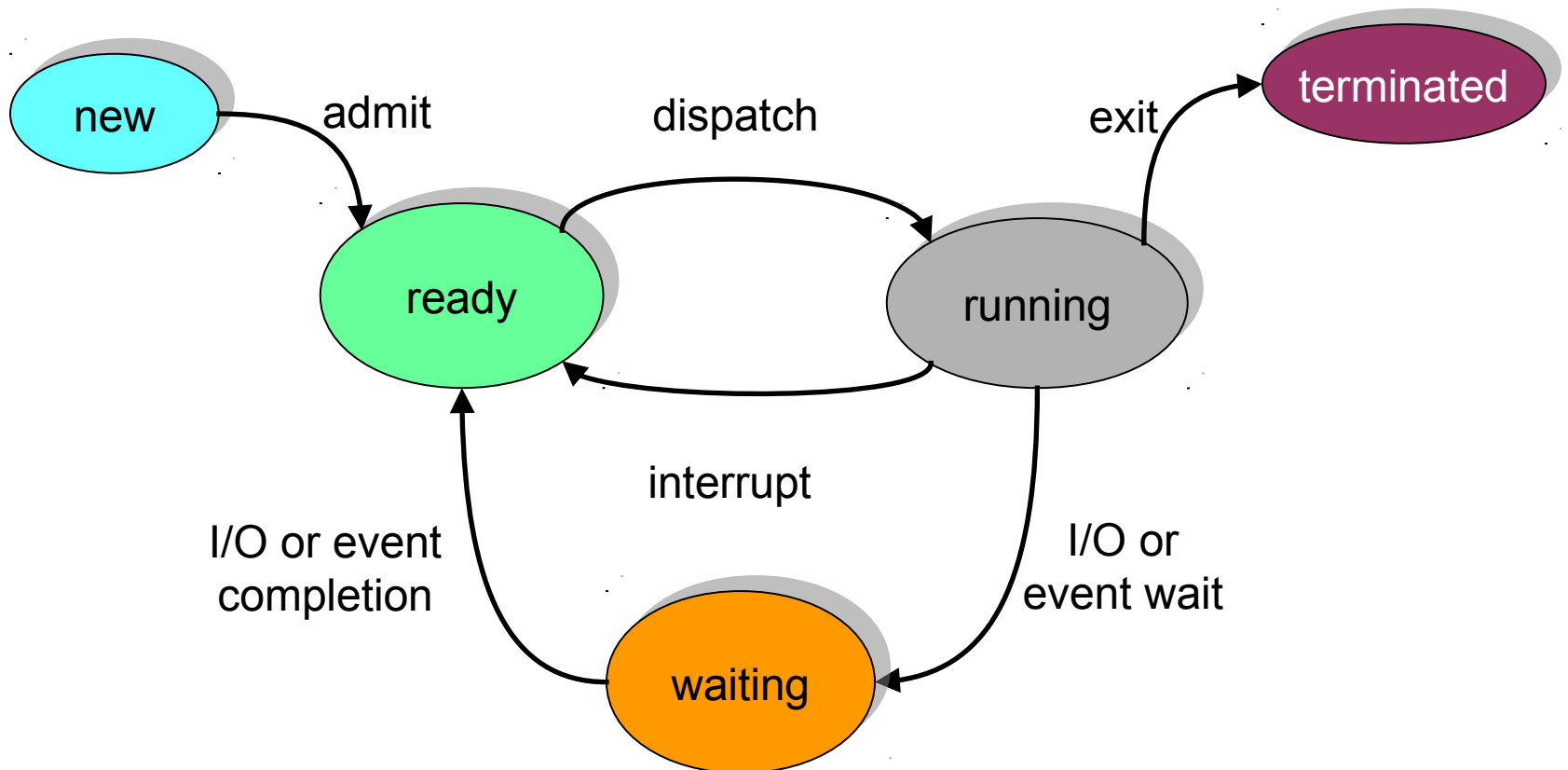
Lập lịch

Giới thiệu về lập lịch (4)

- Lựa chọn trong số các tiến trình trong bộ nhớ mà sẵn sàng thực hiện, và cấp CPU cho chúng
- Bộ lập lịch CPU quyết định chọn Tiến trình cho chạy khi một tiến trình:
 1. Chuyển từ trạng thái đang chạy về trạng thái bị chặn lại
 2. Chuyển từ trạng thái đang chạy về trạng thái sẵn sàng
 3. Chuyển từ trạng thái bị chặn hay một tiến trình mới tạo lập sẵn trạng thái sẵn sàng
 4. Kết thúc
- *Thuật toán lập lịch độc quyền (Nonpreemptive scheduling algorithm)* bốc chọn các tiến trình và đưa đến CPU cho chạy cho đến khi bị chặn lại hay cho đến khi tự nguyện giải phóng CPU
- *Thuật toán lập lịch không độc quyền (preemptive scheduling algorithm)* bốc chọn các tiến trình và đưa đến CPU cho chạy với khoản thời gian xác định

Lập lịch

Giới thiệu về lập lịch (5)



Lập lịch

Giới thiệu về lập lịch (6)

Tiêu chí lập lịch

- CPU utilization – giữ cho CPU luôn bận khi có thể
- Throughput – số tiến trình được xử lý trên một đơn vị thời gian
- Turnaround time – thời gian xoay vòng
- Waiting time – thời gian tiến trình chờ trong hàng đợi sẵn sàng (ready queue)
- Response time – thời gian hồi đáp

Lập lịch

Giới thiệu về lập lịch (7)

Tiêu chí tối ưu

- CPU utilization: Max
- Throughput: Max
- Turnaround time: Min
- Waiting time: Min
- Response time: Min

Lập lịch

Giới thiệu về lập lịch (8)

Mục đích của thuật toán lập lịch

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Lập lịch

Lập lịch trong Batch Systems (1)

Lập lịch First-Come, First-Served (FCFS)

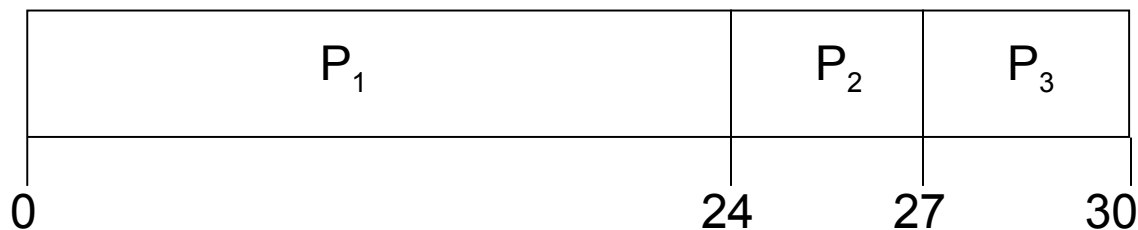
Tiến trình Thời gian đốt cháy CPU (Burst Time)

P_1 24

P_2 3

P_3 3

- Giả sử các tiến trình đến hàng đợi theo thứ tự: P_1, P_2, P_3
Thứ tự lập lịch là:



- Thời gian chờ $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Thời gian chờ trung bình: $(0 + 24 + 27)/3 = 17$

Lập lịch

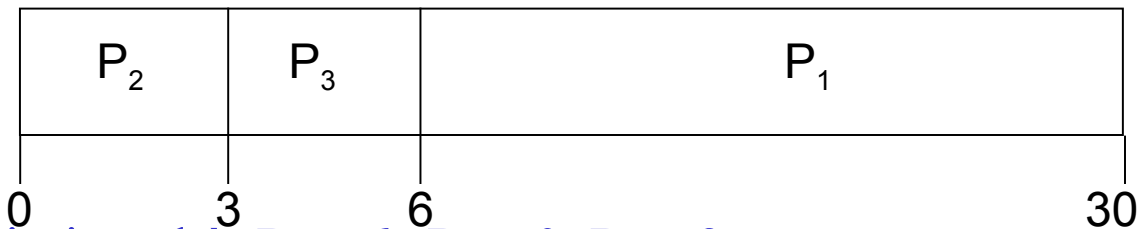
Lập lịch trong Batch Systems (2)

Lập lịch FCFS (tiếp)

Giả sử các Tiến trình đến theo thứ tự

P_2, P_3, P_1

- Thứ tự lập lịch sẽ là:



- Thời gian chờ $P_1 = 6; P_2 = 0; P_3 = 3$
- Thời gian chờ trung bình: $(6 + 0 + 3)/3 = 3$
- Tốt hơn nhiều so với trường hợp trước
- Phụ thuộc vào thời gian xử lý của những tiến trình đầu danh sách*

Lập lịch

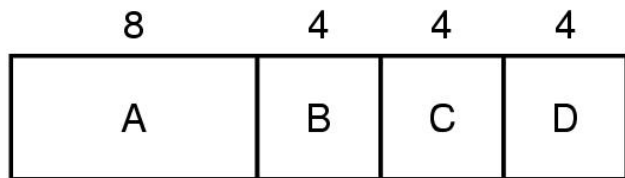
Lập lịch trong Batch Systems (3)

Lập lịch Shortest-Job-First (SJF)

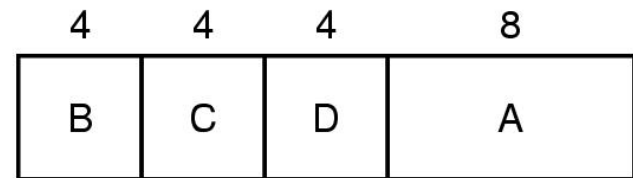
- Kết hợp mỗi tiến trình với độ dài thời gian đốt cháy CPU của tiến trình tiếp theo. Sử dụng độ dài này để lập lịch với thời gian ngắn nhất
- Có hai lược đồ:
 - Độc quyền (nonpreemptive) – mỗi khi CPU đã cấp cho tiến trình thì nó không thể cấp cho tiến trình khác cho đến khi hết thời gian đốt cháy CPU của tiến trình
 - Không độc quyền (preemptive) – nếu một tiến trình mới đến với thời gian đốt cháy CPU dài hơn thời gian xử lý còn lại của tiến trình thì cho tiến trình tiếp tục thực hiện. Ngược lại thì cấp cho tiến trình mới đến. Lược đồ này còn được hiểu là ***Shortest-Remaining-Time-First (SRTF)***
- SJF là tối ưu – cho thời gian chờ trung bình là nhỏ nhất cho tất cả tập các tiến trình.

Lập lịch

Lập lịch trong Batch Systems (4)



(a)

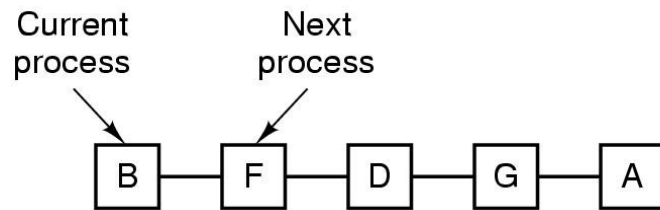


(b)

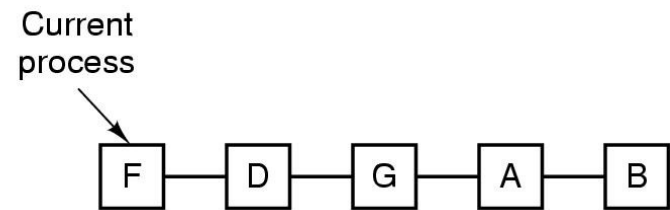
Ví dụ về lập lịch công việc ngắn nhất trước

Lập lịch

Lập lịch trong các hệ thống tương tác Interactive Systems (1)



(a)



(b)

- **Lập lịch Round Robin**

- (a) danh sách các tiến trình có thể chạy thực hiện
- (b) danh sách các tiến trình có thể chạy thực hiện sau B, sử dụng quantum

Lập lịch

Lập lịch trong các hệ thống tương tác

Interactive Systems (2)

Round Robin (RR)

- Mỗi tiến trình được cấp một khoản thời gian sử dụng CPU rất nhỏ gọi là thời gian quantum (*time quantum*), thông thường trong khoản từ 10-100 miligiây. Hết thời gian này mà tiến trình chưa kết thúc thì lại được đưa vào cuối hàng đợi sẵn sàng (ready queue).
- Nếu có n tiến trình trong hàng đợi và thời gian quantum là q , thì mỗi tiến trình có $1/n$ lần CPU ít nhất q đơn vị thời gian cùng một lúc. Không có tiến trình nào phải chờ hơn $(n-1)q$ đơn vị thời gian
- Hiệu suất:
 - q lớn \Rightarrow FIFO
 - q nhỏ \Rightarrow mất thời gian chuyển đổi ngữ cảnh, hiệu suất thấp

Lập lịch

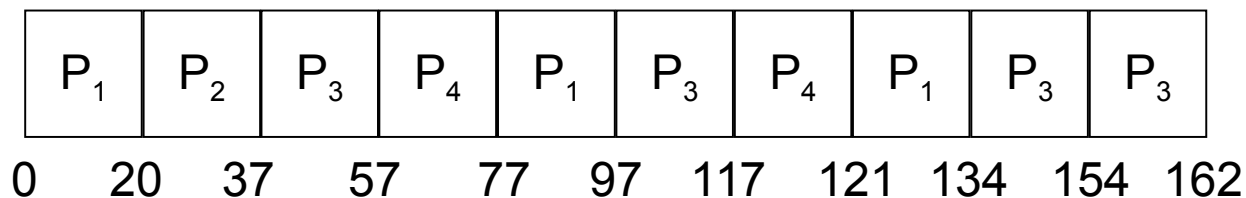
Lập lịch trong các hệ thống tương tác

Interactive Systems (3)

Ví dụ RR với thời gian Quantum = 20

<u>Process</u>	<u>Burst Time</u>
P_1	53
P_2	17
P_3	68
P_4	24

- Thứ tự lập lịch:



Trong mọi trường hợp, *thời gian xoay vòng* trung bình cao hơn SJF, nhưng *thời gian hồi đáp* thì tốt hơn.

Lập lịch

Lập lịch trong các hệ thống tương tác

Interactive Systems (4)

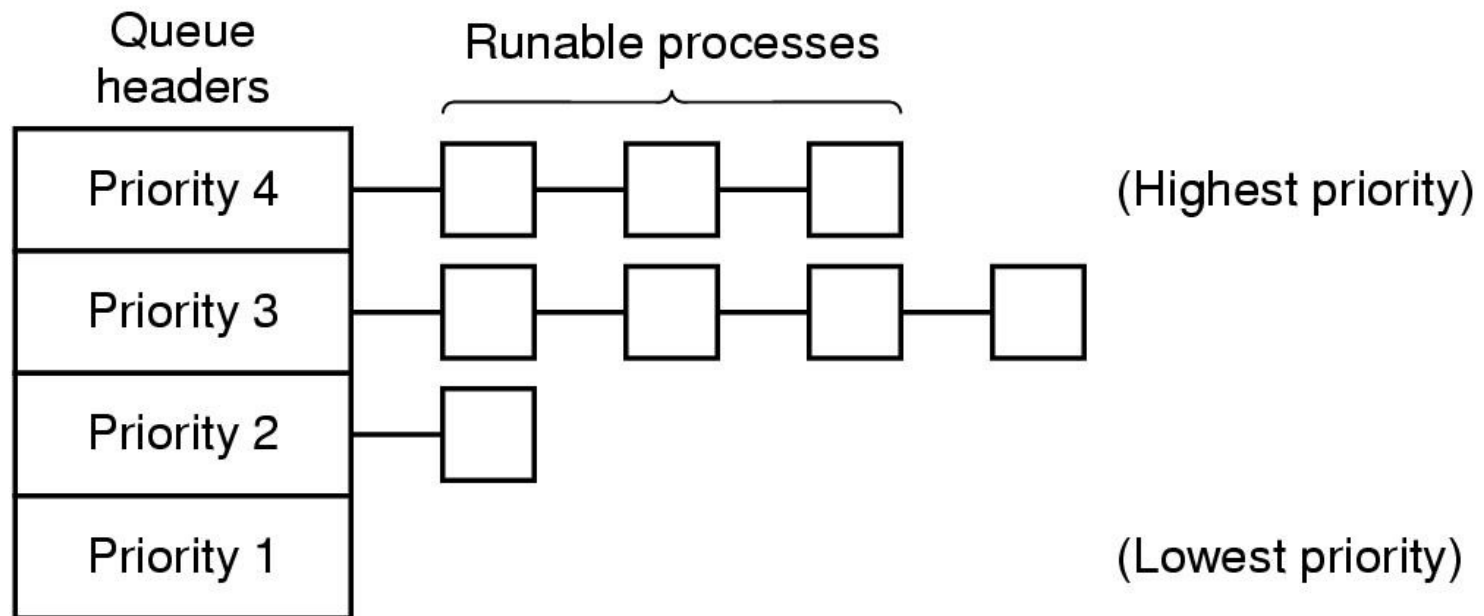
Lập lịch theo mức ưu tiên: Một số ưu tiên (số nguyên) được gán cho mỗi tiến trình

- CPU sẽ được cấp cho tiến trình có ưu tiên cao nhất
- Không độc quyền
- Độc quyền
- SJF là trường hợp đặc biệt của ưu tiên
- Vấn đề \equiv Sự đói khát (Starvation) CPU – những tiến trình ưu tiên thấp có thể không bao giờ được thực hiện
- Giải pháp \equiv Lão hóa (Aging) – ngay thời điểm gia tăng mức ưu tiên của tiến trình

Lập lịch

Lập lịch trong các hệ thống tương tác Interactive Systems (5)

Thuật toán lập lịch với 4 lớp ưu tiên



Lập lịch

Lập lịch trong các hệ thống thời gian thực Real-Time Systems (1)

- *Hard real-time* systems – được yêu cầu để hoàn thành các tác vụ quan trọng trong khoảng thời gian được đảm bảo
- *Soft real-time* computing – yêu cầu các tiến trình quan trọng nhận được ưu tiên hơn các tiến trình khác

Lập lịch

Lập lịch trong các hệ thống thời gian thực

Real-Time Systems (2)

Lập lịch real-time system

- Được cho
 - m sự kiện định kỳ
 - sự kiện i xảy ra trong chu kỳ P_i và yêu cầu C_i giây
- Sau đó bộ tải chỉ có thể được điều khiển

nếu

$$\sum_{i=1}^m \frac{C_i}{P_i} \leq 1$$

Lập lịch

Chính sách so với Cơ chế

- Separate what is allowed to be done with how it is done
 - a process knows which of its children threads are important and need priority
- Scheduling algorithm parameterized
 - mechanism in the kernel
- Parameters filled in by user processes
 - policy set by user process

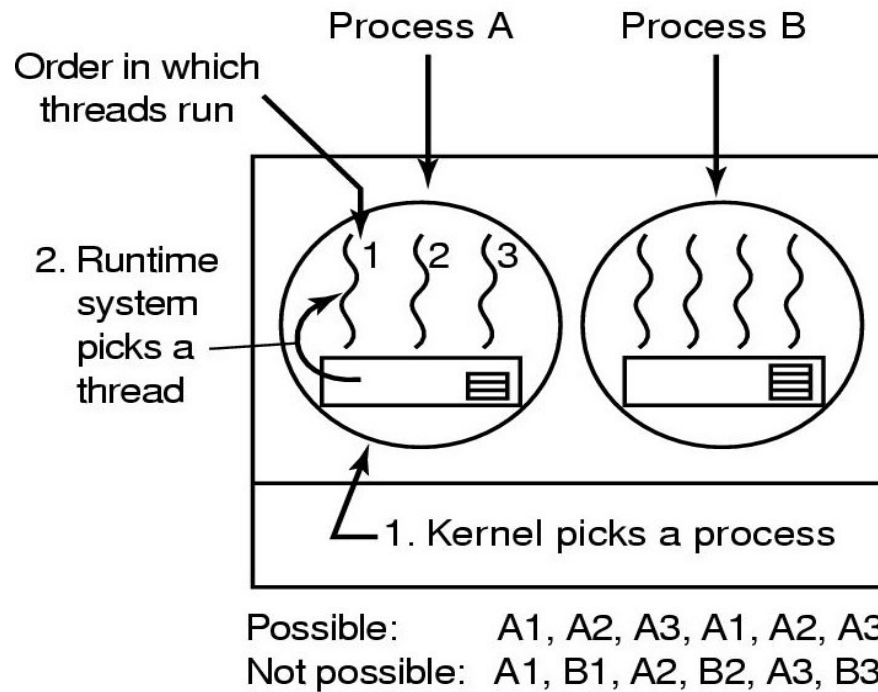
Lập lịch

Lập lịch Luồng (1)

- Lập lịch cục bộ (Local Scheduling) –
Làm thế nào để thư viện các luồng quyết định luồng nào được lập lịch
- Lập lịch toàn cục (Global Scheduling) –
Làm thế nào để kernel quyết định luồng kernel nào chạy tiếp theo

Lập lịch

Lập lịch Luồng (2)

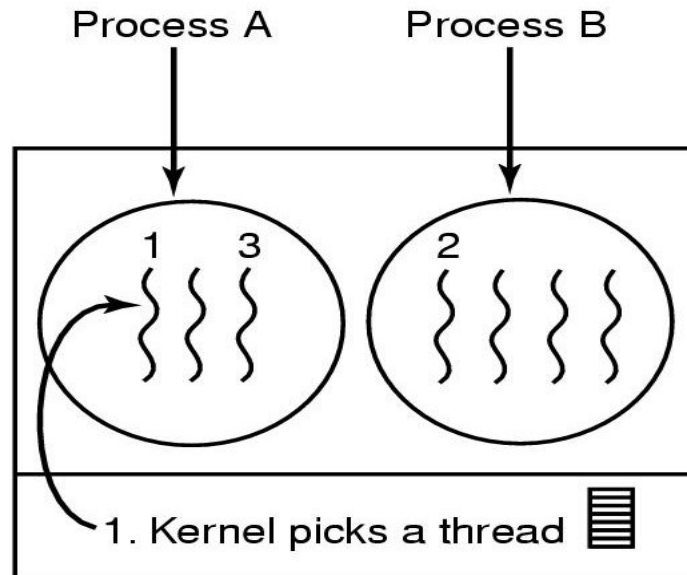


Có thể lập lịch các luồng ở mức User

- quantum mỗi tiến trình: 50-mili giây
- mỗi luồng: 5 mili giây /đốt cháyCPU

Lập lịch

Lập lịch Luồng (3)



Possible: A1, A2, A3, A1, A2, A3

Also possible: A1, B1, A2, B2, A3, B3

Có thể lập lịch các luồng ở mức kernel

- quantum mỗi tiến trình: 50-mili giây
- mỗi luồng: 5 mili giây /đốt cháyCPU

2.4 Giao tiếp liên tiến trình

Interprocess Communication

Các tiến trình tương tác

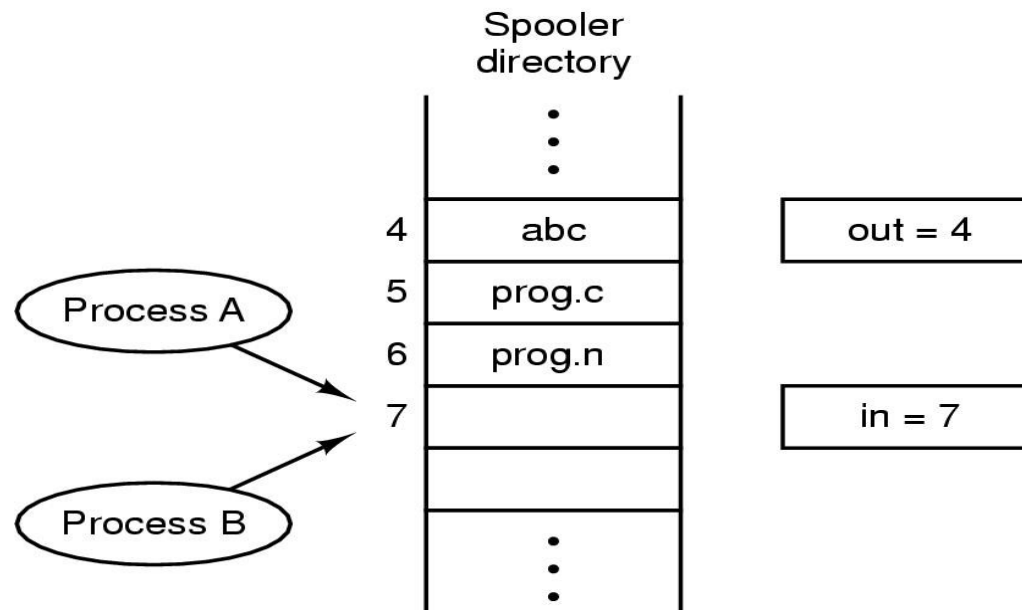
- Tiến trình **độc lập (Independent process)** không bị ảnh hưởng hay ảnh hưởng đến việc thực thi của tiến trình khác
- Tiến trình **tương tác (Cooperating process)** có thể bị ảnh hưởng hay ảnh hưởng đến việc thực thi của tiến trình khác
- Ưu điểm của tiến trình tương tác
 - Chia sẻ thông tin
 - Tăng tốc độ tính toán
 - Modun hóa
 - Tiện lợi

Vấn đề của dữ liệu chia sẻ

- Việc truy cập đồng thời vào dữ liệu chia sẻ có thể dẫn đến sự không thống nhất về dữ liệu
- Duy trì tính nhất quán dữ liệu đòi hỏi các cơ chế để đảm bảo thực hiện có trật tự các tiến trình tương tác
- Cần có cơ chế để giao tiếp và đồng bộ các hoạt động giữa các tiến trình

Các điều kiện tranh đua

- Hai tiến trình muốn truy cập bộ nhớ chia sẻ cùng một lúc và kết quả cuối cùng phụ thuộc vào tiến trình nào chạy chính xác, được gọi là điều kiện tranh đua
- Loại trừ lẫn nhau (Mutual exclusion)** là cách để cấm nhiều tiến trình truy cập vào dữ liệu chia sẻ cùng một lúc



Đoạn găng-Critical Regions (1)

Một phần của chương trình mà ở đó bộ nhớ dùng chung được truy cập được gọi là:

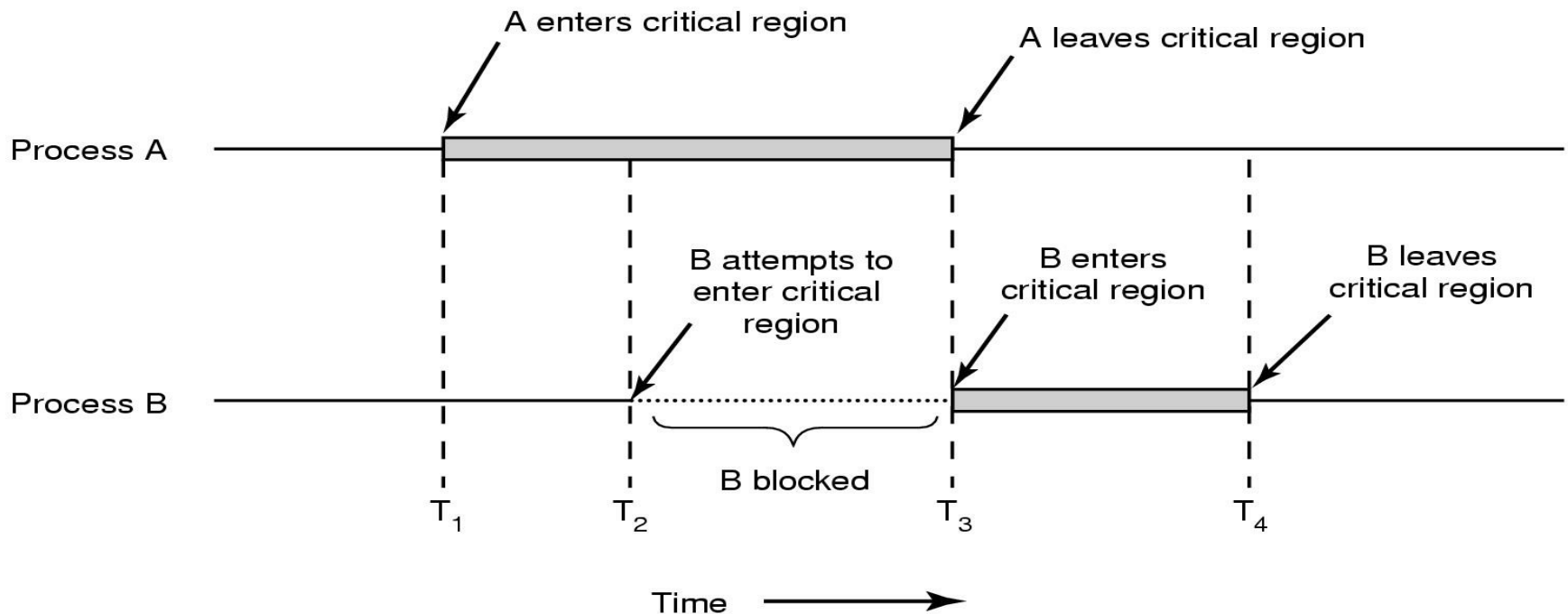
Đoạn găng hay Miền găng (Critical Regions-Critical Section)

Bốn điều kiện để cung cấp sự loại trừ lẫn nhau:

1. Không có hai tiến trình cùng đồng thời ở trong đoạn găng
2. Không có ràng buộc về tốc độ hay số CPU có trong hệ thống
3. Không một tiến trình nào đang chạy bên ngoài đoạn găng có thể chặn tiến khác vào đoạn găng
4. Không có tiến trình nào phải chờ vô hạn để vào đoạn găng

Critical Regions (2)

Ví dụ: Loại trừ nhau sử dụng đoạn găng



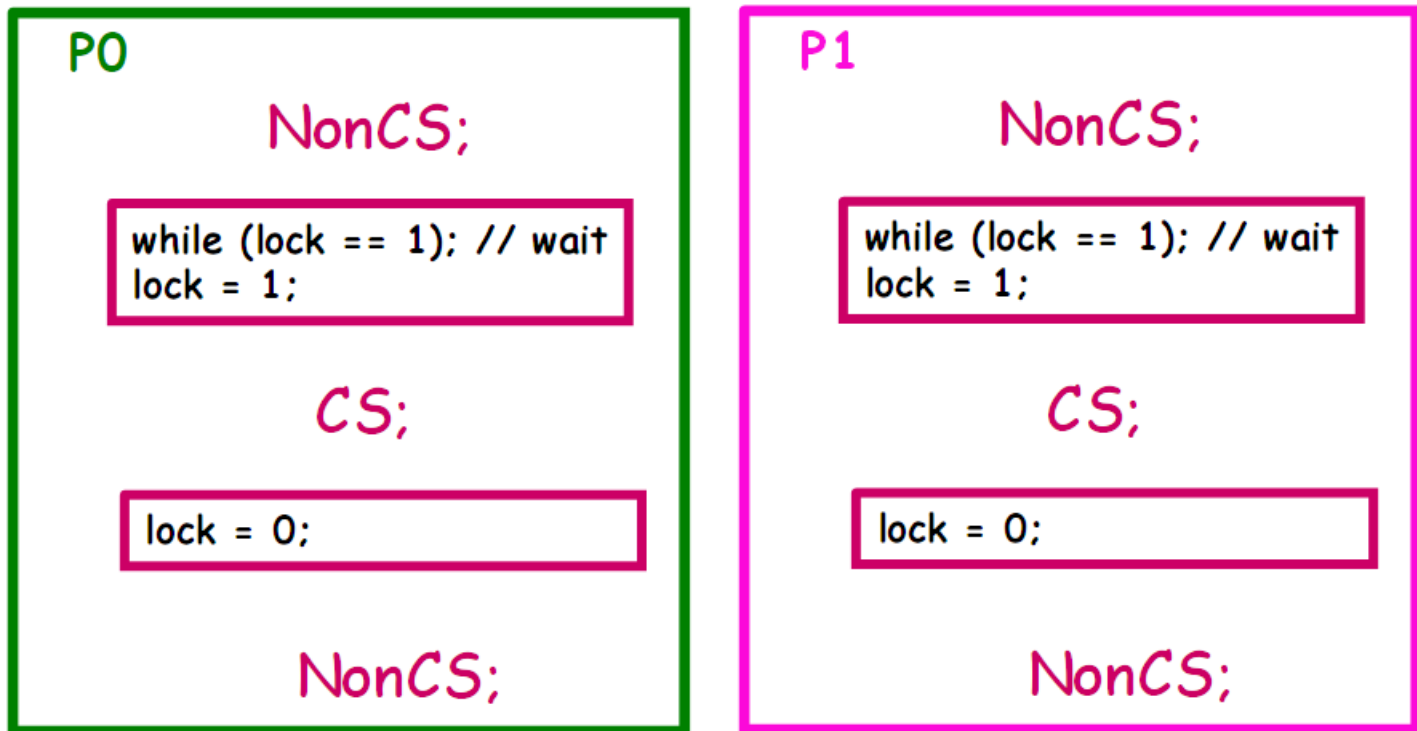
Giải pháp: Loại trừ nhau với chờ bận-Busy waiting

- Các giải pháp mềm
 - Lock Variables
 - Strict Alternation
 - Peterson's Solution
- Các giải pháp cứng
 - Disabling Interrupts
 - The TSL Instruction

Loại trừ nhau với chờ bận

Giải pháp mềm 1: Khóa-Lock

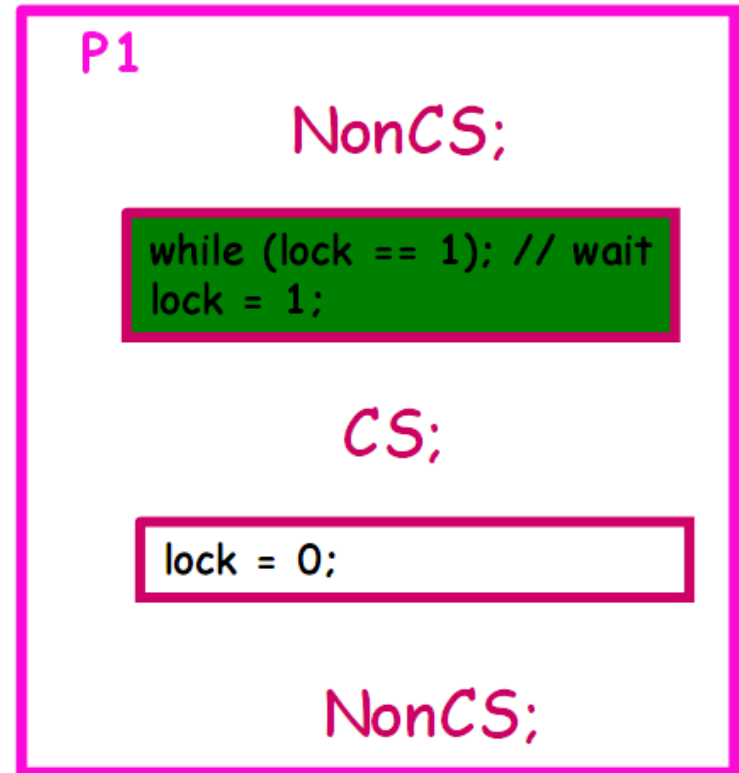
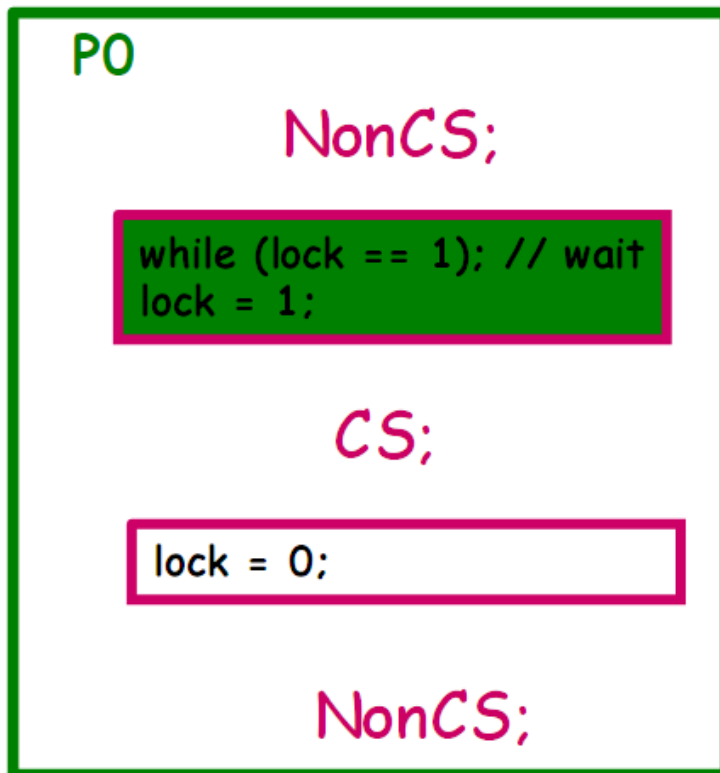
int lock = 0



Loại trừ nhau với chờ bận

Giải pháp mềm 1: Event

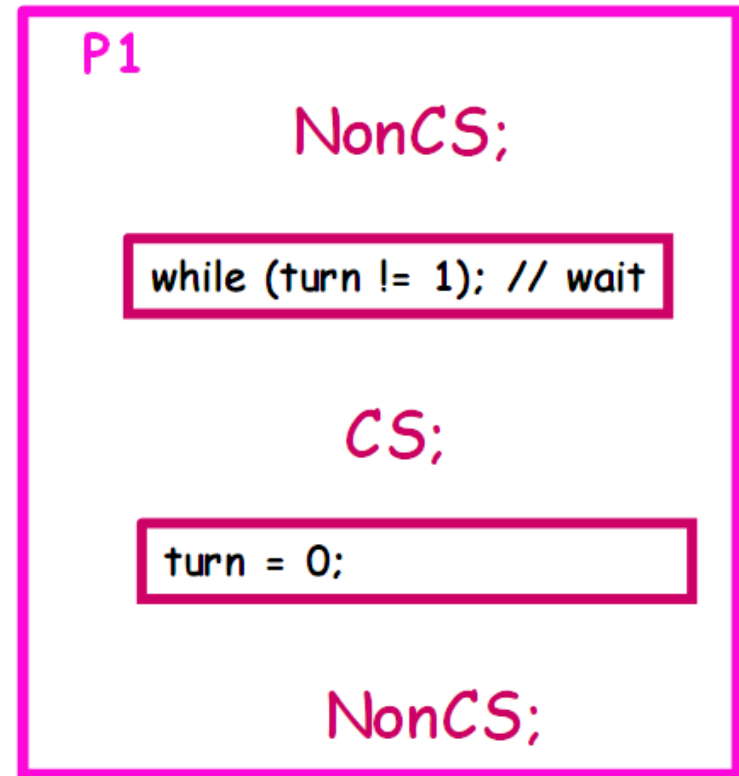
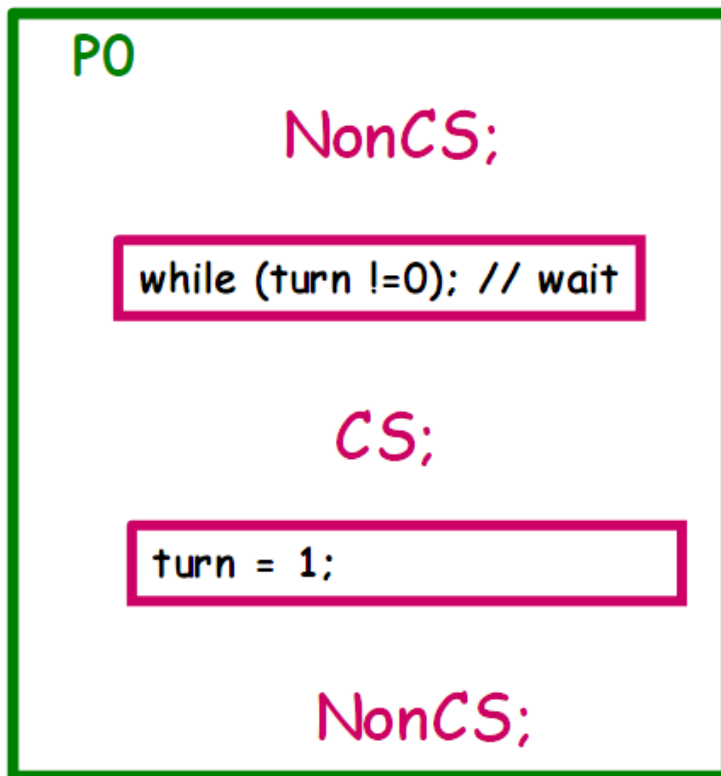
int lock = 0



Loại trừ nhau với chờ bận

Giải pháp mềm 2: Luân phiên

int **turn** = 1



Loại trừ nhau với chờ bận

Giải pháp mềm 2: Luân phiên

- Dành cho 2 tiến trình
- Có khả năng loại trừ
 - Sử dụng biến “*turn*”, một tiến trình đến phiên “*turn*” vào CS.

Loại trừ nhau với chờ bận

Giải pháp mềm 3: Giải pháp của Peterson

- `int turn;`
- `Boolean interest[2] = FALSE;`

Pi

NonCS;

```
j = 1 - i;  
interest[i] = TRUE;  
turn = j;  
while (turn==j && interest[j]==TRUE);
```

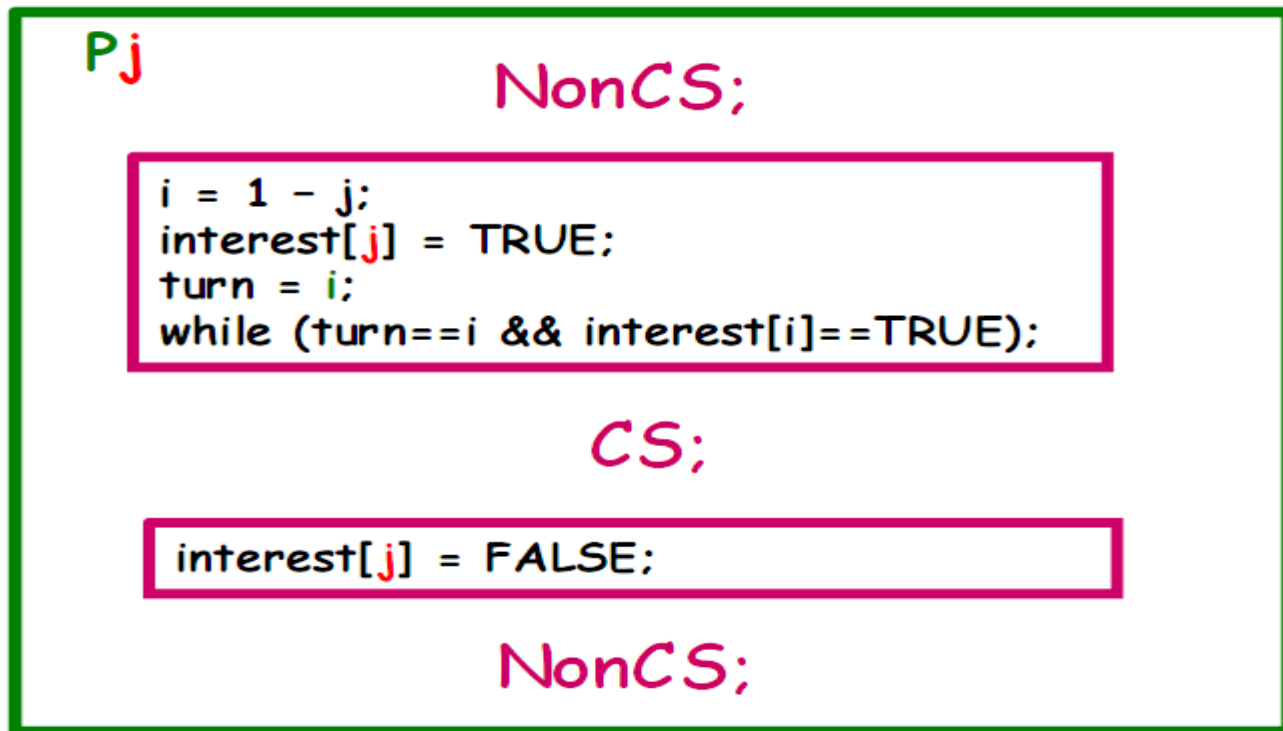
CS;

```
interest[i] = FALSE;
```

NonCS;

Loại trừ nhau với chờ bận

Giải pháp mềm 3: Giải pháp của Peterson



Loại trừ nhau với chờ bận

Nhận xét giải pháp mềm 3: Giải pháp của Peterson

- Đáp ứng 3 điều kiện:
 - Loại trừ nhau
 - P_i có thể vào CS khi $interest[j] == F$, hoặc $turn == i$
 - Nếu cả hai cùng muốn quay lại $turn$ có thể chỉ nhận giá trị 0 hoặc 1, vì thế chỉ một tiến trình vào được CS
 - Tiến độ
 - Sử dụng 2 biến khác biệt $interest[i] \implies$ opposing cannot lock
 - Bounded Wait: cả $interest[i]$ và $turn$ thay đổi giá trị
- Không mở rộng N tiến trình

Loại trừ nhau với chờ bận

Nhận xét cho các giải pháp chờ bận

- Lãng phí thời gian của CPU
- Khó mở rộng
- Giải pháp 1 tốt hơn khi các tiến trình cùng mức ưu tiên

Loại trừ nhau với chờ bận

- Giải pháp mềm
 - Lock Variables
 - Strict Alternation
 - Peterson's Solution
- Giải pháp cứng
 - Disabling Interrupts
 - The TSL Instruction

Loại trừ nhau với chờ bận

Giải pháp cứng 1: Vô hiệu hóa Ngắt (1)

NonCS;

Disable Interrupt;

CS;

Enable Interrupt;

NonCS;

- Disable Interrupt: cấm tất cả các Ngắt
- Enable Interrupt: khôi phục ngắt trở lại

Loại trừ nhau với chờ bận

Giải pháp cứng 1: Vô hiệu hóa Ngắt (2)

- Không cần thận
 - Nếu tiến trình bị khóa trong CS?
 - Hệ thống đứng
 - Cho phép tiến trình sử dụng lệnh ưu tiên
 - Nguy hiểm!
- Hệ thống với N CPU?
 - Không chắc chắn loại trừ nhau

Loại trừ nhau với chờ bận

Giải pháp cứng 2: TSL Instruction

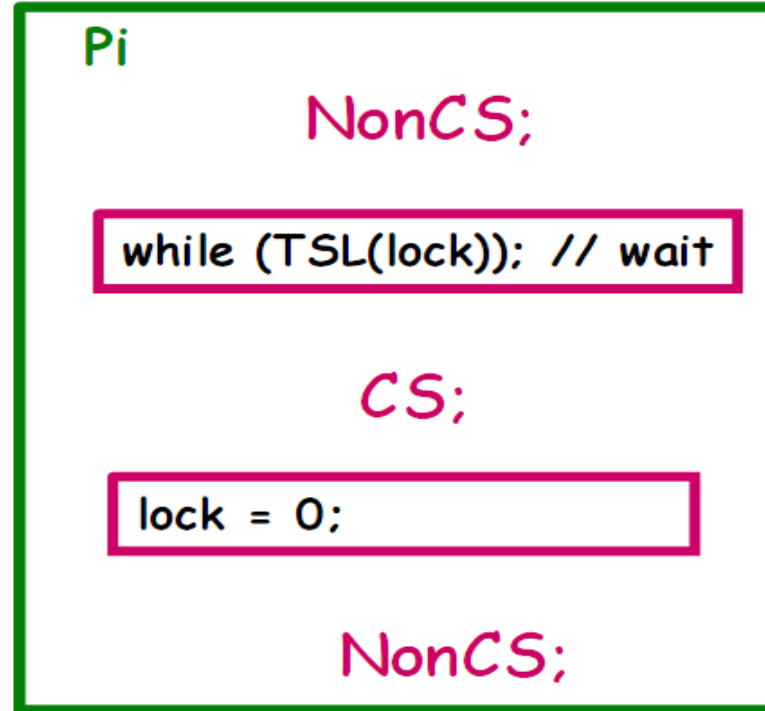
- CPU cung cấp kiểm tra và xác lập Lock nguyên thủy
 - Trả về giá trị hiện tại của biến, thiết lập biến giá trị True
 - Không thể chia ra để biểu diễn

```
TSL (boolean &target)
{
    TSL = target;
    target = TRUE;
}
```

Loại trừ nhau với chờ bận

Giải pháp cứng 2: Áp dụng TSL

```
int lock = 0
```



Loại trừ nhau với chờ bận

Nhận xét cho các giải pháp cứng

- Phù hợp hỗ trợ cho cơ chế phần cứng
 - Tuy nhiên, không dễ với hệ thống nhiều CPU system
- Dễ dàng mở rộng cho N tiến trình

Loại trừ nhau với chờ bận

Nhận xét

- Sử dụng CPU không hiệu quả
 - Thường xuyên lặp kiểm tra điều kiện để vào CS
- Khắc phục
 - Các tiến trình nên bị chặn lại nếu không đủ điều kiện để vào đoạn găng, để nhường CPU cho tiến trình khác
 - Sử dụng Scheduler
 - Chờ và xem xét ...

Giải pháp đồng bộ với Sleep & Wakeup

- Semaphore
- Monitor
- Message passing

Giải pháp "Sleep & Wake up"

if not, Sleep();

CS;

Wakeup(somebody);

- Tù bỏ CPU khi không vào CS
- Sau khi ra khỏi CS, sẽ đánh thức các tiến trình khác để vào CS
- Cần thiết cung cấp cho HĐH
 - Bởi vì làm thay đổi trạng thái của tiến trình

Giải pháp "Sleep & Wake up": Ý kiến

- HĐH cung cấp 2 lời gọi hệ thống:
 - **Sleep()**: Làm cho các tiến trình nhận được trạng thái bị chặn
 - **WakeUp(P)**: Tiến trình P nhận được trạng thái sẵn sàng
- Ứng dụng
 - Sau khi kiểm tra điều kiện, vào CS hoặc gọi Sleep() phụ thuộc vào kết quả kiểm tra.
 - Tiến trình sử dụng CS trước, sẽ đánh thức các Tiến trình bị chặn.

Áp dụng Sleep() và Wakeup()

- int busy;
- int blocked;

```
if (busy) {  
    blocked = blocked + 1;  
    Sleep();  
}  
else busy = 1;
```

CS;

```
busy = 0;  
if(blocked) {  
    WakeUp(P);  
    blocked = blocked - 1;  
}
```

Vấn đề với Sleep và WakeUp

- Lý do:
 - Kiểm tra điều kiện và từ bỏ CPU có thể bị chặn
 - Biến Lock không được bảo vệ

Giải pháp đồng bộ với Sleep và Wakeup Semaphore

- Được đề xuất bởi Dijkstra, 1965
- Thuộc tính: **Semaphore s**;
 - Giá trị nguyên
 - Xây dựng 2 lời gọi hệ thống:
 - **Down(s)**
 - **Up(s)**
 - Down và Up được thực hiện tuần tự

Giải pháp đồng bộ với Sleep và Wakeup

Thiết lập Semaphore (Sleep & Wakeup)



Semaphore: similar to resource

Processes "request" semaphore: call Down(s)

If Down(s) is not finished: resource is not allocated

Blocked, insert to s.L

Need OS's support

Sleep() & Wakeup()

Giải pháp đồng bộ với Sleep và Wakeup

Thiết lập Semaphore (Sleep & Wakeup)

Down (S)

```
{  
    S.value --;  
    if S.value < 0  
    {  
        Add(P, S.L);  
        Sleep();  
    }  
}
```

Up(S)

```
{  
    S.value ++;  
    if S.value ≤ 0  
    {  
        Remove(P, S.L);  
        Wakeup(P);  
    }  
}
```

Giải pháp đồng bộ với Sleep và Wakeup

Sử dụng Semaphore

Semaphore $s = 1$

P_i

Down (s)
CS;
Up(s)

Semaphore $s = 0$

P_1 :

Job1;

Up(s)

P_2 :

Down (s);

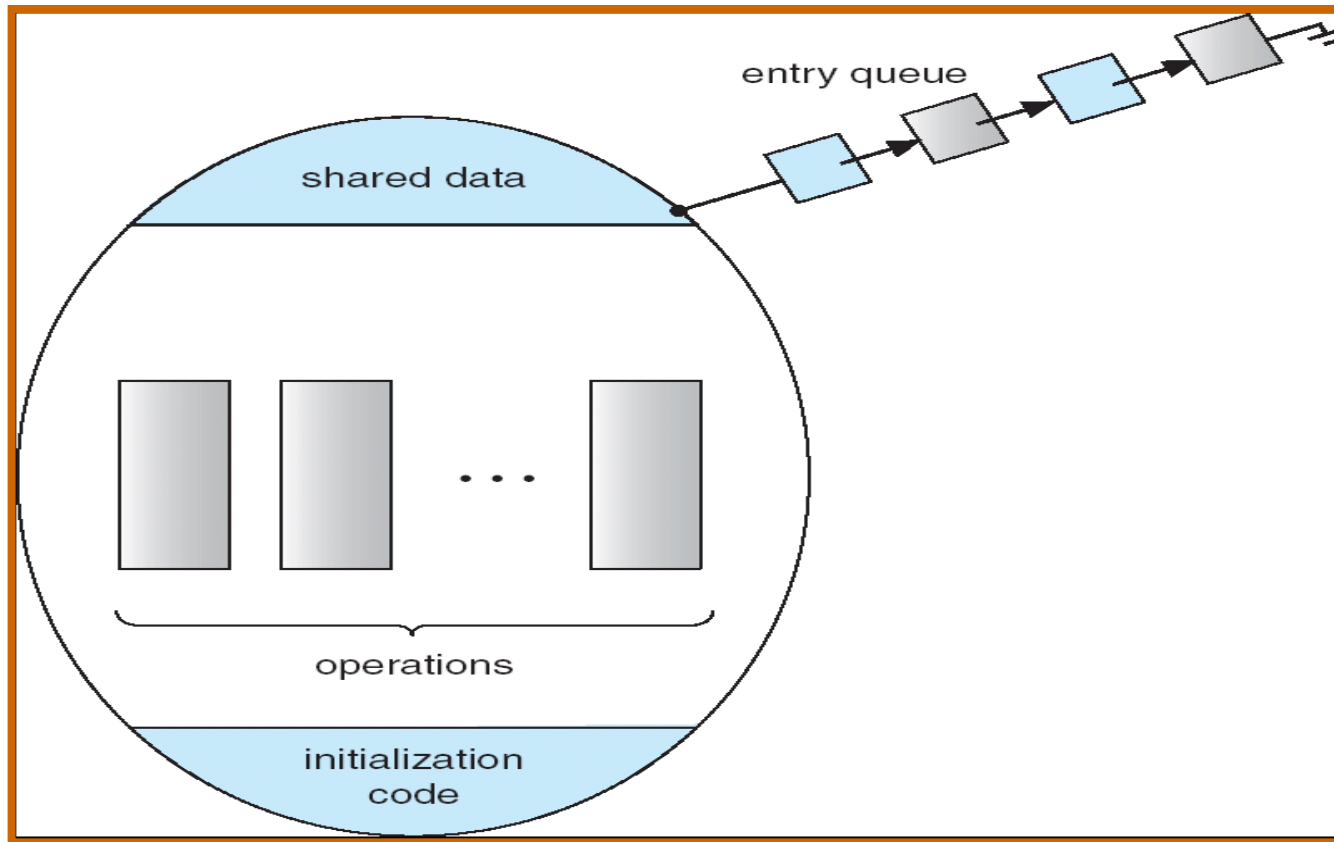
Job2;

Giải pháp đồng bộ với Sleep và Wakeup Monitor

- Hoare (1974) và Brinch (1975)
- Cơ chế đồng bộ được cung cấp bởi ngôn ngữ lập trình
 - Cung cấp các hàm cũng giống như Semaphore
 - Dễ sử dụng và dễ xóa hơn Semaphore
 - Chắc chắn loại trừ được thực hiện một cách tự động
 - Sử dụng biến điều kiện để thực hiện đồng bộ

Giải pháp đồng bộ với Sleep và Wakeup

Monitor: structure



Giải pháp đồng bộ với Sleep và Wakeup

Monitor: structure

```
monitor monitor-name
{
    shared variable declarations
    procedure body P1 (...) {
        . . .
    }
    procedure body P2 (...) {
        . . .
    }
    procedure body Pn (...) {
        . . .
    }
    {
        initialization code
    }
}
```

Giải pháp đồng bộ với Sleep và Wakeup

Sử dụng Monitor


```
Monitor    M  
<resource type> RC;  
Function   AccessMutual  
           CS; // access RC
```

```
Pi  
M.AccessMutual(); //CS
```

```
Monitor    M  
Condition  c;  
Function   F1  
           Job1;  
           Signal(c);  
Function   F2  
           Wait(c);  
           Job2;
```

```
P1 :  
M.F1();
```

```
P2 :  
M.F2();
```



Giải pháp đồng bộ với Sleep và Wakeup

Message Passing

- Tiến trình phải biết rõ tên tiến trình khác:
 - **send** ($P, message$) – gửi message đến tiến trình P
 - **receive**($Q, message$) – nhận message từ tiến trình Q
- Các đặc tính
 - Các liên kết được thiết lập một cách tự động
 - Một liên kết được thiết lập chính xác đến một cặp tiến trình giao tiếp
 - Giữa mỗi cặp có một liên kết chính xác
 - Liên kết có thể đơn hướng, nhưng thông thường phải là hai hướng

Phân loại các vấn đề đồng bộ

- Bounded-Buffer Problem
(Producer-Consumer Problem)
- Readers and Writers Problem
- Dining-Philosophers Problem