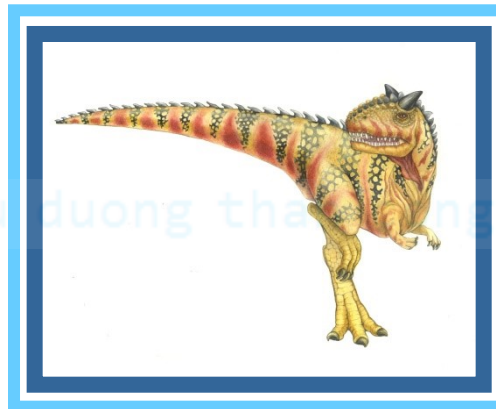


Chương 5: Đồng bộ - 1





Ôn tập chương 4

- Tại sao phải định thời? Nêu các bộ định thời và mô tả về chúng?
- Các tiêu chuẩn định thời CPU?
- Có bao nhiêu giải thuật định thời? Kể tên?
- Mô tả và nêu ưu điểm, nhược điểm của từng giải thuật định thời? FCFS, SJF, SRTF, RR, Priority Scheduling, HRRN, MQ, MFQ.





Bài tập chương 4

- Sử dụng các giải thuật FCFS, SJF, SRTF, Priority - Pre, RR (10) để tính các giá trị thời gian đợi, thời gian đáp ứng và thời gian hoàn thành trung bình và vẽ giản đồ Gantt

Thread	Priority	Burst	Arrival
P_1	20	20	0
P_2	30	25	25
P_3	15	25	20
P_4	35	15	35
P_5	5	35	10
P_6	10	50	15





Mục tiêu

- Hiểu được vấn đề tranh chấp giữa các tiến trình trong hệ điều hành
- Biết được các giải pháp để giải quyết tranh chấp
- Hiểu được các vấn đề trong giải quyết tranh chấp
- Biết được các yêu cầu của các giải pháp trong việc giải quyết tranh chấp và phân nhóm các giải pháp





Nội dung

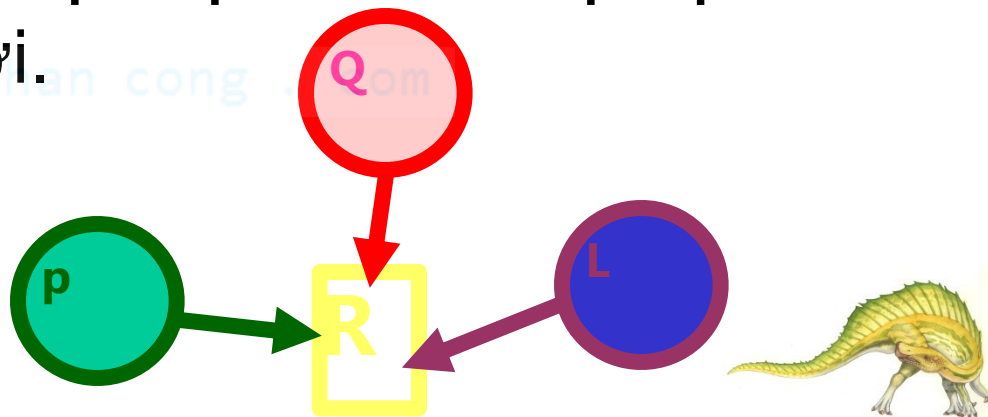
- Giới thiệu về race condition
- Giới thiệu các giải pháp tổng quát để giải quyết tranh chấp
- Phân tích các chi tiết các vấn đề trong việc giải quyết tranh chấp
- Yêu cầu của giải pháp trong việc giải quyết tranh chấp
- Phân nhóm các giải pháp





Đặt vấn đề

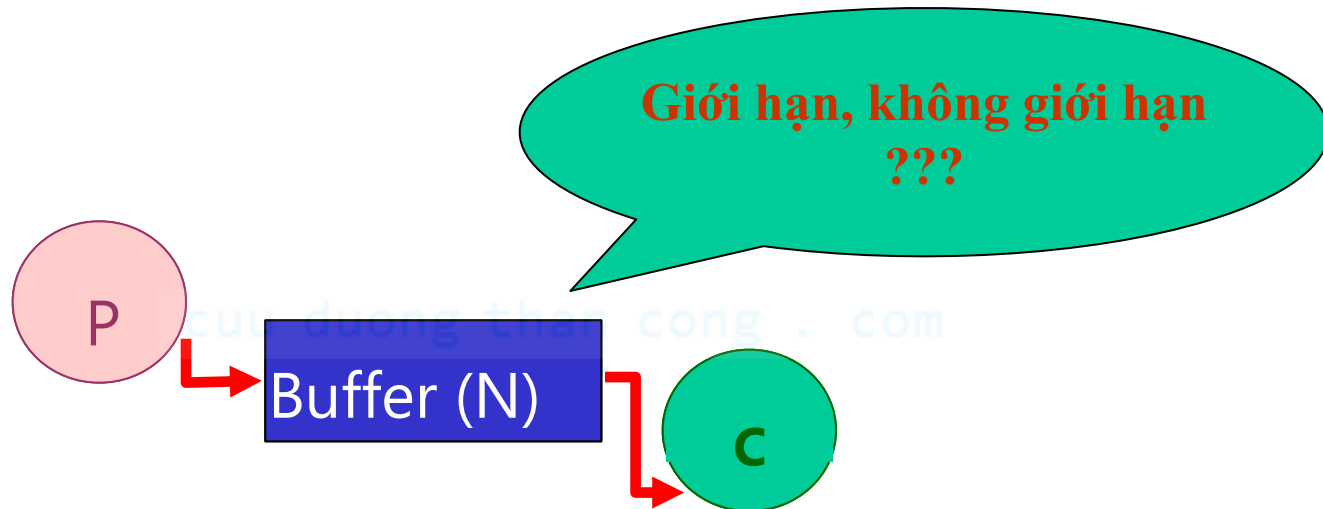
- Khảo sát các process/thread **thực thi đồng thời** và **chia sẻ dữ liệu** (qua shared memory, file).
- Nếu không có sự kiểm soát khi truy cập các dữ liệu chia sẻ thì có thể đưa đến ra trường hợp **không nhất quán dữ liệu** (data inconsistency).
- Để duy trì sự nhất quán dữ liệu, hệ thống cần có cơ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





Bài toán Producer - Consumer

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





Bounded buffer

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Xét trường hợp Bounded buffer:

■ Quá trình Producer

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

biến count được chia sẻ
giữa producer và consumer

■ Quá trình Consumer

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





Bounded buffer (tt)

■ Các lệnh tăng, giảm biến count tương đương trong ngôn ngữ máy là:

- ✓ Producer (count++)
 - register1 = count
 - register1 = register1 + 1
 - count = register1
- ✓ Consumer (count--)
 - register2 = count
 - register2 = register2 - 1
 - count = register2

(Trong đó, các register là các thanh ghi của CPU)





Bounded buffer (tt)

- Mã máy của các lệnh tăng và giảm biến count có thể bị thực thi xen kẽ
- Giả sử count đang bằng 5. Chuỗi thực thi có thể xảy ra, khi **quantum time = 2** chu kỳ lệnh

T0: producer	register1 := count	{register1 = 5}
T1: producer	register1 := register1 + 1	{register1 = 6}
T2: consumer	register2 := count	{register2 = 5}
T3: consumer	register2 := register2 - 1	{register2 = 4}
T4: producer	count := register1	{count = 6}
T5: consumer	count := register2	{count = 4}





Bounded buffer (tt)

- Mã máy của các lệnh tăng và giảm biến count có thể bị thực thi xen kẽ
- Giả sử count đang bằng 5. Chuỗi thực thi có thể xảy ra khi **quantum time = 3** chu kỳ lệnh

T0: producer	register1 := count	{register1 = 5}
T1: producer	register1 := register1 + 1	{register1 = 6}
T2: producer	count := register1	{count = 6}
T3: consumer	register2 := count	{register2 = 6}
T4: consumer	register2 := register2 - 1	{register2 = 5}
T5: consumer	count := register2	{count = 5}

⇒ Cần phải có giải pháp để các lệnh $count++$, $count--$ phải là đơn nguyên (atomic), nghĩa là thực hiện như một lệnh đơn, không bị ngắt nửa chừng.





Bounded buffer (tt)

- **Race condition:** nhiều process truy xuất và thao tác đồng thời lên dữ liệu chia sẻ (như biến count) và kết quả cuối cùng của việc truy xuất đồng thời này phụ thuộc thứ tự thực thi của các lệnh thao tác dữ liệu.
- Để dữ liệu chia sẻ được nhất quán, cần bảo đảm sao cho tại mỗi thời điểm chỉ có một process được thao tác lên dữ liệu chia sẻ. Do đó, cần có cơ chế đồng bộ hoạt động của các process này.





Vấn đề Critical Section

- Giả sử có n process truy xuất đồng thời dữ liệu chia sẻ
- Cấu trúc của mỗi process P_i có đoạn code như sau:

Do {

entry section

/ vào critical section */*

critical section

/ truy xuất dữ liệu chia sẻ */*

exit section

/ rời critical section */*

remainder section

/ làm những việc còn lại */*

} While (1);

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





Vấn đề Critical Section (tt)

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

cuu duong than cong . com





Yêu cầu của lời giải cho CS Problem

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





Yêu cầu của lời giải cho CS Problem

Tính chất (2) Progress

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

Được sửa lại như sau:

Progress. If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely (**Sách tham khảo trang 206**)

- ➔ nếu không có process nào đang thực thi trong vùng tranh chấp và đang có một số process chờ đợi vào vùng tranh chấp thì:
 - Chỉ những process không đang thực thi trong Remainder Section mới có thể tham gia trong việc quyết định process nào sẽ vào vùng tranh chấp.
 - Và việc chọn lựa này không được trì hoãn vô hạn





Phân loại giải pháp

■ Nhóm giải pháp **Busy Waiting**

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

■ Nhóm giải pháp **Sleep & Wakeup**

- Semaphore
- Monitor
- Message





Các giải pháp “Busy waiting”

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

cuu duong than cong . com

While (chưa có quyền) do_nothing() ;

CS;

cuu duong than cong . com

Từ bỏ quyền sử dụng CS





Các giải pháp “Sleep & Wake up”

- Tắt CPU khi chưa được vào miền găng
- Cần Hệ điều hành hỗ trợ

```
if (chưa có quyền) Sleep();
```

```
CS;
```

```
WakeUp (somebody);
```





Tổng kết

- Race condition
- Các giải pháp tổng quát để giải quyết tranh chấp
- Các chi tiết các vấn đề trong việc giải quyết tranh chấp
- Yêu cầu của giải pháp trong việc giải quyết tranh chấp
- Các nhóm các giải pháp





UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Câu hỏi ôn tập

cuu duong than cong . com

cuu duong than cong . com

