BÀI TẬP TRÊN LỚP

MÔN HỌC: HỆ PHÂN TÁN

CHƯƠNG 6: ĐỒNG BỘ HÓA

HỌ TÊN SV: Mạc Quang Huy MSSV: 20173169

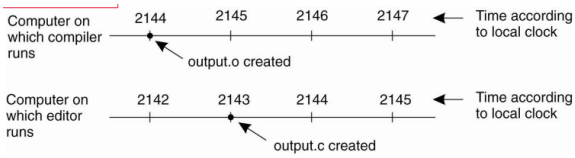
MÃ LỚP: 118636 MÃ HỌC PHẦN: IT4611

Câu hỏi 1: Trình bày 1 ví dụ để mô phỏng vấn đề gặp phải khi các máy tính/tiến trình hoạt động trong hệ thống phân tán mà không có đồng hộ vật lý dùng chung.

Trả lời:

Trường hợp editor, compiler:

- Khi cả 2 trên cùng một máy => compiler chỉ hiệu chinht các file được chỉnh sửa sau khi complider
- Khi nằm trong bối cảnh hệ phân tán => do lệch thời gian, việc sửa ở file edit, compiler sẽ không biết và không dịch, gây ra sự sai lệch



Câu hỏi 2: Tại sao Lamport lại đề xuất sử dụng đồng hồ logic thay cho đồng hồ vật lý trong hệ phân tán?

Trả lời:

- Trong nhiều trường hợp, giữa các tiến trình không cần khớp nhau về thời gian thực tế mà chỉ cần khớp nhau về thứ tự, cái nào đến trước, cái nào đến sau. Thêm vào đó, đồng bộ đồng hồ logic sẽ đơn giản hơn so với đồng hồ vật lý.
- Ứng dụng: Đảm bảo thứ tự toàn cục của gửi thông điệp theo nhóm (ở tầng Middleware)

Câu hỏi 3: Đặc điểm gì của mạng không dây (wireless network) khiến cho thiết kế các giải thuật đồng bộ khác các kiểu mạng khác?

Trả lời:

- Vấn đề của giải thuật đồng bộ trong mạng không dây là làm sao để có thể tiết kiệm năng lượng nhất có thể, và chống nhiễu trên đường truyền tín hiệu. Ngoài ra, hình trạng mạng trong mạng không dây cũng thay đổi liên tục, cũng ảnh hưởng đến việc thiết kế giải thuật đồng bộ dành riêng cho mạng không dây.

Câu hỏi 4: Giải thuật Lamport được đưa ra để thực hiện loại trừ lẫn nhau (mutual exclusion). Giải thuật được mô tả như sau:

Hệ thống có n tiến trình: P1, P2, ... Pn. Có 1 tài nguyên chia sẻ dùng chung gọi là SR (Shared Resource). Mỗi tiến trình sẽ lưu trữ một hàng đợi queuei để lưu các yêu cầu của các tiến trình khác khi chưa được thực hiện.

Khi tiến trình Pi muốn truy cập vào SR, nó sẽ quảng bá 1 thông điệp REQUEST(tsi,i) cho tất cả các tiến trình khác, đồng thời lưu trữ thông điệp đó vào hàng đợi của mình (queuei) trong đó tsi là timestamp của yêu cầu.

Khi 1 tiến trình Pj nhận được yêu cầu REQUEST(tsi,i) từ tiến trình Pi thì nó đưa yêu cầu đó vào hàng đợi của mình (queuej) và gửi trả lại cho Pi thông điệp REPLY.

Tiến trình Pi sẽ tự cho phép mình sử dụng SR khi nó kiểm tra thấy yêu cầu của nó nằm ở đầu hàng đợi queuei và các yêu cầu khác đều có timestamp lớn hơn yêu cầu của chính nó.

Tiến trình Pi, khi không dùng SR nữa sẽ xóa yêu cầu của nó khỏi hàng đợi và quảng bá thông điệp RELEASE cho tất cả các tiến trình khác.

Khi tiến trình Pj nhận được thông điệp RELEASE từ Pi thì nó sẽ xóa yêu cầu của Pi trong hàng đợi của nó.

Câu hỏi:

a) Để thực hiện thành công 1 tiến trình vào sử dụng SR, hệ thống cần tổng cộng bao nhiêu thông điệp?

b) Có 1 cách cải thiện thuật toán trên như sau: sau khi Pj gửi yêu cầu REQUEST cho các tiến trình khác thì nhận được thông điệp REQUEST từ Pi, nếu nó nhận thấy rằng timestamp của REQUEST nó vừa gửi lớn hơn timestamp của REQUEST của Pi, nó sẽ không gửi thông điệp REPLY cho Pi nữa.

Cải thiện trên có đúng hay không? Và với cải thiện này thì tổng số thông điệp cần để thực hiện thành công 1 tiến trình vào sử dụng SR là bao nhiêu? Giải thích.

Trả lời:

a.

- 1 thông điệp dùng để quảng bá yêu cầu REQUEST
- Ít nhất N – 1 thông điệp REPLY

-> Ta cần ít nhất là N thông điệp

- Ngoài ra, số lượng thông điệp RELEASE phụ thuộc vào việc có bao nhiêu tiến trình có thông điệp có timestamp nhỏ hơn của Pi: 0<= số thông điệp RELEASE <= n – 1

⇨ Tổng số thông điệp cần dùng nằm trong khoảng n đến 2n – 1.

b.

Cải thiện trên, chỉ những tiến trình nào phải sử dụng tài nguyên sau tiến trình Pi, hoặc không cần sử dụng tài nguyên mới gửi lại REPLY cho Pi.

⇨ Pi nếu như nhận được đủ n – 1 thông điệp REPLY là Pi có thể dùng SR. Vì thế, số thông điệp cần gửi trong mạng sẽ ít đi, làm tăng hiệu năng mạng, tổng số thông điệp cần để thực hiện thành công 1 tiến trình sử dụng SR luôn là N thông điệp.

Câu hỏi 5: Giải thuật Szymanski được thiết kế để thực hiện loại trừ lẫn nhau. Ý tưởng của giải thuật đó là xây dựng một *phòng chờ* (waiting room) và có *đường ra* và *đường vào*, tương ứng với *cổng*

ra và *cổng vào*. Ban đầu *cổng vào* sẽ được mở, *cổng ra* sẽ đóng. Nếu có một nhóm các tiến trình cùng yêu cầu muốn được sử dụng tài nguyên chung SR (shared resource) thì các tiến trình đó sẽ được xếp hàng ở *cổng vào* và lần lượt vào *phòng chờ*. Khi tất cả đã vào *phòng chờ* rồi thì tiến trình cuối cùng vào phòng sẽ đóng *cổng vào* và mở *cổng ra*. Sau đó các tiến trình sẽ lần lượt được sử dụng tài nguyên chung. Tiến trình cuối cùng sử dụng tài nguyên sẽ đóng *cổng ra* và mở lại *cổng vào*.

Mỗi tiến trình Pi sẽ có 1 biến $flag_i$, chỉ tiến trình P_i mới có quyền ghi, còn các tiến trình P_j ($j \neq i$) thì chỉ đọc được. Trạng thái mở hay đóng cổng sẽ được xác định bằng việc đọc giá trị flag của các tiến trình khác. Mã giả của thuật toán đối với tiến trình *i* được viết như sau:

```
#Thực hiện vào ph ng đợi
flag[i] ← 1 await(all
flag[1..N]∈ {0,1,2}) flag[i]
← 3 ifany flag[1..N]=1:
    flag[i] ← 2
await(any flag[1..N]=4)
 flag[i]   ←   4   await(all
flag[1..i-1]∈ {0,1})


#Sử dụng tài nguyên
#...


#Thực hiện giải phóng tài nguyên
await(all flag[i+1..N]∈ {0,1,4})
 flag[i] ←
0
```

Giải thích ký pháp trong thuật toán:

await(điều_kiện): chờ đến khi thỏa mãn điều_kiện

all: tất cả

any: có bất kỳ 1 cái nào **Câu**

hỏi:

flag[i] sẽ có 5 giá trị trạng thái từ 0-4. Dựa vào giải thuật trên, 5 giá trị đó mang ý nghĩa tương ứng nào sau đây (có giải thích): - Chờ tiến trình khác vào *phòng chờ*

- *Cổng vào* được đóng

- Tiến trình *i* đang ở ngoài *phòng chờ*

- Rời phòng, mở lại *cổng vào* nếu không còn ai trong *phòng chờ*

- Đứng đợi trong *phòng chờ*

Trả lời:

- Ban đầu, các tiến trình đều đang xếp hàng bên ngoài phòng chờ:

flag[i] ← 1

- Sau đó, chúng đợi cửa vào mở và bên trong không có tiến trình nào của lượt chờ trước đang đợi trong phòng chờ (đợi đến khi không có tiến trình nào có flag 3 và 4), sau khi tất cả đã vào phòng chờ rồi thì chúng đợi trong phòng (flag 3).

await(all flag[1..N]∈{0,1,2})

flag[i] ← 3

Nếu như có tiến trình nào đang bên ngoài (if any flag[1..N]=1:) thì chuyển sang trạng thái chờ chúng vào phòng (flag 2), và đợi đến khi có một tiến trình nào đó đóng cửa (flag 4) thì tất cả bọn chúng cũng chuyển sang flag 4, báo hiệu cửa vào đã đóng.

- Đợi cho các tiến trình có index nhỏ hơn mình dùng xong tài nguyên (mang flag 1 và 0), tiến trình i lúc này sẽ dùng tài nguyên.
- Đợi cho các tiến trình có index lớn hơn nó dùng xong tài nguyên, nó sẽ chuyển flag sang 0: Rời phòng, mở lại cổng vào nếu không còn ai trong phòng chờ.