BÀI TẬP TRÊN LỚP

MÔN HỌC: HỆ PHÂN TÁN

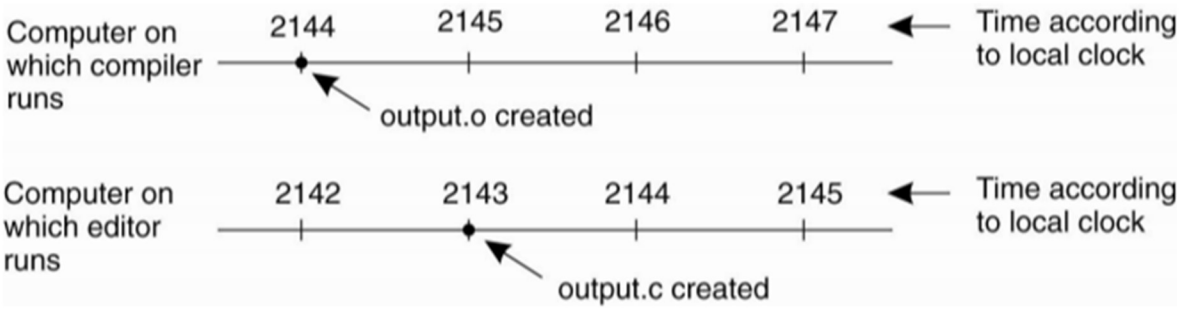
CHƯƠNG 4: ĐỒNG BỘ HÓA

HỌ TÊN SV: TRẦN TRUNG PHONG MSSV: 20210676

MÃ LỚP: 149501 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 hồ vật lý dùng chung.

- Trong chương trình *make* của Unix, các chương trình được chia nhỏ thành nhiều file nguồn để khi có 1 file bị thay đổi thì không cần biên dịch lại toàn bộ chương trình. Khi lập trình viên chạy *make* thì thời điểm đó được đánh dấu cho tất cả các file đã bị modify. Ví dụ file output.c có thời gian là 1000, file output.o có thời gian là 999 tức là nó đã bị thay đổi và cần biên dịch lại, còn nếu file output.o có thời gian là 1001 thì không cần. Ví dụ trường hợp sau đây khi không có đồng hồ vật lý dùng chung:



- Trên máy *compiler* file output.o được đánh dấu thời gian là 2144, ngay sau đó file output.c được thay đổi trên máy *editor* nhưng vì không có đồng hồ vật lý dùng chung nên thời gian trên máy *editor* chậm hơn 1 chút và file output.c được đánh dấu thời gian là 2143, vì thế nên *make* sẽ không biên dịch lại, điều này sẽ gây ra lỗi mà lập trình viên không thể kiểm soát được.

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

Lamport lại đề xuất sử dụng đồng hồ logic thay cho đồng hồ vật lý trong hệ phân tán vì có những tiến trình không cần phải chính xác theo thời gian thực tế mà chỉ cần quan tâm đến thứ tự trước sau của các tiến trình. Thời gian bắt đầu của một tiến trình luôn xảy ra sau khi đã kết thúc tiến tình xảy ra trước.

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?
Đặc điểm của mạng không dây khiến cho thiết kế các giải thuật đồng bộ khác các kiểu mạng khác: Trong mạng không dây, nhiều thiết bị không được cắm điện trực tiếp mà thường chạy bằng pin, vì thế cần thiết kế giải thuật đồng bộ để giảm bớt thao tác gửi nhận và xử lý so với các kiểu mạng khác.

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.

a. Để Pi sử dụng SR, nó quảng bá thông điệp REQUEST đến các tiến trình khác và các tiến trình khác lại gửi lại Pi thông điệp REPLY, như vậy có tất cả 2(n-1) thông

điệp. Mặt khác, giả sử có k tiến trình trước Pi trong hàng đợi, như vậy Pi cần thêm k thông điệp RELEASE khác để xóa các tiến trình trước nó, mà các thông điệp này đều theo hình thức quảng bá nên tổng cộng cần (k+2)(n-1) với k là số tiến trình trước Pi trong hàng đợi.

b. Cải thiện trên là đúng. Vẫn xét tiến trình Pi cần sử dụng SR, vì không nhận REPLY từ các tiến trình trước nên chỉ cần (k+2)(n-1) – k thông điệp, với k là số tiến trình trước Pi trong hàng đợi.

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 flagi, chỉ tiến trình Pi mới có quyền ghi, còn các tiến trình Pj (j ≠ 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 phòng 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*

- Chờ tiến trình khác vào *phòng chờ:* 2 – flag có giá trị 2 khi có bất kỳ tiến trình nào đang ở ngoài phòng nên đây là trạng thái chờ tiến trình khác vào phòng.
- *Cổng vào* được đóng: 4 – giá trị sau khi thoát khỏi trạng thái chờ tiến trình khác vào phòng, như vậy tất cả các tiến trình đã ở trong phòng, khi đó cổng được đóng.
- Tiến trình *i* đang ở ngoài phòng chờ: 1 – dễ thấy khi đây là trạng thái đầu tiên của flag.
- Rời phòng, mở lại *cổng vào* nếu không còn ai trong *phòng chờ*: 0 – đây là trạng thái cuối cùng của flag là rời phòng.
- Đứng đợi trong *phòng chờ*: 3 – trạng thái tiếp theo sau 1 là 3, như vậy đây là sau khi vào phòng và có trạng thái đang ở trong phòng.