BÀI LÝ THUYẾT SỐ 4

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

CHƯƠNG 4: ĐỒNG BỘ HÓA TRONG HỆ PHÂN TÁN

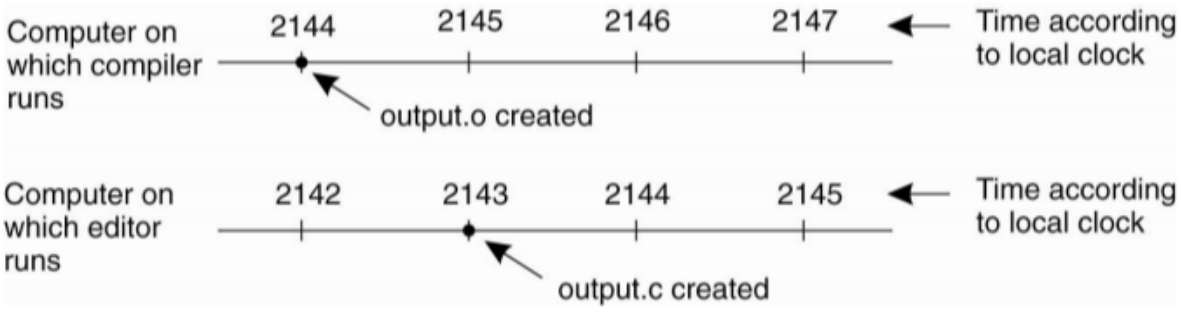
Họ và tên: Nguyễn Duy Khánh Mã lớp: 157542

MSSV: 20225019 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

Ví dụ: Vấn đề lập trình trong hệ phân tán:

Khi xây dựng chương trình C/C++ thì khi dịch, compiler chỉ dịch lại những file .c mà có file .o tương ứng có thời gian thay đổi trước thời gian của file .c. Ví dụ file output.c có thời gian là 2143, file output.o có thời gian là 2142 tức là nó đã bị thay đổi và cần biên dịch lại, còn nếu file output.o có thời gian là 2144 thì file output.c sẽ không được biên dịch lại.



Với ví dụ trên, khi lập trình viên thay đổi file output.c thì do sự lệch nhau của đồng hồ vật lý giữa hai máy tính nên chương trình compiler sẽ hiểu nhầm file .o có thời gian thay đổi sau file .c Vì vậy compiler sẽ không dịch lại file output.c

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

Trong các hệ thống phân tán (distributed system), thứ tự của các sự kiện (gửi, nhận message) cần được ghi nhận. Tuy nhiên, việc sử dụng đồng hồ vật lý có nhiều hạn chế: có thể do các máy ở các múi giờ khác nhau, có thể do đồng hồ vật lý trên mỗi máy tính có tốc độ khác nhau hoặc bị sai số thời gian, làm cho việc đồng bộ hóa và quản lý thời gian của toàn bộ hệ thống trở nên khó khăn. Ngoài ra, do độ trễ lan truyền nên thời điểm ghi nhận một message đến các node có thể khác nhau (không đảm bảo giống nhau 100%). Do đó, Lamport đề xuất sử dụng đồng hồ logic (logical clock) để đồng bộ thời gian của các sự kiện trong hệ thống phân tán. Sử dụng đồng hồ logic có thể cho kết quả nhanh chóng và hiệu quả hơn cho việc đồng bộ hóa và quản lý thời gian

trong hệ thống phân tán. Nó giúp giải quyết các vấn đề xoay quanh xung đột thời gian và thứ tự của các sự kiện một cách hiệu quả và chính xác.

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 chính của mạng không dây (wireless network) khiến việc thiết kế các giải thuật đồng bộ khác biệt so với mạng có dây là:

- Độ trễ truyền và biến động thời gian truyền cao: Trong mạng không dây, thời gian truyền gói tin không ổn định do chịu ảnh hưởng bởi nhiễu, môi trường, khoảng cách, khả năng thu phát của thiết bị.
- Tài nguyên hạn chế(năng lượng, băng thông): Các thiết bị trong mạng không dây thường là các thiết bị di động, cảm biến nên hạn chế về năng lượng và khả năng xử lý. Vì vậy các giải thuật đồng bộ cần phải gọn nhẹ, ít truyền thông, không phức tạp.

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 tiến trình Pi sử dụng tài nguyên chia sẻ SR, nó cần gửi thông điệp REQUEST đến tất cả các tiến trình khác trong hệ thống. Mỗi tiến trình nhận được REQUEST sẽ phản hồi bằng một thông điệp REPLY. Sau khi hoàn thành việc sử dụng tài nguyên, Pi gửi thông điệp RELEASE đến tất cả các tiến trình để thông báo rằng nó đã sử dụng xong tài nguyên. Do đó, tổng số thông điệp cần thiết để một tiến trình thực hiện thành công việc sử dụng SR là 3(n – 1), bao gồm (n – 1) REQUEST, (n – 1) REPLY và (n – 1) RELEASE.
b. Cải thiện trên là đúng. Với cải tiến này, số lượng thông điệp cần thiết cho mỗi lần truy cập CS giảm xuống còn 2(n − 1), bao gồm (n – 1) REQUEST, (n – 1) REPLY, nếu timestamp của REQUEST Pj vừa gửi lớn hơn timestamp của REQUEST của Pi

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 ($j \neq i$) 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 phòng đợi
flag[i] ← 1
 await(all flag[1..N]∈{0,1,2})
flag[i] ← 3
if any 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 = 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 – 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, đây là sau khi vào phòng và có trạng thái đang ở trong phòng.