

**BÀI TẬP TRÊN LỚP**  
**MÔN HỌC: HỆ PHÂN TÁN**  
**CHƯƠNG 6: ĐỒNG BỘ HÓA**

HỌ TÊN SV:

MSSV:

MÃ LỚP:

MÃ HỌC PHẦN:

**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.

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

**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?

**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:  $P_1, P_2, \dots, P_n$ . 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  $P_i$  muốn truy cập vào SR, nó sẽ quảng bá 1 thông điệp REQUEST( $t_{si}, i$ ) cho tất cả các tiến trình khác, đồng thời lưu trữ thông điệp đó vào hàng đợi của mình (queuei) trong đó  $t_{si}$  là timestamp của yêu cầu.

Khi 1 tiến trình  $P_j$  nhận được yêu cầu REQUEST( $t_{si}, i$ ) từ tiến trình  $P_i$  thì nó đưa yêu cầu đó vào hàng đợi của mình (queuej) và gửi trả lại cho  $P_i$  thông điệp REPLY.

Tiến trình  $P_i$  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  $P_i$ , 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  $P_j$  nhận được thông điệp RELEASE từ  $P_i$  thì nó sẽ xóa yêu cầu của  $P_i$  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  $P_j$  gửi yêu cầu REQUEST cho các tiến trình khác thì nhận được thông điệp REQUEST từ  $P_i$ , nếu nó nhận thấy rằng timestamp của REQUEST nó vừa gửi lớn hơn timestamp của REQUEST của  $P_i$ , nó sẽ không gửi thông điệp REPLY cho  $P_i$  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.

**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  $P_i$  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 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*