**Câu 2:**

Bàn về mục đích đạt đến sự đồng thuận (consensus problem) của một hệ thống có lỗi với sự tham gia của các tiến trình lỗi Byzantine. Xét 1 hệ thống có 7 tiến trình: A,B,C,D,E,F và G với các giả định sau:

- Các thông điệp trao đổi trong hệ thống chỉ có thể là một trong hai giá trị: 0 hoặc 1. Mục tiêu của hệ thống là cuối cùng tất cả các tiến trình không lỗi sẽ thống nhất một trong hai giá trị đó.

- Tiến trình A là coordinator.

- Các tiến trình có khả năng trao đổi thông điệp với nhau. Các tiến trình chỉ có thể nhận thông điệp từ Coordinator

(mà không gửi lại được cho coordinator).

- Hệ thống có 2 tiến trình là Byzantine (có thể là coordinator hoặc không).

Hãy thiết kế một giải thuật để cho các tiến trình không lỗi luôn đạt được đồng thuận cuối cùng và giải thích điều đó (với cả 2 trường hợp coordinator không lỗi và coordinator là Byzantine). Biết rằng bước đầu tiên của giải thuật luôn là coordinator quảng bá 1 thông điệp đến các tiến trình khác.

(Nhắc lại lý thuyết: Byzantine là một tiến trình lỗi và nó sẽ gửi các thông điệp 0 hoặc 1 không theo qui tắc nào).

**GIẢI**

Do bước đầu tiên của giải thuật là A sẽ gửi thông điệp tới tất cả các tiến trình còn lại nên ta xét các trường hợp sau:

TH 1: A không lỗi

- Sau khi A gửi thông điệp tới các tiến trình còn lại thì do A không lỗi thì thông điệp được gửi từ A là xác định đối với các tiến trình còn lại.

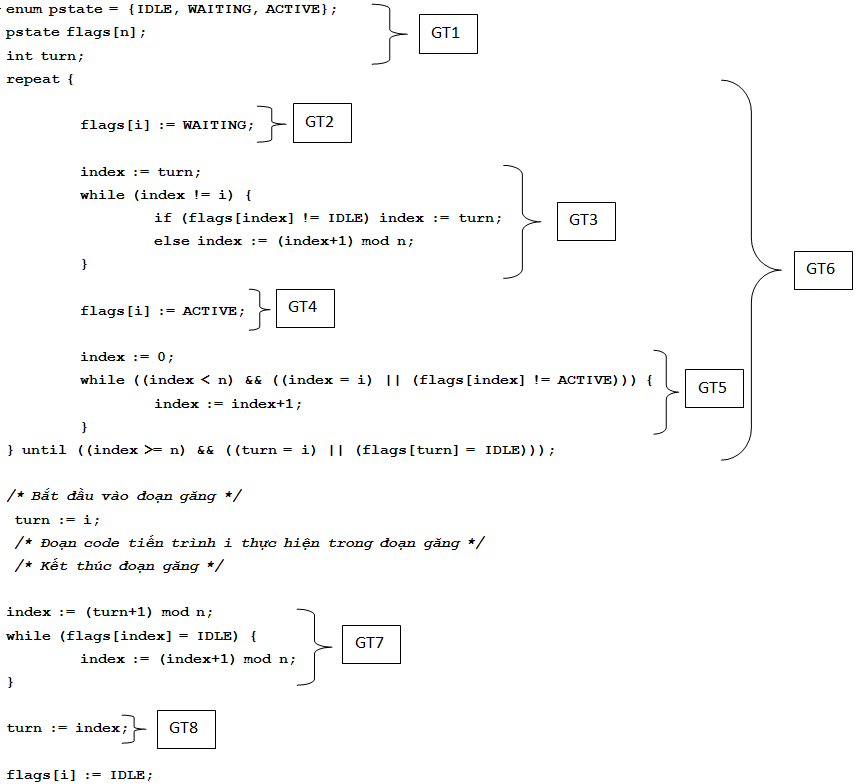
- Do các tiến trình còn lại có thể trao đổi thông tin qua lại lẫn nhau, nên trong mỗi tiến trình ta lập một bảng thống kê trạng thái mà A đã gửi cho các tiến trình còn lại.cụ thể là sau khi các tiến trình nhận được thông điệp từ A thì nó sẽ gửi thông điệp nhận được tới các tiến trình khác. các tiến trình khác sẽ lưu lập ra bảng thống kê giá trị của các thông điệp đó.

- Sau khi hoàn tất bước đó, ta bắt đầu so sánh thông điệp của các tiến trình gửi cho các tiến trình khác được lưu trên các tiến trình khác đó. Nếu có sự khác nhau về thông điệp thì tín trình gửi các thông điệp khác nhau đó là tiến trình lỗi.

TH 2: A lỗi

- Mặc dù A lỗi nhưng nó vẫn gửi thông điệp tới các tiến trình khác. tuy nhiên giá trị của thông điệp đó là ngẫu nhiên. tuy vậy, các thông điệp đó là xác định với 1 tiến trình mà nhận thông điệp đó. thực hiện tương tự các bước của trường hợp 1 ta có thể tìm được các tiến trình lỗi.

**Câu 2 (đề tài năng)**



**Hãy giải thích 8 đoạn đóng ngoặc GT. Ví dụ:**

**- GT6: đoạn repeat until này có chức năng là ...**

**GIẢI**

Thuật toán Eisenberg & McGuire algorithm

Thuật toán này giải quyết vấn đề đoạn găng cho n tiến trình

GT1: Khởi tạo, khai báo

tất cả các N tiến trình đều chung các biến

Enum pstate ={idle, waiting, active} khởi tạo kiểu dữ liệu chỉ trạng thái của tiến trình

Pstate flag[n] : mảng lưu trạng thái của n tiến trình

Int turn : được đặt một cách ngẫu nhiên , mang giá trị trong khoảng 0 …n-1 ở đầu thuật toán

Các biến flag của mỗi tiến trình được thiết lập trạng thái Waitting bất cứ khi nào để vào đoạn găng yêu cầu muốn dùng tài nguyên. flag có thể nhận 1 trong 3 giá trị IDLE , WAITTING, ACTIVE. Ban đầu , tất cả flag cho mỗi tiến trình được khởi tạo là IDLE (Nhà rỗi, để không, không chạy). Nếu được phép dùng tài nguyên thì flag chuyển thành ACTive

GT2: thông báo với tất cả là tiến trình đang đợi sử dụng tài nguyên, đang vào đoạn găng, đang ở trạng thái waitting. Đợi đến khi các tiến trình ưu tiên hơn đã ở trạng thái IDLE (nhàn rỗi), tức dùng xong tài nguyên.

GT3: thông báo với các tiến trình khác là đang trong đoạn găng, xem các tiến trình có ưu tiên lớn hơn process i đã ở trạng thái IDLE chưa, nếu chưa thì kế tiếp cho dùng tài nguyên, còn nếu đã ở trạng thái IDLE thì sẽ kiểm tra tiến trình có ưu tiên lớn hơn khác. Vòng lặp dừng lại khi tất cả tiến trình khác đã ở trạng thái IDLE (tức index =i).

GT4: Sau khi các tiến trình ưu tiên hơn process i đã ở trạng thái IDLE thì tiến trình process i bắt đầu thay trạng thái ACTIVE, tức là đến lượt nó dùng tài nguyên.

GT5: tìm kiếm trong vòng đợi xem có tiến trình nào cạnh nó mà đang trong trạng thái ACTIVE không. Vì là đây là xử lí đa luồng, cùng lúc nhiều tiến trình chạy nên phải kiểm tra liên tục.

GT6: Đoạn repeat until này có ý nghĩa là : dùng để kiểm tra xem các tiến tình có độ ưu tiên hơn process i đã hoàn toàn ở trạng thái IDLE chưa,tìm kiến có cái nào đang dùng tài nguyên không, nếu tìm kiếm thất bại và đến lượt process i or là tất cả các tiến trình ưu tiên hơn đã ở trạng thái IDLE thì process i bắt đầu vào miền găng.

GT7: tìm một tiến trình khác đầu tiên tiếp theo , theo vòng chiều kim đồng hồ, bắt đầu từ tiến trình đang dùng tài nguyên (turn), tức là process i, cái mà không ở trạng thái IDLE, và cho nó thành process tiếp được dùng tài nguyên, tức là turn kế.

GT8: trao quyền sử dụng tài nguyên cho process tiếp theo tìm ra, còn nếu không có process nào khác, thì turn sẽ dừng sau process i dùng xong, trở về trạng thái IDLE.

----------

Ps: turn chứa id tiến trình đang trong miền găng, đang sd tài nguyên, index là biến chạy .

**Câu 3:** Một giải thuật bầu chọn (election algorithm) được mô tả như sau:

Các giả định hệ thống:

- Mỗi nút trong hệ thống được gán 1 số ID từ 1 đến N. Mục đích các giải thuật bầu chọn là luôn chọn nút có ID lớn nhất làm coordinator.

- Mỗi nút sẽ lưu trữ một bảng trạng thái lưu các thông tin của các nút khác như: còn chạy tốt hay hỏng, số ID. Giả định là lỗi có thể xảy ra chỉ là lỗi yên lặng (nút hỏng sẽ không trả lời bất cứ thông điệp nào).

- Hệ thống là kết nối đầy đủ (fully connected).

- Hệ thống xác định khoảng timeouts, nếu sau khoảng thời gian này không nhận được câu trả lời từ 1 nút thì coi như nút đó hỏng.

- Hệ thống áp dụng đồng hồ logic của Lamport.

Mô tả thuật toán:

Khi một nút Pi phát hiện coordinator hiện thời bị hỏng, nó sẽ tra trong bảng trạng thái và gửi ELECTION cho nút có ID ngay dưới nút coordinator vừa hỏng. Nếu nút nhận được ELECTION không bị hỏng thì sẽ gửi OK cho Pi và sau đó quảng bá cho cả hệ thống thông điệp COORDINATOR để báo rằng mình là coordinator mới. Nếu nút đó bị hỏng, Pi tiếp tục lặp lại bước trên với lần lượt các nút có ID lớn hơn mình (thông tin lấy trong bảng trạng thái) đến bao giờ nhận được OK, hoặc ID giảm đến chính ID của mình. Nếu ID giảm đến chính ID của mình thì biết rằng mình là coordinator và gửi quảng bá COORDINATOR cho cả hệ thống.

Câu hỏi 1: Số thông điệp cần để hoàn thành bầu chọn coordinator là bao nhiêu?

Câu hỏi 2: Thiết kế bổ sung cho thuật toán trên để xử lý vấn đề khi 1 nút đang từ trạng thái hỏng và được phục hồi về trạng thái chạy. Số thông điệp cần thiết cho việc phục hồi đó là bao nhiêu? (gợi ý: xét 2 trường hợp là ID nút đó nhỏ hơn và lớn hơn coordinator hiện tại).

Câu hỏi 1:

- Gọi số thông điệp cần để hoàn thành bầu chọn coordinator: x

ID nút lớn nhât: N

ID nút coordinator hiện tại: y

ID nút bên dưới gần coordinator nhất và không hỏng : z

=> x= (y-z)+1+(N-1) = N+y-z

Giải thích: y-z: số thông điệp mà nút Pi cân gửi cho đến khi tìm thấy coordinator mới

1: Số thông điệp mà coordinator mới gửi OK cho Pi

N-1: Số thông điệp mà coordinator mới cần phải gửi quảng bá đến N-1 nút còn lại

Câu hỏi 2:

Bổ xung thuật toán: Khi nút w được phục hồi, nó sẽ gửi thông điệp tới N-1 nút còn lại trong hệ thống. Nút Pi nếu phát hiện ra thông điệp của w sẽ xảy ra 2 trường hợp:

TH1: ID của w lớn hơn coordinator hiện tại:

Pi sẽ gửi thông điệp ELECTION cho w sau đó w gửi lại OK cho Pi và gửi thông điệp quảng bá tới N-1 nút còn lại

x = (N-1)+1+(N-1)=2N

TH2: ID của w nhỏ hơn coordinator hiện tại:

Thực hiện lần lượt các bước như thuật toán đề bài

x = (N-1)+N+y-z