Chuyên đề hệ thống phân tán
Schiper-Eggli-Sandoz Algorithm

19127355 - Nguyễn Đức Đạt

Contents

1 Giới thiệu về SES algorithm

- SES Algorithm được phát triển bởi các nhà nghiên cứu Rachid Guerraoui, André Schiper và Willy Zwaenepoel vào năm 1988. Schiper-Eggli-Sandoz Algorithm (SES Algorithm) là một giải thuật phân tán được sử dụng để đạt được đồng thuận (consensus) trong một mạng phân tán trong trường hợp một số tiến trình trong mạng gặp sự cố hoặc bị đánh cắp. Điều này đặc biệt quan trọng trong các hệ thống phân tán, trong đó sự cố là không tránh khỏi được và yếu tố an toàn và độ tin cậy của hệ thống là rất quan trọng
- Khác với BSS, SES không broadcast để gửi gói tin, thay vào đó sử dụng kĩ thuật định tuyến phân tán để truyền thông tin giữa các tiến trình trong mạng phân tán

2 Tóm tắt thuật toán

- Mỗi process lưu 1 vector V_P kích thước N - 1, N số lượng processes. N là số lượng process
- Mỗi phần tử của V_P chứa (P',t): P' là id của process đích và t là vector timestamp
- Khi gửi gói tin (msg) tới 1 process khác sẽ gửi kèm V_P đến process đích
- Khi nhận được gói tin sẽ so sánh đồng hồ logical hiện tại ở process i và giá trị vector lưu trữ trạng thái trong gói tin được gửi tới.
    - Nếu V_P không chứa bất kì phần tử nào thì mà có trạng thái đã gửi tin nhắn tới Pi thì ta deliver
    - Ngược lại thì ta xét:
        - T_m > tPi thì buffer msg
        - T_m <= tPi thì deliver
    - Sau đó ta cập nhập thông tin với V_P
    - Kiểm tra msg đang được buffered, update đồng hồ logical

3 Chương trình code

- Env: win 10
- Ngôn ngữ: golang 1.20.1
- IDE: Visual studio code
- Trong file code được chia thành 4 packages:

a, main: Chạy và thực thi chương trình, input vào số lượng process

b, constant: Chứa các tham số constant của chương trình như IP ADDR, PORT, ...

c, ses: Xử lí gói tin, cập nhật các logical clock dùng thuật toán SES

d, network: Dùng để tạo server connect các process, lắng nghe và nhận các gói tin

e, powershell script: Dùng thực hiện các tác vụ chạy process trên hệ thống máy tính

4 Chi tiết chương trình

A, Dữ liệu lưu trữ

- Trong hệ thống có n process. Mỗi process sẽ mở 1 server để n – 1 process còn lại kết nối tới để gửi tin nhắn tới.

- tP_i : thời điểm logic tại process i
- VP_i : vector có kích thước n – 1 lữu trữ các thông tin khác như V_P, instanceID, numberProcess. Nó sẽ cho biết msg được gửi tới có logical clock như thế nào, cũng như logical của các gói tin khác đã gửi tới process đó.

B, Gửi tin nhắn (msg)

- Tạo ra một thông điệp với nội dung là số thứ tự của thông điệp và ID của tiến trình gửi. Thông điệp này sau đó được mã hóa và bao gồm cả độ dài của nó, sau đó được ghi vào kết nối với tiến trình đích
- Sau khi gửi thông điệp, hàm sẽ ngủ trong một khoảng thời gian ngẫu nhiên nằm trong khoảng giới hạn 100 milis và 1000 mils. Quá trình ngủ này giúp giảm độ tải của tiến trình gửi và đồng thời giúp đảm bảo sự ngẫu nhiên trong thời gian giữa các thông điệp được gửi.
- In lại các tin nhắn từ process đó tới process khác trong file log
- Cấu trúc file log:
  - Sender ID: x
  - Receiver ID: y
  - Packet Content: Message number 2 from process x
    - Sender Clock: Local logical clock: []
    - Local process vectors:
      - <P_i: []>

C, Nhận tin nhắn (msg)

- Nhận dữ liệu từ một kết nối TCP và giao tiếp với session clock để đồng bộ thời gian. Đọc dữ liệu được mã hoá sau đó giải mã chúng lại thành dữ liệu nguyên mẫu.
- Kiểm tra dữ liệu nhận được với process I tại thời điểm đó. Thực hiện thuật toán SES để so sánh sau đi in ra kết quả vào file log
- Cấu trúc file log:
  - Sender ID: x
  - Receiver ID: y
  - Packet Content: Message number 2 from process x
  - Packet Clock:
    - Time local process vectors send: []
    - Time msg local logical clock send: []
  - Time at receive logical Clock (tP_rcv):
    - []
  - Status: ?
  - Delivery Condition: [] > []

NOTE: in kết quả vào file log chứ không in ra màn hình vì quá nhiều mess được gửi và nhận → in ra cũng không thấy được

D, Cài đặt chương trình

- Ở package network thực hiện kết nối:
  - Mỗi process sẽ tự tạo 1 server ở PORT + id (process) để tạo kết nối tới n – 1 process còn lại. Server kết nối sẽ là senders, lúc đó n -1 process còn lại sẽ là receivers, chúng chạy song song với nhau.
  - Network sẽ gọi SenderWorkder để xử lý, khi senders gửi đủ 150 message tới mỗi kết nối tới server, sẽ tự động đóng kết nối từ server của connection đó.
  - Đối với server làm nhiệm vụ receiver msg thì ta sẽ đóng sau 7 giây để xử lý các gói tin chưa xử lý xong khi nhận được đủ chan string DataClose từ ReceiverWorker.
- ReceiverWorker khi nhận gói tin sẽ gọi SES trong package ses để xử lý với thuật toán SES như sau:

Giả sử máy i đã gửi gói tin tới máy j, khi máy j nhận thực hiên các thao tác sau:

- Bước 1: Kiểm tra, nếu trong V_m của gói tin không chứa (P_j , t) thì chuyển tới bước 3.
- Bước 2: Nếu có thì ta so sánh (P_j , t) trong V_m với tP_j , nếu t < tP , cho phép chuyển gói tin, ngược lại lưu vào buffer và chuyển tới bước 4. Điều kiện này không thỏa cho biết rằng máy j chưa kịp cập nhật đồng hồ đến thời điểm mà trước khi gửi gói tin máy i "mong đợi"(đồng nghĩa với việc chưa nhận đủ gói tin). Do đó khi điều kiện này không thỏa ta cần lưu lại gói tin này để xử lý sau.
- Bước 3: Cập nhật. Gồm cập nhật lại tP_j dựa trên t_m theo thuật toán cập nhật vector clock thông thường và cập nhật VP_i dựa theo Vm bằng cách: xét từng phần tử trong V_m, gọi là (P_k, t)
  - Nếu (P_k, t) không có trong V_{Pi} và k # j thì thêm (P_k, t) vào V_{Pi}
  - Nếu (P_k, t) có trong V_{Pi} và k # j thì cập nhật lại bằng cách lấy giá trị lớn nhất từng phần tử (maximum-wise).
- Bước 4: Xét lại các phần tử trong buffer, thực hiện lại các thao tác kiểm tra và cập nhật ở bước 2 và 3.
- Ở package ses thì gồm 3 phần để lưu trữ:
  - LogicClock: dùng để implement Logical clock, các hàm update Logical clock
  - VectorClock: dùng để implement Vector clock, các hàm để get, update, increase Vector clock
  - SES: dùng để implement SES

Trong struct SES sẽ có VectorClock, lock (để đồng bộ hoá), Queue.

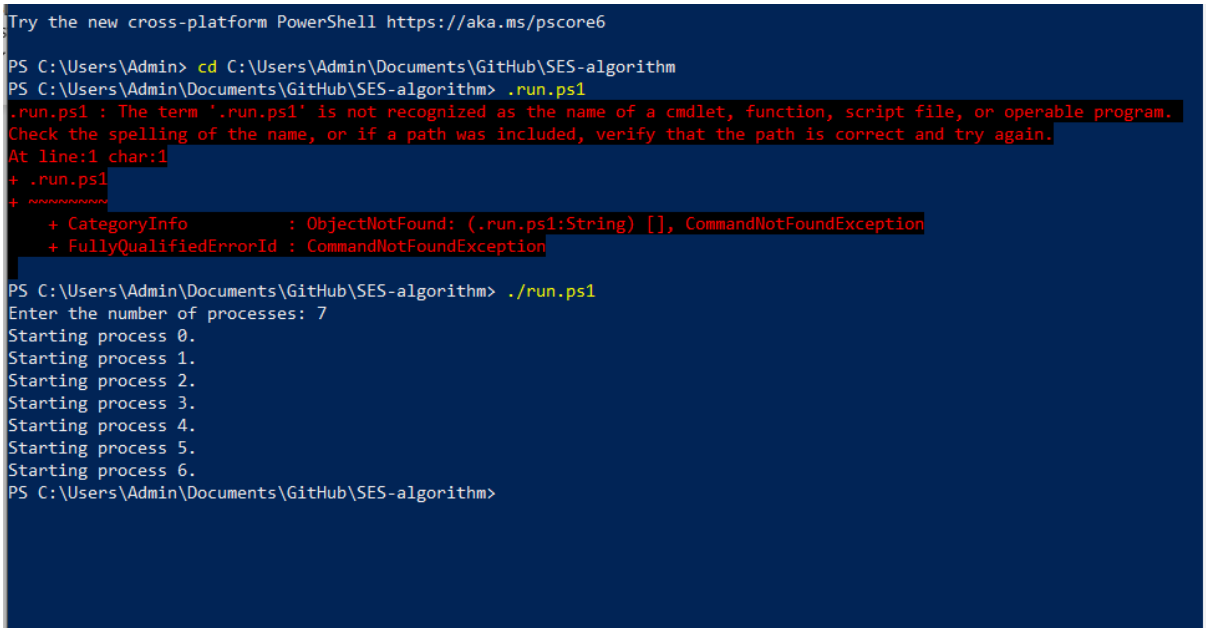
- Quá trình gửi, ta thực hiện xử lý tăng đồng hồ khi gửi msg, in ra vào logs file và cập nhật đồng hồ
- Quá trình nhận ta thực hiện kiểm tra theo thuật toán SES và cũng in ra logs file

Vì trong truyền kết nối TCP gửi file nhị phân nên các struct trong package ses đều có các hàm deserialize và serialize để biểu diễn dữ liệu dưới dạng nhị phân.

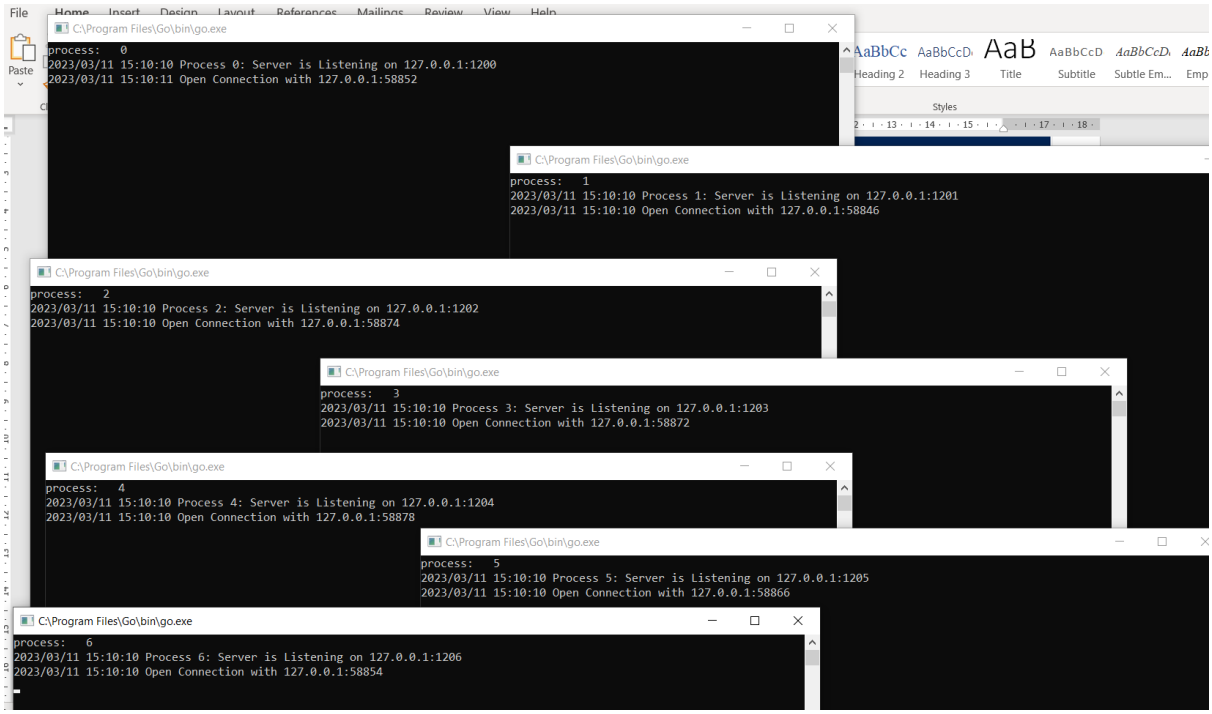
5 Chương trình

a, Cách chạy

- Ta thực hiện cd vào thư mục chứa chương trình, sau đó mở powershell để chạy file powershell script hoặc mở cmd để thực thi.
- Câu lệnh run trong powershell: ./run.ps1
- Câu lệnh run trong cmd: powershell -File ./run.ps1
- Sau đó ta nhập số process muốn thực thi:

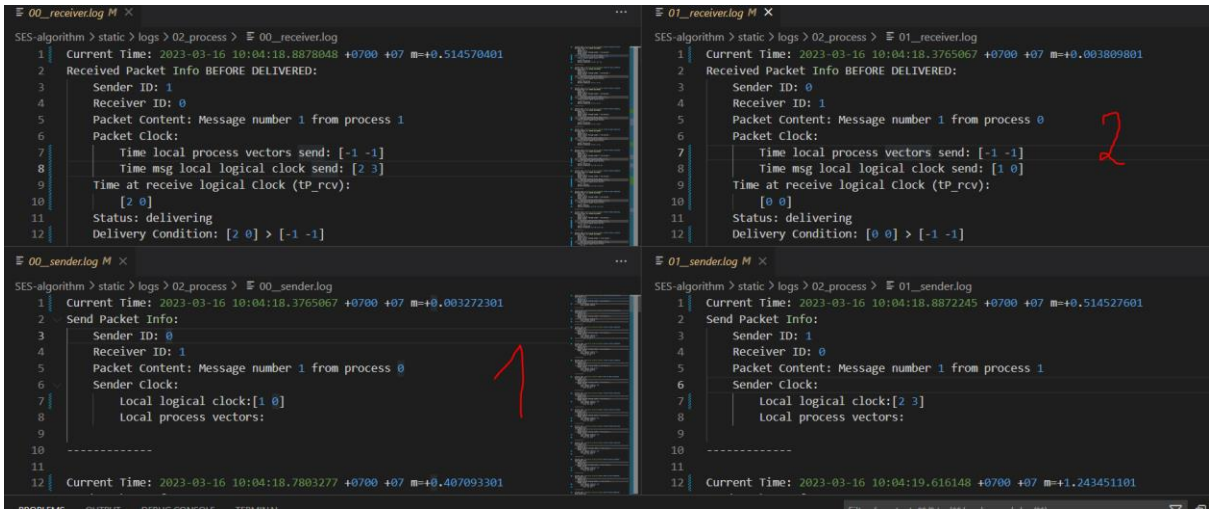


- Chương trình thực thi ta có thể thấy thời gian mở server, process id, khi hoàn thành sẽ có in close connect, nếu đủ n-1 close connect ta có thể đóng chương trình bằng signal Ctrl + C hoặc click X để tắt (vì server phải mở để tự động nhận connect nên không thiết lập được việc tự đóng server)

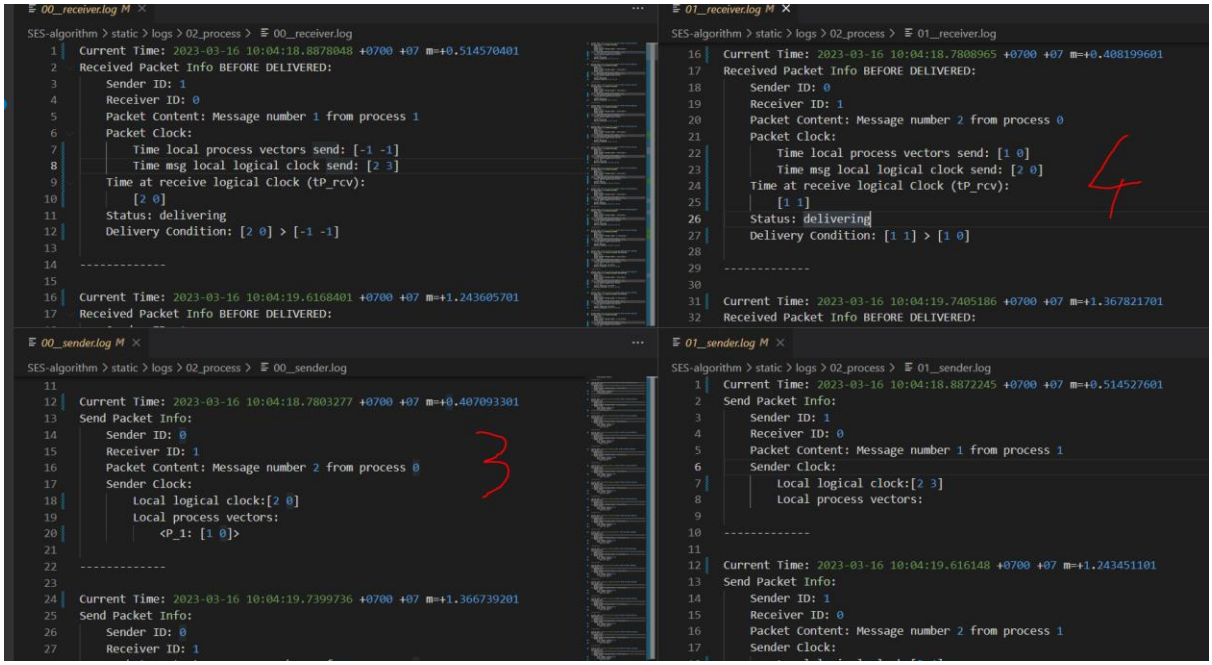


b, Giải thích chương trình

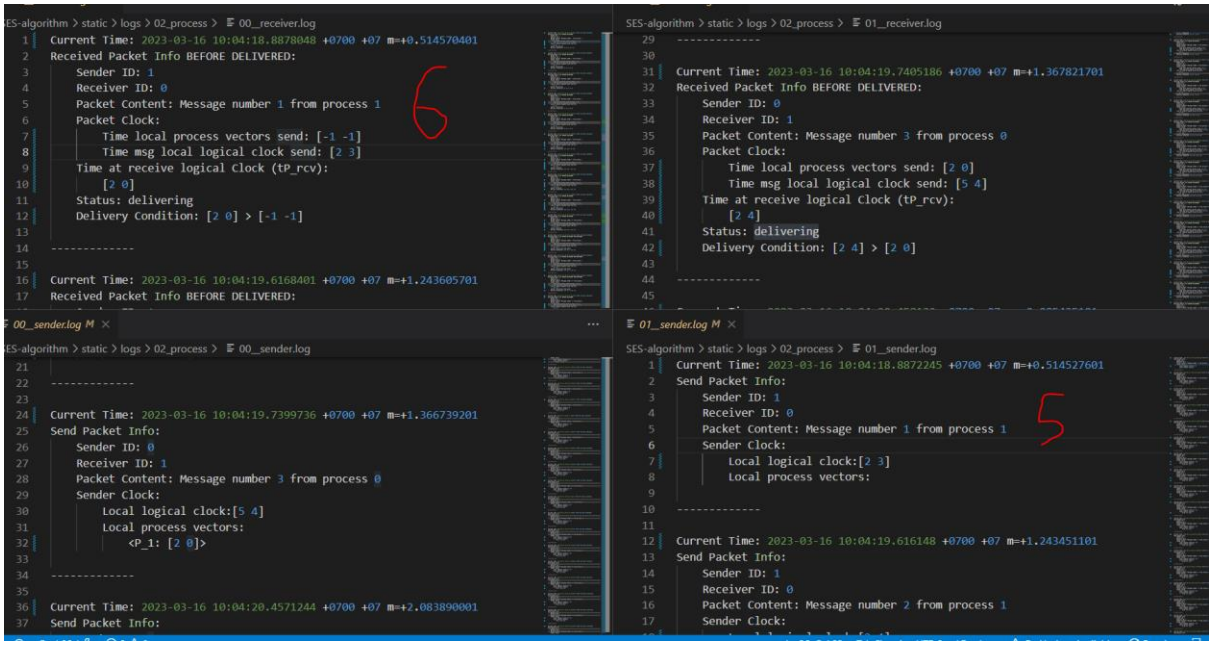
Đây là chương trình vd để giải thích với 2 process:



- Tại thời điểm p0 gửi msg 1 tới p1 thì p1 nhận msg và so sánh với local process vectors send. Nếu >= thì delivery Ta mặc định setting time local process vectors sẽ là mảng -1 nếu lúc gửi msg 1 từ process khác. Thời điểm so sánh là thời điểm nhận msg. Ta có [0, 0] > [-1, -1] ➔ delivery
- ***Trong chương trình không khi thời điểm update khi nhận delivery VD khi nhận dc msg 1 thì p1 sẽ thành [1, 1]***

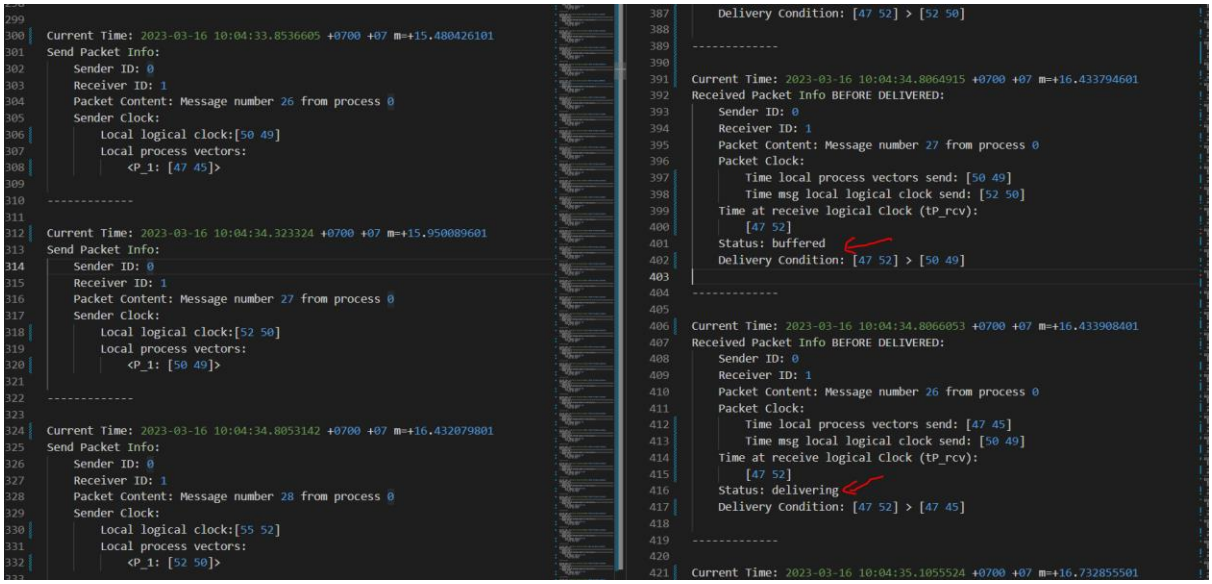


- P0 gửi msg tới p1 thì ta thấy local clock đã update thành [1,1] dùng để so sánh với local process time tại thời điểm gửi [1, 1] > [1, 0] ➔ delivery



- Tương tự cách setting trên nên ở giai đoạn 5 khi p1 gửi msg tới p0 thì ta so sánh [2, 0] > [-1, -1] ➔ delivery. Ở đây time at receive logical clock là [2, 0] vì trước đó p0 đã gửi tới p1 2 message. Tương tự với các msg còn lại

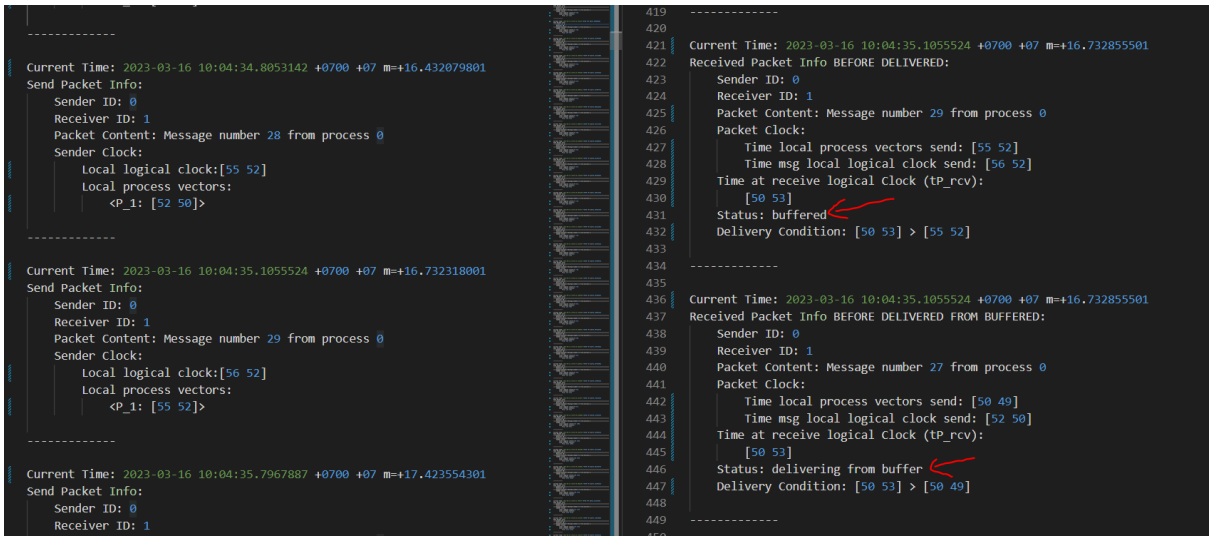
Với trường hợp buffered:



```
Current Time: 2023-03-16 10:04:33.8536605 +0700 +07 m=+15.480426101
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 26 from process 0
    Sender Clock:
        Local logical clock:[50 49]
        Local process vectors:
            <P_1: [47 45]>

-------------

Current Time: 2023-03-16 10:04:34.323324 +0700 +07 m=+15.950089601
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 27 from process 0
    Sender Clock:
        Local logical clock:[52 50]
        Local process vectors:
            <P_1: [50 49]>

-------------

Current Time: 2023-03-16 10:04:34.8053142 +0700 +07 m=+16.432079801
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 28 from process 0
    Sender Clock:
        Local logical clock:[55 52]
        Local process vectors:
            <P_1: [52 50]>
```

```
    Delivery Condition: [47 52] > [52 50]

-------------

Current Time: 2023-03-16 10:04:34.8064915 +0700 +07 m=+16.433794601
Received Packet Info BEFORE DELIVERED:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 27 from process 0
    Packet Clock:
        Time local process vectors send: [50 49]
        Time msg local logical clock send: [52 50]
    Time at receive logical Clock (tP_rcv):
        [47 52]
    Status: buffered
    Delivery Condition: [47 52] > [50 49]

-------------

Current Time: 2023-03-16 10:04:34.8066053 +0700 +07 m=+16.433908401
Received Packet Info BEFORE DELIVERED:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 26 from process 0
    Packet Clock:
        Time local process vectors send: [47 45]
        Time msg local logical clock send: [50 49]
    Time at receive logical Clock (tP_rcv):
        [47 52]
    Status: delivering
    Delivery Condition: [47 52] > [47 45]

-------------

Current Time: 2023-03-16 10:04:35.1055524 +0700 +07 m=+16.732855501
```

- Ở đây msg 27 từ p0 tới p1 xảy ra trước msg 26, ta có thể thấy tại thời điểm nhận thì time at receive logical clock là [47, 52] < [50, 49] là time local process vectors send. ➔ buffered
- Tiếp đó khi msg 26 tới thì [47, 52] >= [47, 45] ➔ delivery



```
-------------

Current Time: 2023-03-16 10:04:34.8053142 +0700 +07 m=+16.432079801
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 28 from process 0
    Sender Clock:
        Local logical clock:[55 52]
        Local process vectors:
            <P_1: [52 50]>

-------------

Current Time: 2023-03-16 10:04:35.1055524 +0700 +07 m=+16.732318001
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 29 from process 0
    Sender Clock:
        Local logical clock:[56 52]
        Local process vectors:
            <P_1: [55 52]>

-------------

Current Time: 2023-03-16 10:04:35.7967887 +0700 +07 m=+17.423554301
Send Packet Info:
    Sender ID: 0
    Receiver ID: 1
```

```
-------------

Current Time: 2023-03-16 10:04:35.1055524 +0700 +07 m=+16.732855501
Received Packet Info BEFORE DELIVERED:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 29 from process 0
    Packet Clock:
        Time local process vectors send: [55 52]
        Time msg local logical clock send: [56 52]
    Time at receive logical Clock (tP_rcv):
        [50 53]
    Status: buffered
    Delivery Condition: [50 53] > [55 52]

-------------

Current Time: 2023-03-16 10:04:35.1055524 +0700 +07 m=+16.732855501
Received Packet Info BEFORE DELIVERED FROM BUFFERED:
    Sender ID: 0
    Receiver ID: 1
    Packet Content: Message number 27 from process 0
    Packet Clock:
        Time local process vectors send: [50 49]
        Time msg local logical clock send: [52 50]
    Time at receive logical Clock (tP_rcv):
        [50 53]
    Status: delivering from buffer
    Delivery Condition: [50 53] > [50 49]

-------------
```

- Tương tự nên msg 29 bị buffered (ở trên msg 28 cũng đã buffered), ta xét msg 27 sau khi msg 26 được delivery thì [50, 53] >= [50 49] ➔ delivery from buffer

6 Nguồn tham khảo

- chat.openai.com
- day2 CDIO.pdf
- https://www.geeksforgeeks.org/schiper-eggli-sandoz-protocol/
- https://github.com/johannmeyer/Schiper-Eggli-Sandoz
- https://github.com/hatanlinh13/ses-algorithm
- https://github.com/ndhp2000/Distributed-System-Hcmus-Project-01