**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO ĐỒ ÁN**

**CHUYÊN ĐỀ HỆ THỐNG PHÂN TÁN**

**Thành phố Hồ Chí Minh**

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO ĐỒ ÁN**

**| Đề tài |**

**Cài đặt thuật toán SES**

**| Giáo viên hướng dẫn |**

**Thầy Dũng Trần Trung**

**Thầy Lê Giang Thanh**

**Thành phố Hồ Chí Minh**

**LỜI CẢM ƠN**

Để có thể hoàn thành đồ án này, chúng em xin gửi lời cảm ơn chân thành tới thầy Dũng Trần Trung, thầy Lê Giang Thanh khoa Công Nghệ Thông Tin, trường Đại học Khoa học Tự nhiên vì đã hỗ trợ tụi em trong suốt quãng thời gian làm đồ án, giúp đồ án này được hoàn thiện nhất có thể.

Do thời gian có hạn cũng như các kiến thức của chúng em vẫn có nhiều hạn chế, rất khó để tụi em tránh khỏi những sai sót khi hoàn thiện đồ án cũng như làm báo cáo. Vì vậy rất mong có thể nhận được những lời góp ý từ các thầy cô để chúng em có thể có thêm kiến thức để hoàn thành những đồ án tiếp theo.

Chúng em xin chân thành cảm ơn.

Mục Lục

[1. Giới thiệu: 9](#_Toc130168064)

[2. Thành viên và bảng phân công: 9](#_Toc130168065)

[3. Mô tả chi tiết hệ thống: 9](#_Toc130168066)

[4. Chức năng chi tiết của từng Class: 10](#_Toc130168067)

[a) Hàm main() 10](#_Toc130168068)

[b) class myProcess 10](#_Toc130168069)

[c) class SchiperEggliSandoz 11](#_Toc130168070)

[d) class Message 11](#_Toc130168071)

[e) class S 11](#_Toc130168072)

[f) class SBuffer 11](#_Toc130168073)

[g) class VectorClock 12](#_Toc130168074)

**BÁO CÁO**

# Giới thiệu:

* Chương trình chạy trên ngôn ngữ Java. Khi chạy chương trình sẽ tạo ra 7 process và tự động gửi tới 6 process còn lại 1600 messages.
* Mỗi messages có nội dung là “message 1”, “message 2”, …
* Thời gian các message gửi đi sẽ là ngẫu nhiên. Cũng chính vì thế sẽ có hiện tượng buffer và thông tin về timestamp, buffer, delivery hay trạng thái của messages cũng sẽ được in ra terminal và file log.
* Dưới đây trình bày cách cài đặt thuật toán SES (Schiper-Eggli-Sandoz).

# Thành viên và bảng phân công:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Nhiệm vụ** |
| 1 | Nguyễn Gia Minh | 19126054 |  |
| 2 | Lê Trung Hiếu | 19126044 |  |

# Mô tả chi tiết hệ thống:

Chương trình này dùng thuật toán SES để giải quyết vấn đề đồng bộ hóa các sự kiện trong hệ thống phân tán. Nó tạo ra process và sử dụng class SchiperEggliSandoz để gửi và nhận thông điệp giữa các process.

Các object MyProcess được sử dụng để thực hiện việc gửi message giữa các process, mỗi object MyProcess đại diện cho một process và chứa thông tin về các process đích (process cần gửi tới) và messages cần được gửi đi.

Hệ thống được tổ chức gồm 6 class chính dưới đây:

* **main**(): Hàm chính của chương trình. Nó bắt đầu bằng việc tạo một object Registry để tạo ra một máy chủ RMI (Remote Method Invocation cho phép các tiến trình chạy trên các máy khác nhau giao tiếp với nhau) và sau đó tạo một số process SchiperEggliSandoz và các class MyProcess. Mỗi process gửi messages đến các process khác bằng MyProcess().
* **MyProcess**: Đây là class dùng Runnable và đại diện cho tiến trình được thực thi. Nó có đầu vào là ( chiperEggliSandoz, danh sách các ID process đích tới, các messages, delayTime). Trong phương thức run(), nó gửi các messages đến các process đích đến.
* **SchiperEggliSandoz** là class chính triển khai SchiperEggliSandoz, bao gồm các hàm để gửi và nhận tin nhắn, và xử lý các tin nhắn trong bộ đệm.
* **Message**: Lớp này đại diện cho một messages được gửi đi. Nó bao gồm một chuỗi thông điệp, một List<S> và một int[ ] timeStamp.
* **S**: Class này được sử dụng trong việc triển khai đồng bộ hóa tiến trình bằng thuật toán SES. Nó được truyền input gồm (ID, timeStamp). Các function của lớp này cho phép setup ID process và timeStamp và sao chép object S.
* **SBuffer**: định nghĩa các phương thức để dùng cho class S.
* **VectorClock** dùng để đồng bộ hóa thời gian được sử dụng trong hệ thống và theo dõi các sự kiện xảy ra trong các mốc thời gian khác nhau.

# Chức năng chi tiết của từng Class:

## Hàm main()

Khi bắt đầu chương trình sẽ khởi chạy hàm main().

Hàm này sẽ bắt đầu tạo ra các process và gửi tin nhắn giữa chúng. Đầu tiên, nó tạo ra một object Registry với port 8080.

Sau đó, nó tạo ra các thứ sau:

* Mảng 2 chiều destIDs để lưu trữ danh sách các Process nơi mà cần gửi messages.
* Mảng 2 chiều Messages để lưu trữ các tin nhắn cần gửi
* Mảng 2 chiều thời gian delays (độ trễ giữa việc gửi các tin nhắn, ở đây là Random).

Sau đó, nó tạo ra một số luồng MyProcess để gửi các tin nhắn theo mảng destIDs được input. Cuối cùng, nó start tất cả các luồng đó.

## class myProcess

Class này thực hiện gửi các tin nhắn giữa các quy trình. Mỗi object MyProcess là 1 process và sẽ gửi tin nhắn đến một hoặc nhiều quy trình khác (ở đây là gửi broadcast cho tất cả process còn lại).

Các tham số đầu vào là (destIDs, messages, delays và process).

Trong hàm run(), vòng lặp đầu tiên duyệt qua các phần tử của mảng destIDs.

Đối với mỗi phần tử, vòng lặp thứ hai hàm process.send() sẽ được sử dụng để gửi thông điệp đến process với ID tương ứng với messages và thời gian delay như đã input vào.

Nếu có lỗi xảy ra, thông báo lỗi sẽ được in ra màn hình vì đã có try, catch error.

## class SchiperEggliSandoz

Các hàm chính trong class này là:

* **receive**(Message m): nhận một mesage. Nếu đáp ứng được thì tin nhắn được gửi đi. Nếu không, nó được thêm vào bộ đệm.
* **send**(int destinationID, String destination, String message): gửi một tin nhắn đến process với định danh destinationID và địa chỉ destination. Tin nhắn bao gồm nội dung message, bộ đệm S hiện tại và đồng hồ vector của process hiện đang gửi.
* **send**(int destinationID, String destination, String message, int delay): giống với phương thức send() trên, nhưng có thể thiết lập một khoảng thời gian trễ trước khi tin nhắn được gửi đi (random thời gian delay).
* **checkBuffer**(): kiểm tra bộ đệm để xem có tin nhắn nào có thể gửi đi hay không.
* **deliver**(Message m): giao nhận một tin nhắn. Sau đó cập nhật bộ đệm S hiện tại, đồng hồ vector và thực hiện tăng đồng hồ vector của process hiện tại lên.

## class Message

Class này chứa các giá trị sau (messages, 1 list Sbuffer và mảng timeStamp)

Trong đây có hàm toString() để có thể in ra messages và trạng thái hiện tại của messages đó.

## class S

Class này dùng trong giao thức Schiper-Eggli-Sandoz.

Nó chứa hai thuộc tính: pid (đại diện cho ID của tiến trình tương ứng) và timeStamp (chứa thời gian tương ứng của tiến trình đó)

## class SBuffer

Các hàm trong class SBuffer được sử dụng để thêm mới hoặc cập nhật thông tin về 1 message trong bộ đệm, xác định điều kiện để giao tiếp và trộn các bộ đệm.

* **insert**(List<S> sBuffer, S newS): mục đích cập nhật hoặc thêm một phần tử mới vào buffer S

Tham số:

* + sBuffer: buffer S cần cập nhật
  + newS: phần tử S mới cần thêm vào hoặc cập nhật timestamp của phần tử S có sẵn trong buffer S

Hàm này duyệt qua từng phần tử S trong buffer S để tìm phần tử S có pid giống với newS. Nếu tìm thấy, cập nhật timestamp của phần tử S đó. Nếu không tìm thấy, thêm newS vào buffer S.

* **deliveryCondition**(List<S> sBufferMesg, S currProcess): kiểm tra điều kiện để có message.

Tham số:

* + sBufferMesg: buffer S của message
  + currProcess: tiến trình hiện tại đang xử lý message

Duyệt qua từng phần tử S trong buffer S của message để tìm phần tử S có pid giống với currProcess. Nếu tìm thấy, so sánh timestamp của phần tử S đó và timestamp của currProcess. Nếu timestamp của phần tử S nhỏ hơn hoặc bằng timestamp của currProcess, trả về chính nó. Nếu không tìm thấy phần tử S trong buffer S của message, trả về true.

* **merge**(List<S> ownBuffer, List<S> incomingBuffer): ghép trộn buffer S của tiến trình hiện tại với buffer S của message nhận được.

Tham số:

* + ownBuffer: buffer S của tiến trình hiện tại
  + incomingBuffer: buffer S của message nhận được

Duyệt qua từng phần tử S trong buffer S của message nhận được để thêm hoặc cập nhật timestamp vào buffer S của tiến trình hiện tại.

Nếu phần tử S có cùng pid trong cả hai buffer, cập nhật timestamp của phần tử S đó bằng timestamp lớn hơn giữa phần tử S của tiến trình hiện tại và phần tử S của message.

Nếu không có phần tử S nào trong buffer S của tiến trình hiện tại có cùng pid với phần tử S trong buffer S của message, thêm phần tử S đó vào buffer S của tiến trình hiện tại.

* **toString**(List<S> sBuffer): in ra trả về chuỗi các phần tử trong buffer S.

## class VectorClock

Class VectorClock hỗ trợ các phép toán so sánh và tính toán giá trị lớn nhất giữa các vector trong cùng thời gian, cùng với việc chuyển đổi vector đồng thời sang chuỗi để dễ dàng in ra màn hình và fife log.

* **strictlyLessThan**(int[] timeStamp1, int[] timeStamp2): kiểm tra xem timeStamp1 có nghiêm ngặt bé hơn timeStamp2 hay không. Nếu có, phương thức trả về true, ngược lại trả về false.
* **lessThanEqualTo**(int[] timeStamp1, int[] timeStamp2): kiểm tra xem timeStamp1 có bé hơn hoặc bằng timeStamp2 hay không. Nếu có, phương thức trả về true, ngược lại trả về false. Hàm này được dùng để check condition trong class SBuffer
* **max**(int[] timeStamp1, int[] timeStamp2): trả về vector clock lớn nhất giữa timeStamp1 và timeStamp2. Để làm điều này, phương thức duyệt qua từng phần tử trong vector clock, so sánh giá trị của nó trong timeStamp1 và timeStamp2, lấy giá trị lớn nhất của hai giá trị và gán vào vector clock đầu tiên.
* **toString**(int[] vectorClock): trả về một chuỗi biểu diễn vector đồng thời dưới dạng mảng để chuyển đổi vector clock thành chuỗi, sử dụng phương thức Arrays.toString(vectorClock) để thực hiện việc này.