**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 chat GRPC**

**| 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: 7](#_Toc130246207)

[2. Thành viên và bảng phân công: 7](#_Toc130246208)

[3. Cách chạy chương trình: 7](#_Toc130246209)

[4. Mô tả chi tiết hệ thống: 8](#_Toc130246210)

[5. Chức năng chi tiết của từng Class: 9](#_Toc130246211)

[a) Hàm main() 9](#_Toc130246212)

[b) class MyProcess 9](#_Toc130246213)

[c) class SES\_Algorithm 9](#_Toc130246214)

[d) class Message 10](#_Toc130246215)

[e) class BufferModel 10](#_Toc130246216)

[f) class Buffer 10](#_Toc130246217)

[g) class Clock 11](#_Toc130246218)

**BÁO CÁO**

# Giới thiệu:

* Chương trình chạy trên ngôn ngữ Java và dùng GRPC.
* Gồm có Server và Client. Khởi chạy Sever trước sau đó chạy những Client thì các Client sẽ kết nổi được với nhau.
* Mỗi Client sẽ gồm có 3 chức năng (Gửi tin nhắn, like, disconnect).
* Khi 1 Client gửi tin nhắn thì sẽ được broadcast cho những Client còn lại thấy tin nhắn.
* Các Client khác có thể like tin nhắn đó. Và Client sau khi gửi tin nhắn thì tin nhắn đó cần có ít nhất 2 like trở lên thì mới có thể nhắn tin tiếp theo.l
* Chương trình có thể chạy với số Client không giới hạn, phụ thuộc vào phần cứng.

# Thành viên và bảng phân công:

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Nhiệm vụ** |
| 1 | Nguyễn Gia Minh | 19126054 | * Làm báo cáo. * Xử lý thuật toán. * Gửi nhận buffer và in ra tin nhắn. * Xử lý Vector Clock và gửi nhận thông tin đồng bộ hóa. * Clean code. |
| 2 | Lê Trung Hiếu | 19126044 | * Xử lý thuật toán. * Xử lý đầu ra đầu vào. * Xử lý gửi nhận thông tin qua process và thread . * Lọc data. * Validate code. |

# Cách chạy chương trình:

* Đầu tiên chạy file ChatServer.java
* Sau đó chạy file ChatClient.java
* Ở cmd ChatClient.java thì điền thông tin như sau:
* Name: tùy chọn
* Server address: localhost
* Server port: 50051
* Nếu như thấy thông báo “Connected to ChatServer” thì có nghĩa là Client đã kết nối được với Server.

Graphical user interface, application

Description automatically generated

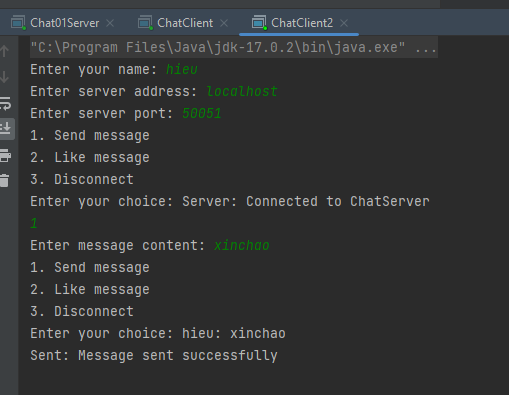
Text

Description automatically generated

# Cách dùng:

Chức năng 1 (Send message):

* Khi nhấn 1 và enter thì ta sẽ gõ tin nhắn
* Nếu tin nhắn được gửi broadcast cho tất cả Client thành công thì sẽ có thông báo “Message sent successfully”
* Và bên phía Client còn lại sẽ thấy được tin nhắn như hình dưới đây.



Text

Description automatically generated

Chức năng 2 (Like message):

* Khi bấm 2 và enter, ta sẽ nhập theo cú pháp sender + content.
* Ví dụ dưới đây là like message của “hieu” và nội dung tin nhắn là “xinchao”
* Nếu nhận được thông báo là “Message liked successfully” có nghĩa là đã like thành công.

Text

Description automatically generated

Chức năng 3 (Disconnect):

* Khi chọn chức năng này Client đó sẽ thoát khỏi cuộc trò chuyện và tất cả Client khác sẽ nhận được thông báo như sau.

Graphical user interface, text, application

Description automatically generated

# 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 MAIN 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 SES\_Algorithm 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 (SES\_Algorithm, danh sách các ID process đích tới, các messages, delayTime). Trong phương thức run(), nó gửi các messages đến các process đích đến.
* **SES\_Algorithm** là class chính triển khai SES\_Algorithm, 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<BufferModel> và một int[ ] timeStamp.
* **BufferModel**: 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 BufferModel.
* **Buffer**: định nghĩa các phương thức để dùng cho class BufferModel.
* **Clock** 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 dưới dạng vector.

# 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 SES\_Algorithm

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 Buffer và mảng timeStamp)

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

## class BufferModel

Class này dùng trong giao thức SES\_Algorithm.

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 Buffer

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< BufferModel > bufferModelBuffer, BufferModel newBufferModel): mục đích cập nhật hoặc thêm một phần tử mới vào buffer của BufferModel

Tham số:

* + bufferModelBuffer: buffer của BufferModel cần cập nhật
  + newBufferModel: phần tử BuferModel mới cần thêm vào hoặc cập nhật timestamp của phần tử BuferModel có sẵn trong buffer của BuferModel.

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

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

Tham số:

* + bufferModelBufferMesg: buffer của BufferModel của message
  + currProcess: tiến trình hiện tại đang xử lý message

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

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

Tham số:

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

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

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

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

* **toString**(List< BufferModel > Buffer): in ra trả về chuỗi các phần tử trong buffer của BufferModel.

## class Clock

Class Clock 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.

* **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 Buffer
* **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.