Đại Học Quốc Gia Hà Nội

**Trường Đại Học Công Nghệ**

🙥🕮🙧



BÁO CÁO LẬP TRÌNH MẠNG

SỬ DỤNG GIAO THỨC MQTT CHO QUẢN LÝ/THEO DÕI NHIỆT ĐỘ PHÒNG KHO

Lớp: 2223II\_INT3304\_20 – Lập trình mạng

Giảng viên: Thầy Nguyễn Ngọc Tân

**Nhóm 11**

Thành viên:

Nông Ngọc Sơn – 20021430

Nguyễn Đức Chính – 20021306

Phan Anh Quân -  20020325

Đặng Việt Linh – 20021383

Đỗ Minh Quân – 20021414

Mục lục

[**1.** **Ý tưởng và kế hoạch** 3](#_Toc136174982)

[**2.** **Giao thức MQTT** 3](#_Toc136174983)

[2.1 Tổng quát 3](#_Toc136174984)

[2.2 Mô hình Xuất bản/Đăng ký 4](#_Toc136174985)

[2.3 Tin nhắn 5](#_Toc136174986)

[2.4 Chủ đề 5](#_Toc136174987)

[2.5 Broker 6](#_Toc136174988)

[2.6 Packet Format 7](#_Toc136174989)

[**3.** **Kiến trúc phần mềm** 9](#_Toc136174990)

[3.1 Gateway 9](#_Toc136174991)

[3.2 Node 10](#_Toc136174992)

[3.3 Linechar 10](#_Toc136174993)

[**4.** **Kiểm thử** 10](#_Toc136174994)

[**5.** **Các thư viện được sử dụng** 21](#_Toc136174995)

[5.1 Thư viện Gnuplot trong C++ 21](#_Toc136174996)

[5.2 Thư viện sys/socket.h 23](#_Toc136174997)

[5.3 Thư viện arpa/inet.h 25](#_Toc136174998)

[5.4 Thư viện unistd.h 27](#_Toc136174999)

[5.5 Thư viện pthread.h 28](#_Toc136175000)

[5.6 Thư viện chrono 29](#_Toc136175001)

[5.7 Thư viện random 30](#_Toc136175002)

[5.8 Thư viện thread 31](#_Toc136175003)

1. **Ý tưởng và kế hoạch**

Sự phổ biến của công nghệ IoT và sự kết nối liên mạng đã mở ra cánh cửa cho khả năng quản lý một loạt các hệ thống, bao gồm cả việc theo dõi nhiệt độ phòng kho. Khả năng này mang lại nhiều lợi ích đáng kể, bao gồm tiết kiệm thời gian, tăng cường sự linh hoạt và cải thiện hiệu quả quản lý.

Ví dụ: Một công ty vận chuyển và lưu trữ hàng hóa vì tính chất đa dạng của hàng hóa, họ xây dựng các phòng kho khác nhau. Để đảm bảo chất lượng lưu trữ, nhiệt độ phòng kho cần được kiểm soát và giám sát một cách hiệu quả và tiết kiệm tài nguyên cũng như đơn giản trong cách tiếp cận, sử dụng và sửa chữa trong khoảng thời gian dịch covid-19. Công ty có thể cài đặt các cảm biến nhiệt độ trong mỗi phòng kho và kết nối chúng với một hệ thống giám sát đơn giản tiết kiệm. Nhờ đó, người quản lý có thể theo dõi nhiệt độ, cảnh báo khi có sai lệch nhiệt độ và điều chỉnh nhiệt độ khi cần thiết, ngay cả khi họ chỉ có một nhân viên giám sát. Điều này giúp tiết kiệm thời gian, tăng cường sự linh hoạt an toàn.

Và để thực hiện việc tạo ra một chương trình quản lý theo dõi nhiệt độ phòng kho và đáp ứng những điều kiện khó khăn, một trong những giao thức mạng được lựa chọn cho quá trình phát triển một hệ thống theo dõi nhiệt độ, đó là giao thức mạng MQTT.

1. **Giao thức MQTT**

## 2.1 Tổng quát

MQTT, viết tắt của Message Queuing Telemetry Transport, là một giao thức nhắn tin đạt tiêu chuẩn của OASIS (Organization for the Advancement of Structured Information Standards) dành cho Internet of Things (IoT). MQTT được thiết kế là một phương thức nhắn tin cực kỳ nhẹ theo mô hình xuất bản/đăng ký (publish/subscribe) phù hợp để kết nối các thiết bị từ xa với mã nguồn nhỏ gọn, đồng thời có thể truyền và nhận dữ liệu qua mạng có tài nguyên và băng thông hạn chế. Hiện nay, MQTT được sử dụng trong nhiều ngành công nghiệp khác nhau như ô tô, sản xuất, viễn thông, dầu khí, v.v.

***Tại sao lại sử dụng MQTT?***

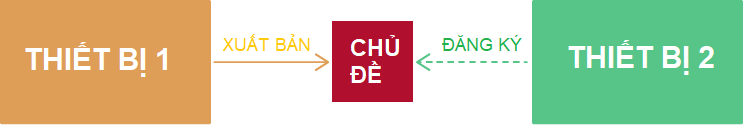
* Nhỏ gọn nhưng hiệu quả: Các MQTT clients rất nhỏ gọn và việc triển khai MQTT trên thiết bị IoT yêu cầu lượng tài nguyên tối thiểu, do đó có thể sử dụng được trên các vi điều khiển nhỏ. Vì vậy, những headers của tin nhắn MQTT nhỏ gọn để tối ưu hóa băng thông mạng. Ví dụ: một thông điệp kiểm soát MQTT tối thiểu có thể nhỏ đến hai byte dữ liệu. Tiêu đề thông điệp MQTT cũng nhỏ nên bạn có thể tối ưu hóa băng thông mạng.
* Kết nối đa chiều: MQTT cho phép truyền tin nhắn giữa thiết bị và đám mây (device to cloud) và từ đám mây đến thiết bị (cloud to device). Điều này giúp việc phát sóng tin nhắn đến một nhóm các thiết bị khác nhau thể trở nên dễ dàng.
* Quy mô linh hoạt: MQTT có thể mở rộng để kết nối với hàng triệu thiết bị IoT. Đồng thời việc triển khai MQTT yêu cầu một lượng mã tối thiểu tiêu thụ rất ít năng lượng trong các hoạt động. Giao thức này cũng có các tính năng tích hợp để hỗ trợ giao tiếp với một lượng lớn các thiết bị IoT. Do đó, bạn có thể triển khai giao thức MQTT để kết nối với hàng triệu thiết bị này.
* Truyền tin đáng tin cậy: Độ tin cậy của việc gửi và nhận tin nhắn là cực kỳ quan trọng trong nhiều trường hợp sử dụng IoT. Nhiều thiết bị IoT kết nối qua mạng di động không đáng tin cậy với băng thông thấp và độ trễ cao, do đó MQTT có các tính năng tích hợp giúp giảm thời gian thiết bị IoT cần để kết nối lại với đám mây. Đó là lý do tại sao MQTT có 3 mức độ chất lượng dịch vụ đã được định nghĩa: 0 - tối đa một lần,

1 - ít nhất một lần, 2 - chỉ một lần.

* Hỗ trợ tốt: Các mạng không ổn định Nhiều thiết bị IoT kết nối thông qua các mạng di động không đáng tin cậy. Khả năng hỗ trợ phiên liên tục của MQTT giảm thời gian để kết nối lại client với broker. Một số ngôn ngữ như Python có hỗ trợ rộng rãi cho việc triển khai giao thức MQTT. Do đó, các nhà phát triển có thể nhanh chóng triển khai giao thức này mà ít cần phải viết mã trong bất kỳ loại ứng dụng nào.
* Tính bảo mật: MQTT giúp dễ dàng mã hóa tin nhắn bằng TLS và xác thực client bằng các giao thức xác thực hiện đại như OAuth. MQTT giúp các nhà phát triển dễ dàng mã hóa thông điệp và xác thực thiết bị và người dùng bằng các giao thức xác thực hiện đại, chẳng hạn như OAuth, TLS1.3, v.v.

## 2.2 Mô hình Xuất bản/Đăng ký

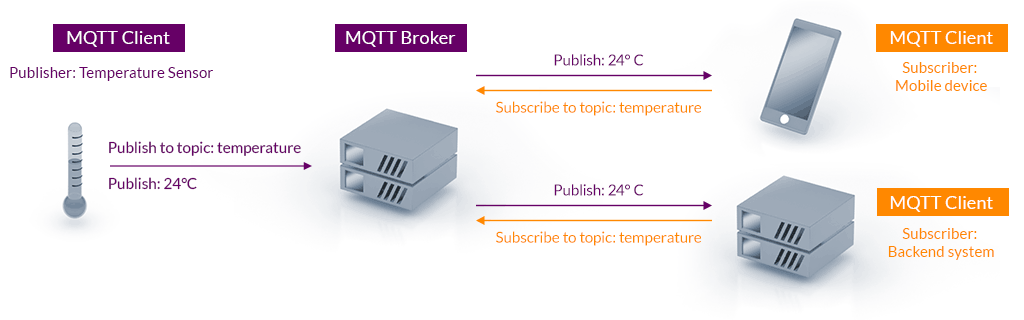
Khái niệm cần biết đầu tiên là về mô hình Xuất bản/Đăng ký (Publish/Subscribe**)**. Trong mô hình này, một thiết bị có thể xuất bản một tin nhắn về một chủ đề hoặc có thể đăng ký một chủ đề cụ thể để có thể nhận tin nhắn. Hình ảnh dưới đây sẽ là minh họa về mô hình này:



Như ở trong hình vẽ trên, ta thấy rằng:

* Thiết bị 1 xuất bản ra một chủ đề
* Thiết bị 2 đăng ký cùng với chủ đề mà thiết bị 1 xuất bản ra.
* Do đó, thiết bị 2 nhận được tin nhắn

Dưới đây là mô hình Xuất bản/Đăng ký theo đề tài theo dõi nhiệt độ phòng kho theo ý tưởng của nhóm chúng em:



## 2.3 Tin nhắn

Tin nhắn (Messages**)** là thông tin mà bạn muốn trao đổi giữa các thiết bị của mình. Nó có thể là một tin nhắn như lệnh (command) hoặc dữ liệu (data).

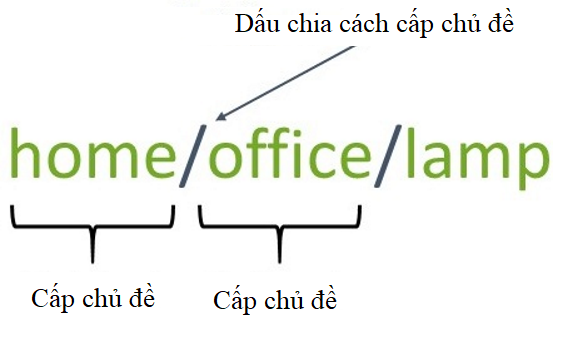
## 2.4 Chủ đề

Chủ đề (Topics) là cách bạn đăng ký sự quan tâm đến các tin nhắn đến hoặc là cách bạn chỉ định nơi bạn muốn xuất bản tin nhắn. Topic có thể coi như một “đường truyền” logic giữa 2 điểm là publisher và subscriber

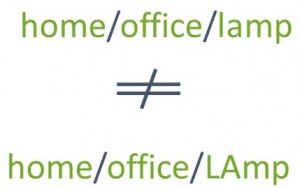
Các chủ đề được xây dựng một cách sao cho người dùng có tùy chọn đăng ký ở level 1, level 2, level 3, level 4 hoặc dữ liệu cảm biến cá nhân. Khi đăng ký cho mỗi level dữ liệu cảm biến, client cần chỉ định cấu trúc của các ID. Ví dụ, để đăng ký dữ liệu cảm biến ở mức độ 4, client cần chỉ định id level / id level / id level 3 / id level 4. Người dùng có thể đăng ký cho bất kỳ loại cảm biến nào bằng cách chỉ định vai trò của cảm biến là phần cuối của chủ đề.

Nếu người dùng không chỉ định vai trò, client sẽ được đăng ký cho tất cả các loại cảm biến ở mức đó. Người dùng cũng có thể chỉ định id của cảm biến mà họ muốn đăng ký. Trong trường hợp đó, họ cần chỉ định toàn bộ cấu trúc của cảm biến, bắt đầu từ id của dự án và kết thúc bằng id của cảm biến.

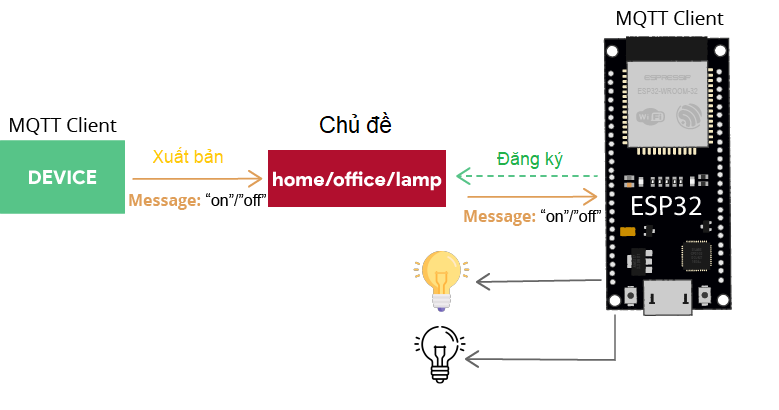
Chủ đề được đại diện bằng chuỗi, các chuỗi được phân tách bằng dấu gạch chéo xuôi (/). Mỗi dấu gạch chéo xuôi đại diện cho một cấp chủ đề. Dưới đây là một ví dụ về cách bạn có thể tạo một chủ đề cho một bóng đèn trong văn phòng tại nhà của bạn:



**Lưu ý:** Chủ đề phân biệt chữ hoa chữ thường, điều này làm cho hai chủ đề sau khác nhau:

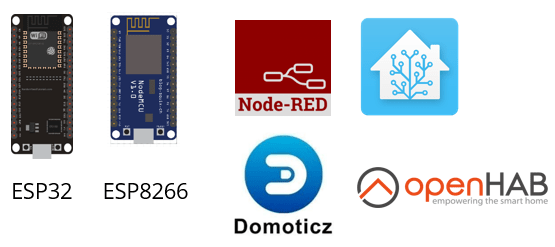


Nếu bạn muốn bật/tắt một chiếc đèn trong phòng nhà bạn bằng cách sử dụng MQTT thì bạn có thể liên tưởng đến ví dụ dưới đây:



1. Một thiết bị gửi đi tín hiệu “on” và “off” messages đến chủ đề **home/office/lamp**
2. Bạn có một thiết bị có khả năng kiểm soát chiếc đèn (nó có thể là ESP32, ESP8266 hoặc bất kỳ bo mạch hoặc thiết bị nào khác). Thiết bị ESP32 có khả năng kiểm soát chiếc đèn cũng đăng ký đến cùng một chủ đề: **home/office/lamp**.
3. Vì vậy, khi một tin nhắn/tín hiệu được xuất bản/gửi lên chủ đề đó, ESP32 nhận được tín hiệu “on” hoặc là “off” và sẽ hành động dựa trên tín hiệu nhận được

Thiết bị đang xuất bản các tin nhắn có thể là ESP32, ESP8266, hoặc một nền tảng điều khiển tự động hóa nhà có hỗ trợ MQTT như Node-RED, Home Assistant, Domoticz hoặc OpenHAB.



*(Các thiết bị và nhà hỗ trợ được nhắc đến ở trên)*

## 2.5 Broker

Định nghĩa cuối cùng nhưng cũng không kém phần quan trọng là về Broker (Máy chủ môi giới)

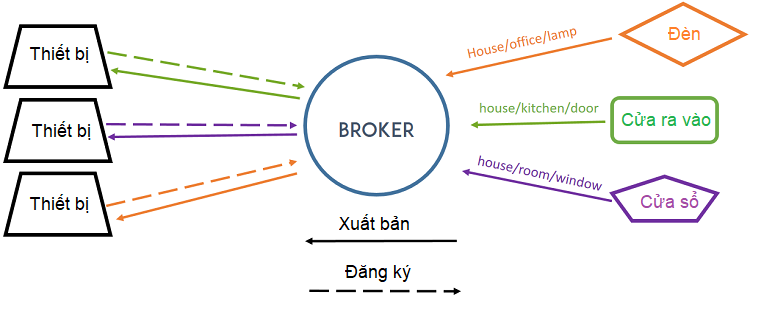
Broker của MQTT là trung tâm của mọi giao thức Xuất bản/Đăng ký. Tùy thuộc vào việc triển khai, một Broker có thể quản lý đến hàng nghìn MQTT clients kết nối đồng thời. Broker chịu trách nhiệm nhận tất cả các tin nhắn, lọc các tin nhắn, xác định ai đã đăng ký nhận từng tin nhắn và gửi tin nhắn đến các client đã đăng ký đó. Người môi giới cũng lưu trữ phiên của tất cả các clients liên tục, bao gồm các đăng ký và các tin nhắn bị lỡ. Một nhiệm vụ khác của người môi giới là xác thực và ủy quyền cho các clients.

Trách nhiệm chính của một Broker là:

* Nhận tất cả tin nhắn
* Lọc tin nhắn
* Quyết định xem khách hàng nào quan tâm.
* Xuất bản tin nhắn cho tất cả khách hàng đã đăng ký

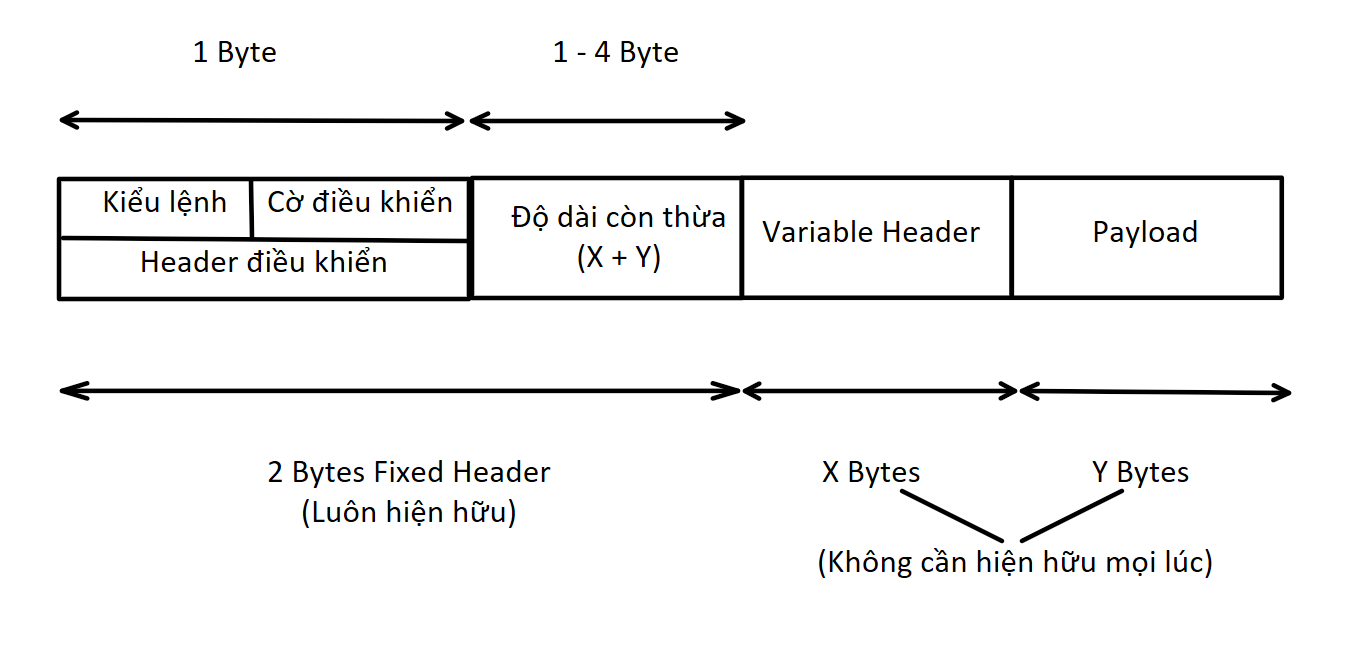
Thông thường, Broker có tính mở rộng, giúp dễ dàng tùy chỉnh xác thực, ủy quyền và tích hợp với hệ thống backend. Tích hợp là đặc biệt quan trọng, vì Broker thường là thành phần trực tiếp tiếp xúc với Internet, phục vụ nhiều clients và phải chuyển tiếp tin nhắn đến các hệ thống phân tích và xử lý phía dưới. Tóm lại, người môi giới là trung tâm mà thông qua đó mọi tin nhắn phải được định tuyến. Do đó, quan trọng rằng Broker của bạn có khả năng mở rộng cao, có thể tích hợp vào hệ thống backend, dễ dàng theo dõi và, tất nhiên, đảm bảo tính an toàn khi gặp sự cố.

Dưới đây là một ví dụ về broker, là một ví dụ mở rộng hơn việc bật/tắt đèn được nói đến ở phần chủ đề (Topics):



Thay vì chỉ đóng mở cửa sổ với một thiết bị từ xa thì thông qua broker, có thể làm thêm 2 việc nữa đó là đóng hoặc mở cửa phòng bếp hoặc cửa sổ phòng. Mỗi vận dụng riêng sẽ có thiết bị từ xa riêng phụ trách cho việc đóng hoặc mở vật dụng đó.

## 2.6 Packet Format



*(Sơ đồ Packet Format của MQTT)*

Trong một gói điều khiển MQTT, có tối đa 3 thành phần bao gồm Fixed Header, Variable Header và Payload. Gói điều khiển MQTT, ngoài việc chứa các định danh, còn chứa một số thông tin khác như tên chủ đề, ID của client, tên người dùng, v.v.

Trường Fixed Header là bắt buộc luôn luôn phải hiện hữu trong các gói điều khiển MQTT, đây là trường phân biệt các gói CONNECT, PUBLISH, SUBSCRIBE, PINGREQ, DISCONNECT... Hai trường Variable Header và Payload là hai trường tùy chọn, không cần hiện hữu mọi lúc, tùy thuộc vào gói điều khiển MQTT, hai trường này có thể có hoặc không có. Ví dụ:

- Gói CONNECT: chỉ có trường Fixed Header.

- Gói PUBACK: Bao gồm trường Fixed Header và trường Variable Header.

- Gói CONNECT: Bao gồm trường Fixed Header, Variable Header và trường Payload.

* Fixed header

Trường này có ít nhất 2 byte. Fixed Header chứa kiểu Gói điều khiển MQTT và Cờ (Flags) và Độ dài còn thừa. Trường độ dài còn thừa sẽ chỉ ra tổng số byte của Variable Header và Payload (từ 0 byte đến 256MB). Ở đây chúng ta sẽ phân tích trường Kiểu Gói điều khiển MQTT và Cờ, được sử dụng để phân biệt các gói điều khiển MQTT với nhau.

- Trong 8 bit của trường Kiểu Gói điều khiển MQTT và Cờ, 4 bit giá trị cao (MSB) đại diện cho Kiểu Lệnh, 4 bit còn lại là các bit Cờ điều khiển chứa các cờ cụ thể cho mỗi Kiểu Lệnh.

- Mỗi Gói điều khiển sẽ có một Kiểu Lệnh riêng. Ví dụ, 0 đại diện cho gói 'dự trữ', 1 là gói 'CONNECT', và cứ thế.

- Ví dụ: Kiểu Lệnh của gói CONNECT bằng 0001 và 4 bit Cờ điều khiển bằng 0000, vì vậy giá trị của trường Kiểu Gói điều khiển MQTT và Cờ sẽ là 0b00010000.

* Variable Header

Đây là một trường tùy chọn trong các gói điều khiển MQTT. Một số gói như CONNECT, PUBLISH, SUBSCRIBE, v.v. sử dụng trường Variable Header này để cung cấp thông tin bổ sung và chúng khác nhau tùy thuộc vào từng gói.

* Payload

Cuối một số gói điều khiển MQTT có thể chứa trường Payload vì trường Payload là tùy chọn và không yêu cầu xuất hiện trong tất cả các gói MQTT như trường Fixed Header. Trường này thường chứa nội dung của dữ liệu sẽ được gửi đi. Ví dụ:

- Trong gói CONNECT, Payload sẽ là ClientID, tên người dùng và mật khẩu (nếu có).

- Trong gói PUBLISH, Payload sẽ là nội dung dữ liệu sẽ được gửi đi.

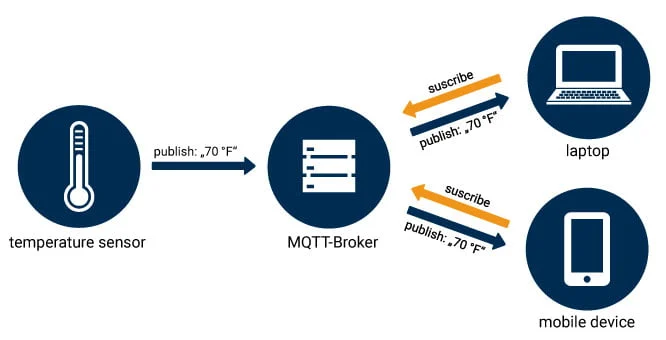
1. **Kiến trúc phần mềm**

Cấu trúc phần mềm gồm các chương trình đơn giản:

* Gateway.cpp
* Node.cpp
* LineChar.cpp
* Và các chương trình biên dịch tương ứng

Môi trường chạy chương trình: Hệ điều hành linux.

Mô hình kiến trúc:



*(Mô hình kiến trúc của chương trình)*

## 3.1 Gateway

Trong hệ thống Gateway đóng vai trò MQTT-Broker nhận nhiệm vụ trung chuyển dữ liệu giữa các Node (publisher) và LineChar (suscriber) theo các topic mà publisher và suscriber đã đăng ký.

Sau khi khởi động chương trình, các socket sẽ được giằng buộc với một cổng, gateway sẽ hoạt động lắng nghe tại cổng đó.

Khi một publisher yêu cầu kết nối thành công và đăng ký topic, hệ thống sẽ kiểm tra và tạo một luồng xử lý theo topic đăng ký.

Để quản lý hoạt động kết nối và ngắt kết nối cũng như số lượng các kết nối tối đa của các Publisher và Suscriber, Gateway sẽ hiện thị danh sách theo dõi dễ hiểu giúp người sử dụng và người quản trị có thể theo dõi cũng như kiểm soát khi xảy ra vấn đề.

## 3.2 Node

Trong hệ thống các Node đóng vai trò là publisher sau khi đăng ký topic, các Node sẽ tự động gửi dữ liệu tới Broker sau một khoảng thời gian cố định (3s hay 3 giây). Các dữ liệu nhiệt độ thu được thông qua sensor được gắn ở các phòng kho

## 3.3 Linechar

LineChar đóng vai trò là các Suscriber, sau khi kết nối và đăng ký topic thành công LineChar sẽ nhận nhiệm vụ lắng nghe dữ liệu của Node (Suscriber) gửi tới topic thông qua Gateway (MQTT-Broker) và hiển thị các dữ liệu thu được ra dưới dạng sơ đồ đường thẳng gấp khúc.

1. **Kiểm thử**

Phần kiểm thử dưới đây sẽ cho thấy demo của chương trình, đồng thời cũng có thể coi như là hướng dẫn cài đặt và cách sử dụng chương trình sao cho hiệu quả nhất.

* Clone project về máy

Để clone chương trình về máy, ta vào terminal và chạy câu lệnh sau

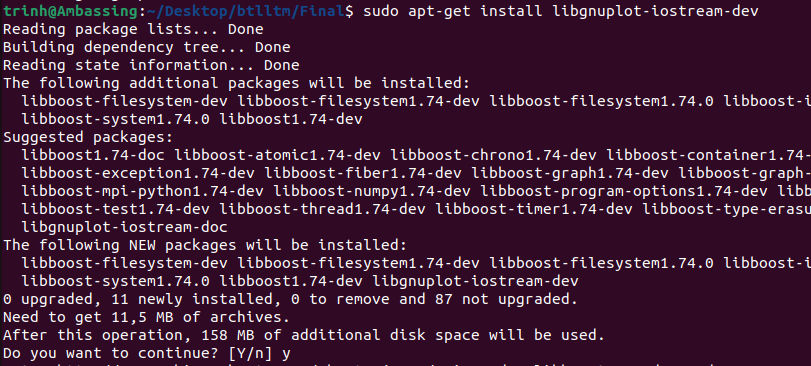
***git clone https://github.com/ngocson2g/Network-programming-MQTT.git***

* Cài đặt thư viện gnuplot-iostream

Trước tiên ta truy cập vào thư mục chứa chương trình (mặc định từ project thì thư mục có tên là Final) bằng terminal và chạy câu lệnh dưới đây:

***sudo apt-get install libgnuplot-iostream-dev***

Sau khi bấm Enter thì terminal sẽ yêu cầu chấp thuận hoặc từ chối từ phía người dùng. Để hoàn tất quá trình, nhập “y” vào terminal và bấm Enter lần nữa:



*(Terminal dùng để download thư viện gnuplot-iostream)*

* Biên dịch chương trình

Để chương trình có thể chạy thì giờ ta cần biên dịch 3 file Gateway, Node và Linechar. Ta bắt đầu với từng file một, trước tiên với File Gateway. Ta sử dụng một terminal đang ở trong file Final, sau đó chạy câu lệnh dưới đây:

***g++ -o Gateway Gateway.cpp -pthread***

Ta cũng áp dụng tương tự với file Node (Lưu ý rằng nếu bạn muốn biên dịch xong chạy các file luôn thì khuyến cáo biên dịch file trên từng terminal riêng):

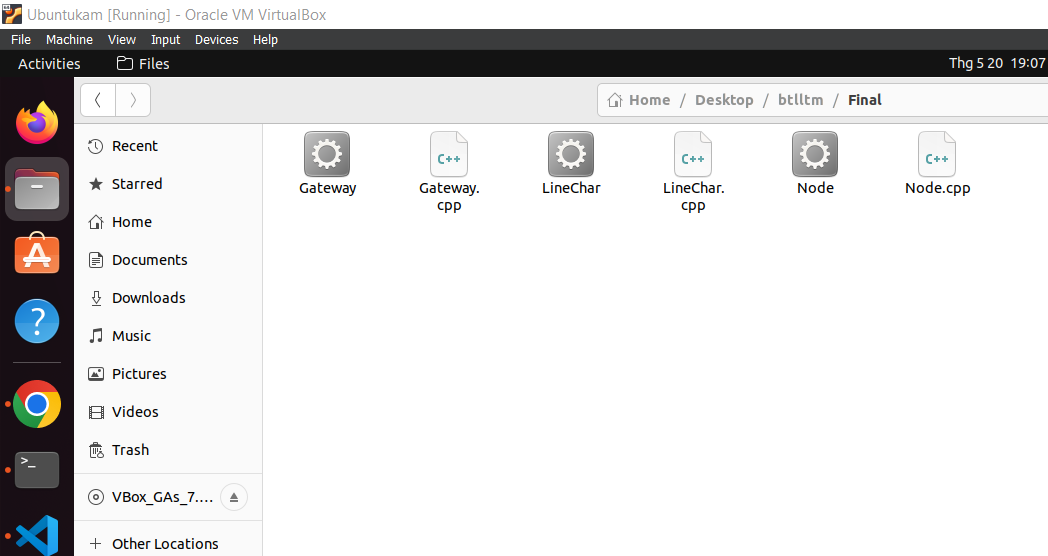
***g++ -o Node Node.cpp -pthread***

Đặc biệt với file Linechar thì sẽ có lệnh biên dịch riêng:

***g++ -o LineChar LineChar.cpp -lboost\_iostreams -pthread***

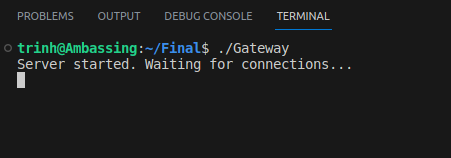
* Chạy chương trình

Sau khi biên dịch cả 3 File xong, ta có thể nhập lệnh ./(Tên file) để chạy file, hoặc chạy trình biên dịch đã được in ra. Trong Linux, trình biên dịch có dạng file hình bánh răng như dưới đây:



*(Ví dụ về dạng file đã được biên dịch)*

Giờ, ta sẽ chạy file Gateway bằng câu lệnh ***./Gateway*** trên terminal hoặc bằng trình biên dịch. Sau khi chạy, sẽ có hiển thị như sau:

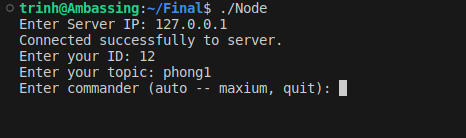


*(Màn hình hiển thị bên phía Gateway)*

Sau khi  thành công, ta quay sang chạy file Node bằng câu lệnh ***./Node*** hoặc bằng chương trình biên dịch. Sau khi chạy file, Node sẽ phải nhập địa chỉ IP sau để có thể truy cập vào Gateway: ”**127.0.0.1**”. Sau đó, chương trình sẽ yêu cầu Node nhập ID dạng số cho Node đó và tạo một Topic với tên bất kỳ. Sau khi hoàn thiện các bước trên, Node sẽ hiển thị 3 lựa chọn cho người dùng có thể nhập vào:  
 + **auto**: Node sẽ sinh dữ liệu bằng cách nhận dữ liệu từ Sensor và gửi đến Gateway. Node sẽ nhận dữ liệu 9 lần liên tiếp với delay giữa mỗi lần nhận dữ liệu là 5s. Người dùng có thể lặp lại lệnh auto nhiều lần trên 1 Node.

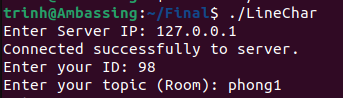
+ **maxium**: Node sẽ hiển thị tổng số ID đang có mặt trong Gateway. ID này có cả ID của Node và ID của Linechar

+ **quit**: Node sẽ ngừng kết nối tới Gateway.



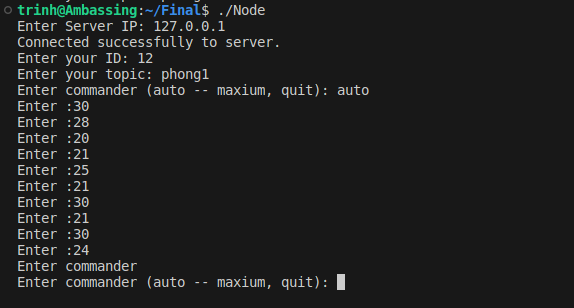
*(Hiển thị ở phía Node)*

Cuối cùng, ta chạy File Linechar bằng câu lệnh ***./Linechar*** hoặc bằng trình biên dịch được in ra. Lúc này, chương trình sẽ bắt Linechar nhập địa chỉ Ip để kết nối đến Gateway. Tương tự như Node, ta nhập địa chỉ “**127.0.0.1**” vào. Sau đó, Chương trình sẽ bắt Linechar nhập ID riêng của mình (không được phép trùng với ID của Node) và Topics hiện đang có (bắt buộc phải nhập tên đúng với một topics đang tồn tại). Lúc này, Linechar sẽ ngồi lắng nghe xem có Node nào gửi dữ liệu không:



*(Nhập thông tin cho Linechar)*

Lúc này, để sinh dữ liệu thì ta sẽ nhập lệnh “auto” vào Node đang có sẵn ở trên. Lúc này Node đó sẽ thu thập từ sensor 9 dữ liệu nhiệt độ và có delay là 5s:



*(Người dùng nhập “auto” và Node sinh dữ liệu)*

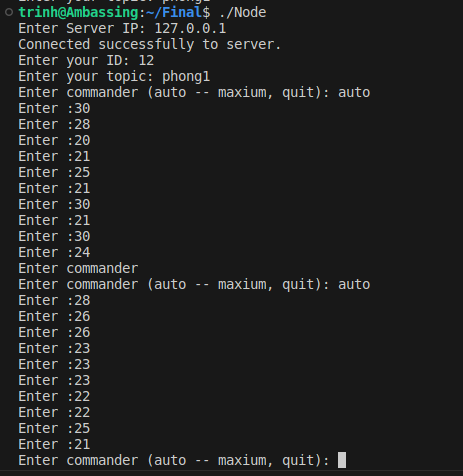
Đồng thời lúc dữ liệu được gửi đi, Linechar lắng nghe và nhận được các dữ liệu. Ngay khi nhận được dữ liệu, Linechar sẽ hiển thị dưới dạng đồ thị đường gấp khúc:

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

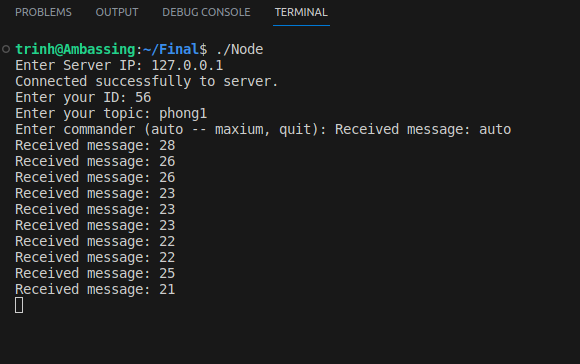
Mô tả được tạo tự động

*(Biểu đồ đường tương ứng với dữ liệu được gửi đi ở trên)*

Mỗi Linechar sẽ chỉ hiển thị dữ liệu của một Topic. Nếu có nhiều hơn 1 Node kết nối đến 1 Topic thì biểu đồ sẽ biểu thị dữ liệu của từng Node gửi một. Nếu có 1 Node khác đang gửi thì các Node còn lại cũng có khả năng lắng nghe dữ liệu của Node đang gửi dữ liệu. Vì chương trình không có giới hạn số Node nên nhiều Node có thể kết nối đến cùng 1 Topic và gửi dữ liệu đến cùng 1 Linechar đang hiển thị dữ liệu của Topic đó.



*(Node với ID là 12 thực hiện lệnh auto lần 2)*

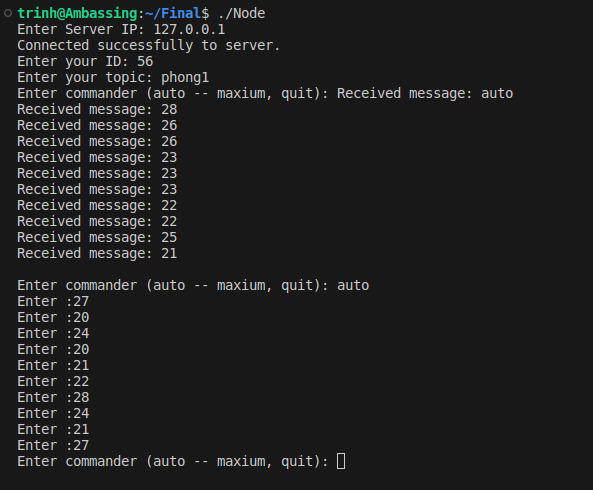


*(Một Node mới với ID là 56 đang lắng nghe dữ liệu của Node có ID là 12)*

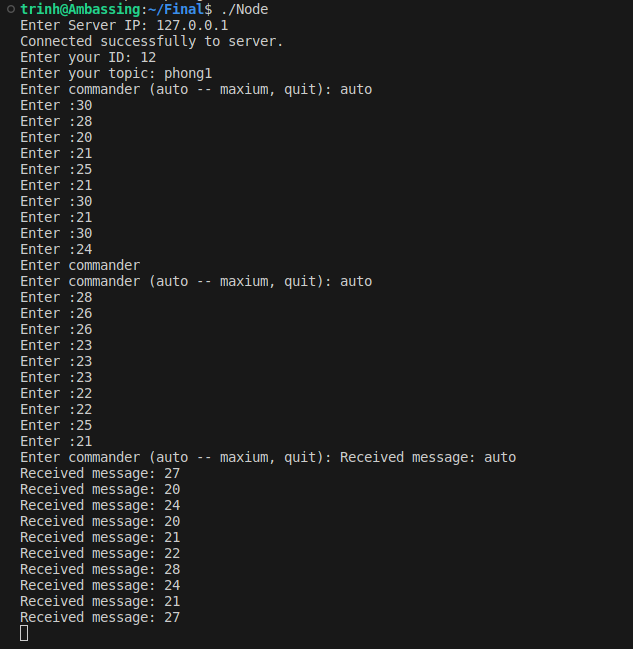
Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Mô tả được tạo tự động

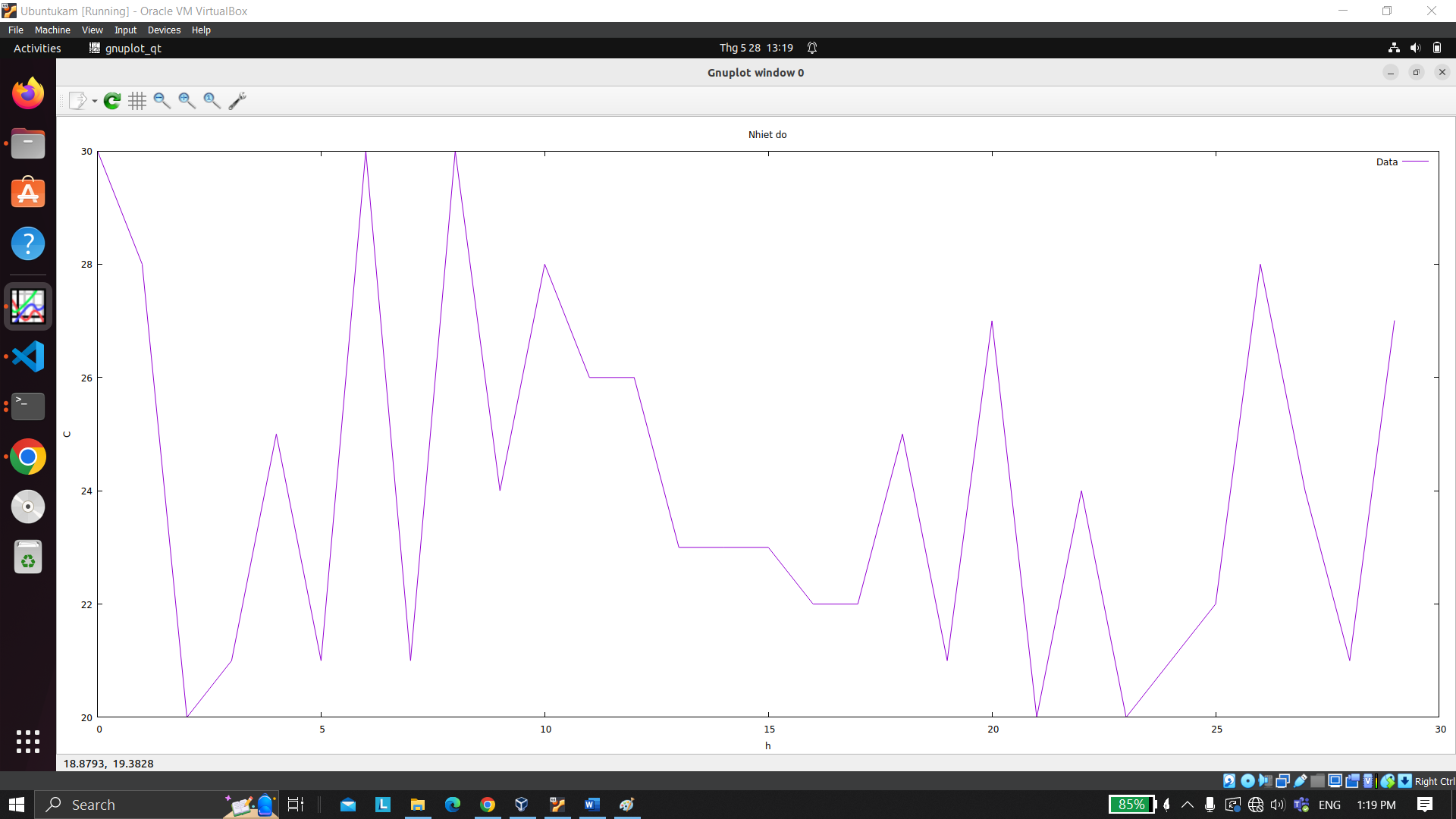
*(Biểu đồ của Linechar được tạo ở trên sau khi Node có ID là 12 chạy 2 lần lệnh “auto”)*



*(Node có ID là 56 nhập lệnh “auto” và sinh dữ liệu)*

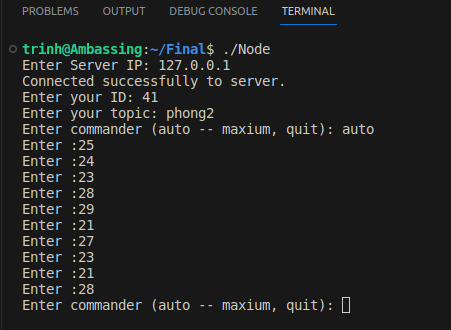


(Node có ID là 12 bắt đầu lắng nghe dữ liệu đến từ Node có ID là 56)

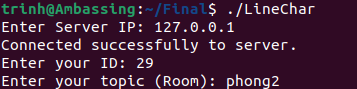


(Linechar được tạo ở trên với dữ liệu của cả 2 Node có ID là 12 và 56)

Đồng thời cũng có thể tồn tại nhiều hơn một Linechar, nhưng vì mỗi Linechar sẽ chỉ hiển thị dữ liệu của một Topic nên mỗi Linechar chỉ có thể hiển thị dữ liệu của Topic mà nó đảm nhận. Ta sẽ lấy ví dụ bằng cách tạo một Node mới có ID là 41 và từ đó tạo 1 Topics mới với tên là “phong2”. Tiếp đó, ta tạo một Linechar mới, cho ID là 29 và đảm nhận Topics “phong2”. Cuối cùng, ta cho Node đó sinh dữ liệu

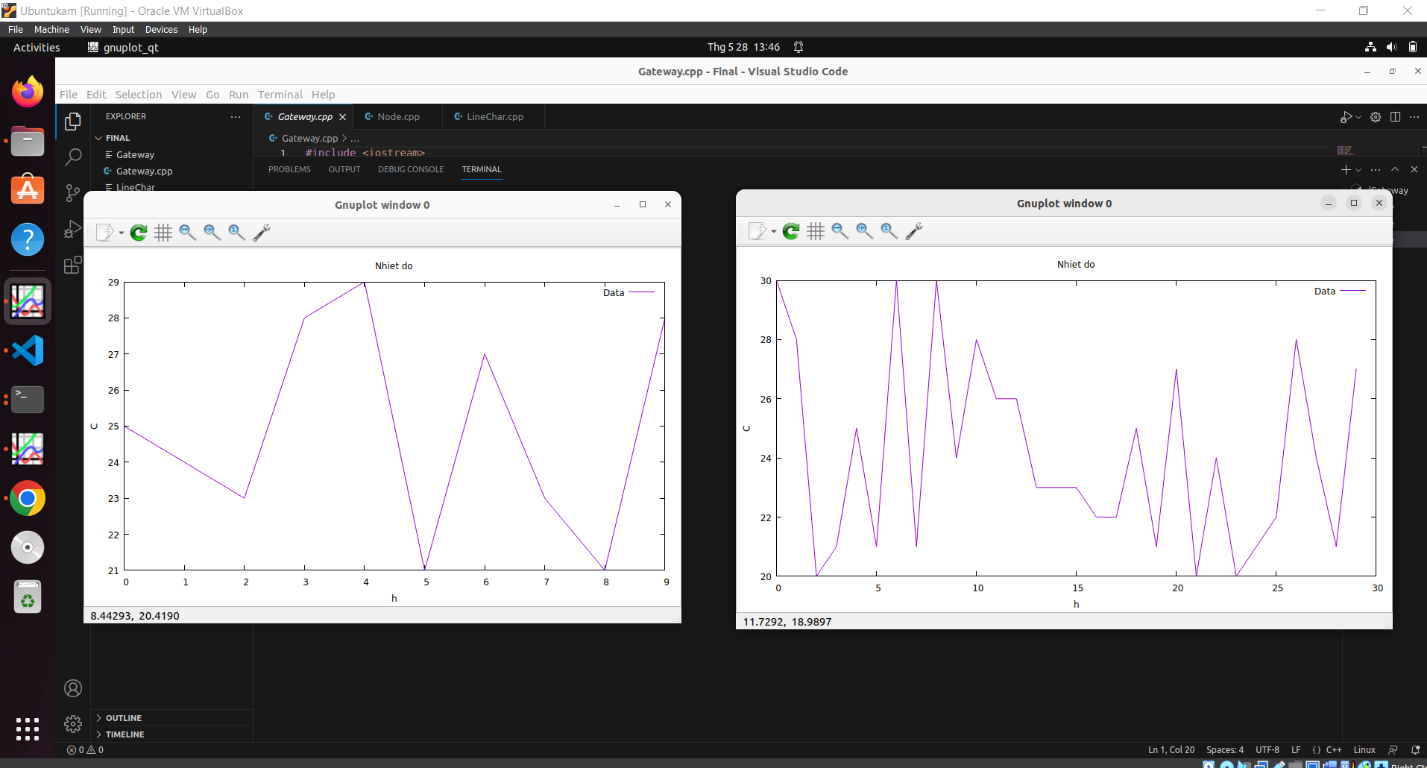


*(Node mới có ID là 41, Topics là “phong2” và sinh dữ liệu)*



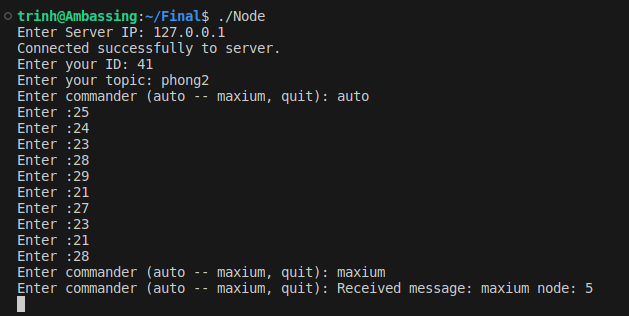
*(Tạo Linechar đảm nhận Topics “phong2”)*

Như có thể thấy ở dưới đây, Linechar đảm nhiệm Topics “phong2” (ở bên trái màn hình) và Linechar đảm nhiệm Topics “phong1” (ở bên phải màn hình) có thể cùng tồn tại mà không ảnh hưởng gì đến nhau.



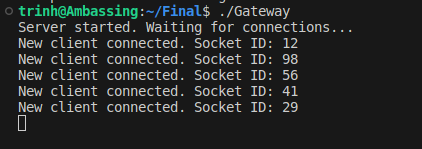
*(Nhiều Linechar tồn tại cùng 1 lúc)*

Node còn có chức năng kiểm tra xem có bao nhiêu ID đang tồn tại ở trong Gateway bằng câu lệnh “maxium”. Để lấy ví dụ, em sẽ dùng Node có ID là 41 được tạo ở trên:



*(Dùng lệnh “maxium” và hiển thị số ID có trong Gateway)*

Để kiểm tra tính xác thực, ta sẽ nhìn vào hiển thị ở bên phía Gateway:



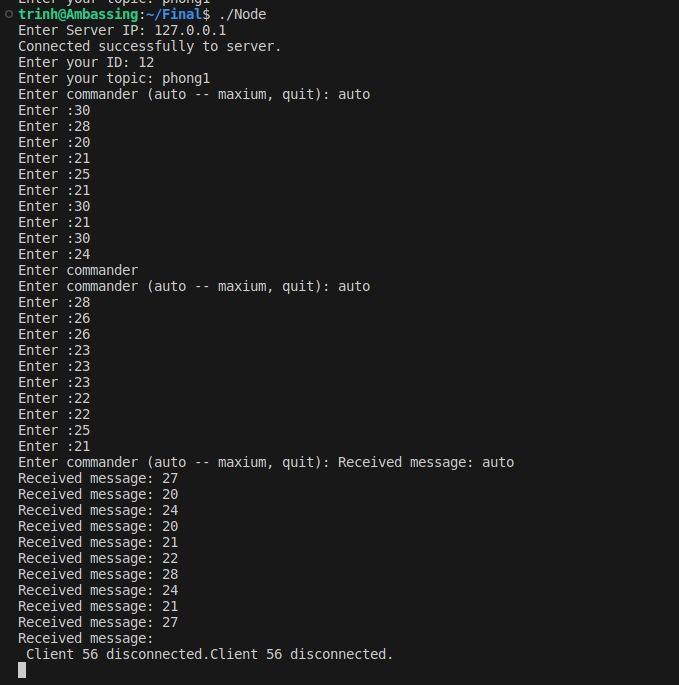
*(Hiển thị phía Gateway)*

Để một Node có thể ngưng kết nối với Gateway, ta sử dụng lệnh “quit” hoặc đơn giản là tắt terminal tương ứng. Nếu có nhiều hơn 1 Node đăng ký cùng 1 Topics thì các Node còn lại sẽ nhận được thông báo ngắt kết nối của Node vừa mới rời đi.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Mô tả được tạo tự động

*(Sử dụng lệnh “quit” để ngưng kết nối của Node có ID là 56 đến Gateway)*

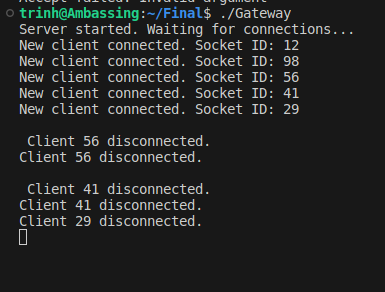


*(Node có ID là 12 đăng ký cùng Topics “phong1” nhận được thông báo)*

Qua tất cả những hoạt động đó, Gateway sẽ lưu lại địa chỉ IP của từng Node và từng Linechar. Nếu có Node hoặc Linechar mới tham gia, Gateway sẽ ra thông báo. Nếu có bất kì một Node hoặc Linechar nào ngưng kết nối đến Gateway, thì bên phía Gateway sẽ hiển thị ra thông báo sau:

***Client (ID của Node) disconnected***

Riêng nếu Node ngưng kết nốt thì sẽ được phân biệt bằng 2 lần in ra thông báo trên. Các điều về connect và disconnect vừa nói ở trên được hiển thị ở hình ảnh minh họa bên dưới:



*(Hình minh họa hiển thị ở phía Gateway)*

1. **Các thư viện được sử dụng**

## 5.1 Thư viện Gnuplot trong C++

Thư viện Gnuplot là một thư viện mã nguồn mở được viết bằng ngôn ngữ C++ để vẽ đồ thị trong chương trình C/C++. Thư viện này cho phép người dùng tạo các đồ thị 2D hoặc 3D từ dữ liệu trong chương trình của mình. Gnuplot cung cấp các chức năng cho phép người dùng tạo các loại đồ thị khác nhau, bao gồm các đồ thị đường, đồ thị điểm, đồ thị hình cầu, đồ thị mặt và nhiều hơn nữa. Nó cũng cho phép người dùng tùy chỉnh các thuộc tính của đồ thị, bao gồm màu sắc, độ dày, kiểu đường và nhiều hơn nữa.

Để sử dụng thư viện Gnuplot trong chương trình C/C++ của bạn, bạn cần tải xuống và cài đặt thư viện Gnuplot trước. Sau đó, bạn có thể bao gồmtệp đầu vào của thư viện và sử dụng các hàm số và tính năng của nó trong mã của mình.

Các bước cơ bản để sử dụng thư viện Gnuplot trong chương trình C/C++ của bạn như sau:

Bước 1: Tải và cài đặt thư viện Gnuplot trên hệ thống của bạn.

Bước 2: Bao gồm tệp đầu vào của thư viện Gnuplot trong mã của bạn. Điều này có thể được thực hiện bằng cách sử dụng lệnh #include và chỉ định đường dẫn đến tệp đầu vào.

Bước 3: Tạo một đối tượng Gnuplot để sử dụng trong mã của bạn. Điều này có thể được thực hiện bằng cách sử dụng lớp Gnuplot và khởi tạo một đối tượng mới.

Bước 4: Gửi các lệnh Gnuplot cho đối tượng Gnuplot để tạo đồ thị. Điều này có thể được thực hiện bằng cách sử dụng toán tử << và chỉ định các lệnh Gnuplot cần thiết để tạo các đồ thị.

Ví dụ, sau đây là một chương trình C++ đơn giản sử dụng thư viện Gnuplot để vẽ đồ thị hàm sin(x):

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Trong đó, chúng ta đã bao gồm tệp đầu vào của thư viện Gnuplot bằng cách sử dụng #include "gnuplot-iostream.h", sau đó tạo một đối tượng Gnuplot mới gp. Tiếp theo, chúng ta đã sử dụng toán tử << để gửi lệnh Gnuplot "plot sin(x) with lines\n" cho đối tượng gp để vẽ đồ thị hàm sin(x). Kết quả sẽ được hiển thị trên một cửa sổ đồ thị mới.

Ngoài ra, thư viện Gnuplot cũng cung cấp các chức năng để tùy chỉnh các thuộc tính của đồ thị, bao gồm màu sắc, độ dày, kiểu đường, tiêu đề, nhãn trục, vùng phạm vi và nhiều hơn nữa. Thư viện này cũng hỗ trợ nhiều định dạng tệp đầu ra khác nhau, cho phép người dùng lưu các đồ thị của mình dưới dạng ảnh hoặc các định dạng khác.

Một số các tính năng khác của thư viện Gnuplot bao gồm:

* Hỗ trợ đa nền tảng: Thư viện Gnuplot có thể chạy trên nhiều nền tảng khác nhau như Linux, Unix, macOS và Windows.
* Hỗ trợ nhiều ngôn ngữ lập trình: Thư viện Gnuplot hỗ trợ nhiều ngôn ngữ lập trình khác nhau, bao gồm C++, Python, Java và Perl.
* Hỗ trợ đồ thị 2D và 3D: Thư viện Gnuplot cho phép người dùng tạo các đồ thị 2D và 3D từ dữ liệu của họ.
* Hỗ trợ các kiểu đồ thị khác nhau: Thư viện Gnuplot cho phép người dùng tạo các loại đồ thị khác nhau, bao gồm các đồ thị đường, đồ thị điểm, đồ thị hình cầu, đồ thị mặt và nhiều hơn nữa.
* Tích hợp dễ dàng: Thư viện Gnuplot có thể được tích hợp vào các chương trình C/C++ của bạn một cách dễ dàng và nhanh chóng.
* Tài liệu phong phú: Thư viện Gnuplot có tài liệu phong phú và chi tiết, với nhiều ví dụ và hướng dẫn để giúp người dùng sử dụng và tùy chỉnh các đồ thị của mình.

## 5.2 Thư viện sys/socket.h

Thư viện sys/socket.h trong C++ là một thư viện hệ thống cho phép người dùng tạo các ứng dụng mạng trên các hệ điều hành UNIX. Thư viện này cung cấp các hàm và cấu trúc để tạo và quản lý các socket, định dạng các địa chỉ mạng và gửi/nhận dữ liệu qua mạng.

Các hàm chính trong thư viện sys/socket.h bao gồm:

* socket(): Tạo một socket mới và trả về socket descriptor.
* bind(): Liên kết một socket với một địa chỉ và cổng.
* listen(): Đặt một socket vào trạng thái lắng nghe yêu cầu kết nối từ phía client.
* accept(): Chấp nhận một yêu cầu kết nối từ phía client và tạo ra một socket mới để giao tiếp với client.
* connect(): Kết nối một socket đến một server.
* send(): Gửi dữ liệu qua mạng.
* recv(): Nhận dữ liệu từ mạng.
* close(): Đóng một socket.

Ngoài ra, thư viện sys/socket.h cũng định nghĩa các cấu trúc dữ liệu quan trọng như sockaddr và sockaddr\_in để định dạng các địa chỉ mạng.

Ví dụ sau đây minh họa cách sử dụng thư viện sys/socket.h để tạo một ứng dụng mạng đơn giản:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, phần mềm

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Trong ví dụ này, chúng ta tạo một socket đơn giản và thiết lập các tùy chọn cho socket sử dụng hàm setsockopt(). Sau đó, chúng ta liên kết socket với địa chỉ và cổng sử dụng hàm bind() và đặt socket vào trạng thái lắng nghe sử dụng hàm listen(). Khi một yêu cầu kết nối được gửi đến, chúng ta chấp nhận yêu cầu kết nối sử dụng hàm accept() và gửi phản hồi đến client.

## 5.3 Thư viện arpa/inet.h

Thư viện arpa/inet.h trong C++ là một thư viện hệ thống cho phép người dùng chuyển đổi các địa chỉ mạng giữa định dạng nhị phân và định dạng chuỗi. Thư viện này cung cấp các hàm để chuyển đổi địa chỉ IPv4 và IPv6, cũng như các hàm để chuyển đổi các cổng và địa chỉ mạng khác.

Các hàm chính trong thư viện arpa/inet.h bao gồm:

* inet\_addr(): Chuyển đổi một địa chỉ IPv4 từ định dạng chuỗi sang định dạng nhị phân.
* inet\_ntoa(): Chuyển đổi một địa chỉ IPv4 từ định dạng nhị phân sang định dạng chuỗi.
* inet\_pton(): Chuyển đổi một địa chỉ IPv4 hoặc IPv6 từ định dạng chuỗi sang định dạng nhị phân.
* inet\_ntop(): Chuyển đổi một địa chỉ IPv4 hoặc IPv6 từ định dạng nhị phân sang định dạng chuỗi.
* ntohs(): Chuyển đổi một số cổng từ định dạng mạng sang định dạng máy tính.
* htons(): Chuyển đổi một số cổng từ định dạng máy tính sang định dạng mạng.
* ntohl(): Chuyển đổi một địa chỉ IPv4 từ định dạng mạng sang định dạng máy tính.
* htonl(): Chuyển đổi một địa chỉ IPv4 từ định dạng máy tính sang định dạng mạng.

Ví dụ sau đây minh họa cách sử dụng thư viện arpa/inet.h để chuyển đổi địa chỉ IP giữa định dạng nhị phân và định dạng chuỗi:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Trong ví dụ này, chúng ta chuyển đổi một địa chỉ IPv4 từ định dạng chuỗi sang định dạng nhị phân sử dụng hàm inet\_aton(). Sau đó, chúng ta in địa chỉ IP dưới định dạng nhị phân sử dụng hàm ntohl(). Tiếp theo, chúng ta chuyển đổi địa chỉ IP từ định dạng nhị phân sang định dạng chuỗi sử dụng hàm inet\_ntop(). Cuối cùng, chúng ta in địa chỉ IP dưới định dạng chuỗi.

## 5.4 Thư viện unistd.h

Thư viện unistd.h trong C++ là một thư viện hệ thống cho phép người dùng truy cập các chức năng hệ thống như quản lý tiến trình, thực thi chương trình, truy cập tập tin và thư mục, thao tác với các tệp và đường ống, và nhiều chức năng hệ thống khác. Thư viện này thường được sử dụng trên các hệ điều hành UNIX và Linux.

Các hàm chính trong thư viện unistd.h bao gồm:

* access(): Kiểm tra quyền truy cập tệp.
* chdir(): Thay đổi thư mục làm việc hiện tại.
* close(): Đóng một tệp hoặc đường ống.
* dup(): Sao chép một đường ống hoặc tệp.
* execvp(): Thực thi một chương trình mới từ một đường dẫn tệp và các tham số.
* fork(): Tạo một tiến trình con.
* getpid(): Lấy ID của tiến trình hiện tại.
* getcwd(): Lấy đường dẫn đến thư mục làm việc hiện tại.
* pipe(): Tạo một đường ống.
* read(): Đọc dữ liệu từ một tệp hoặc đường ống.
* write(): Ghi dữ liệu vào một tệp hoặc đường ống.
* sleep(): Ngủ trong một khoảng thời gian cụ thể.
* unlink(): Xóa một tệp.
* wait(): Chờ tiến trình con kết thúc.

## 5.5 Thư viện pthread.h

Thư viện pthread.h trong C++ là một thư viện hệ thống cho phép người dùng tạo và quản lý các luồng (threads) trong chương trình. Thư viện này cung cấp các hàm để tạo, chuyển đổi, gắn kết, đồng bộ hóa và huỷ bỏ các luồng.

Các hàm chính trong thư viện pthread.h bao gồm:

* pthread\_create(): Tạo một luồng mới.
* pthread\_exit(): Kết thúc một luồng.
* pthread\_join(): Chờ một luồng kết thúc.
* pthread\_detach(): Huỷ bỏ sự gắn kết giữa một luồng và tiến trình chính.
* pthread\_mutex\_init(): Khởi tạo một mutex.
* pthread\_mutex\_lock(): Khóa một mutex.
* pthread\_mutex\_unlock(): Mở khóa một mutex.
* pthread\_cond\_init(): Khởi tạo một biến điều kiện.
* pthread\_cond\_wait(): Chờ một biến điều kiện.
* pthread\_cond\_signal(): Thông báo cho một luồng đang chờ trên một biến điều kiện.
* pthread\_cond\_broadcast(): Thông báo cho tất cả các luồng đang chờ trên một biến điều kiện.

Ví dụ sau đây minh họa cách sử dụng thư viện pthread.h để tạo và quản lý các luồng trong chương trình:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Trong ví dụ này, chúng ta sử dụng hàm pthread\_create() để tạo hai luồng mới, mỗi luồng in ra một thông điệp khác nhau. Sau đó, chúng ta sử dụng hàm pthread\_join() để chờ hai luồng kết thúc trước khi tiến trình chính kết thúc.

## 5.6 Thư viện chrono

Thư viện chrono trong C++ là một thư viện tiêu chuẩn cho phép người dùng đo thời gian và thao tác với thời gian trong chương trình. Thư viện này cung cấp các loại dữ liệu và hàm để đo thời gian, tính toán khoảng cách thời gian, và thực hiện các thao tác khác liên quan đến thời gian.

Các loại dữ liệu chính trong thư viện chrono bao gồm:

* std::chrono::duration: Đại diện cho một khoảng thời gian với đơn vị thời gian cụ thể (ví dụ: giây, mili giây, micro giây, nanosecond).
* std::chrono::time\_point: Đại diện cho một thời điểm cụ thể trong thời gian, được định nghĩa là một duration tính từ một điểm thời gian cố định (thường là Epoch).

Các hàm chính trong thư viện chrono bao gồm:

* std::chrono::duration\_cast(): Chuyển đổi một khoảng thời gian sang đơn vị thời gian khác.
* std::chrono::system\_clock::now(): Lấy thời điểm hiện tại của hệ thống.
* std::chrono::steady\_clock::now(): Lấy thời điểm hiện tại của đồng hồ ổn định (không bị ảnh hưởng bởi điều chỉnh thời gian hệ thống).
* std::chrono::time\_point\_cast(): Chuyển đổi một thời điểm sang đơn vị thời gian khác.
* std::chrono::duration::count(): Lấy số lượng đơn vị thời gian trong khoảng thời gian.

Ví dụ sau đây minh họa cách sử dụng thư viện chrono để đo thời gian trong chương trình:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Trong ví dụ này, chúng ta sử dụng std::chrono::high\_resolution\_clock::now() để lấy thời điểm hiện tại trước và sau khi thực hiện một số thao tác, sau đó sử dụng std::chrono::duration\_cast() để chuyển đổi khoảng thời gian giữa hai thời điểm sang đơn vị thời gian khác (micro giây trong trường hợp này). Cuối cùng, chúng ta sử dụng count() để lấy số lượng đơn vị thời gian trong khoảng thời gian và in ra kết quả.

## 5.7 Thư viện random

Thư viện random trong C++ là một thư viện tiêu chuẩn cho phép người dùng tạo các giá trị ngẫu nhiên trong chương trình. Thư viện này cung cấp các lớp và hàm để tạo các giá trị ngẫu nhiên với phân phối đều hoặc phân phối theo một phân phối xác định.

Các lớp chính trong thư viện random bao gồm:

* std::linear\_congruential\_engine: Lớp sinh số ngẫu nhiên sử dụng thuật toán tuyến tính kết hợp với một số nguyên tố lớn.
* std::mersenne\_twister\_engine: Lớp sinh số ngẫu nhiên sử dụng thuật toán Mersenne Twister, được coi là một trong những thuật toán sinh số ngẫu nhiên tốt nhất hiện nay.
* std::normal\_distribution: Lớp phân phối chuẩn (Gaussian) với giá trị trung bình và độ lệch chuẩn được chỉ định.
* std::uniform\_real\_distribution: Lớp phân phối đều trên khoảng giá trị cho trước.
* std::uniform\_int\_distribution: Lớp phân phối đều trên khoảng số nguyên cho trước.

Các hàm chính trong thư viện random bao gồm:

* std::random\_device: Hàm trả về một số ngẫu nhiên từ thiết bị ngẫu nhiên của hệ thống.
* std::seed\_seq: Hàm tạo một chuỗi gốc ngẫu nhiên từ một số ngẫu nhiên cho trước.
* std::shuffle: Hàm trộn các phần tử của một dãy cho trước.
* std::generate\_canonical: Hàm tạo một số ngẫu nhiên với độ chính xác cho trước.

Ví dụ sau đây minh họa cách sử dụng thư viện random để tạo các giá trị ngẫu nhiên trong chương trình:

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

Trong ví dụ này, chúng ta sử dụng std::random\_device để tạo một thiết bị ngẫu nhiên và std::mt19937 để sử dụng thuật toán Mersenne Twister với seed được lấy từ thiết bị ngẫu nhiên. Sau đó, chúng ta tạo một phân phối đều trên khoảng giá trị 1 đến 6 sử dụng std::uniform\_int\_distribution. Cuối cùng, chúng ta sử dụng dis(gen) để tạo một số ngẫu nhiên và in ra kết quả.

## 5.8 Thư viện thread

Thư viện thread trong C++ là một thư viện tiêu chuẩn cho phép người dùng tạo và quản lý các luồng (threads) trong chương trình. Thư viện này cung cấp một lớp std::thread để tạo và quản lý các luồng, cùng với các hàm để kiểm soát và đồng bộ hóa các luồng.

Các hàm chính trong thư viện thread bao gồm:

* std::thread::thread(): Tạo một luồng mới.
* std::thread::join(): Chờ một luồng kết thúc.
* std::thread::detach(): Huỷ bỏ sự gắn kết giữa một luồng và tiến trình chính.
* std::thread::get\_id(): Lấy ID của một luồng.
* std::this\_thread::yield(): Cho phép một luồng khác chạy trước.
* std::this\_thread::sleep\_for(): Ngừng một luồng trong một khoảng thời gian nhất định.
* std::mutex: Lớp khóa đơn giản để đồng bộ hóa truy cập vào các tài nguyên chung.
* std::condition\_variable: Lớp để đồng bộ hóa giữa các luồng thông qua các biến điều kiện.

Ví dụ sau đây minh họa cách sử dụng thư viện thread để tạo và quản lý các luồng trong chương trình:

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

Trong ví dụ này, chúng ta sử dụng std::thread để tạo một luồng mới để in ra 10 số. Sau đó, chúng ta thực hiện một số thao tác khác trong tiến trình chính trước khi gọi hàm join() để chờ luồng kết thúc.

*XIN CẢM ƠN THẦY VÌ ĐÃ ĐỌC!*