ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
UNIVERSITY OF SCIENCE AND TECHNOLOGY - UD

Đề tài

Hệ thống thời gian thực (RTS)

THINKING - CREATING - HUMANITY CHERISHING

TRƯỜNG ĐẠI HỌC BÁCH KHOA TỪ 1975

MỤC LỤC

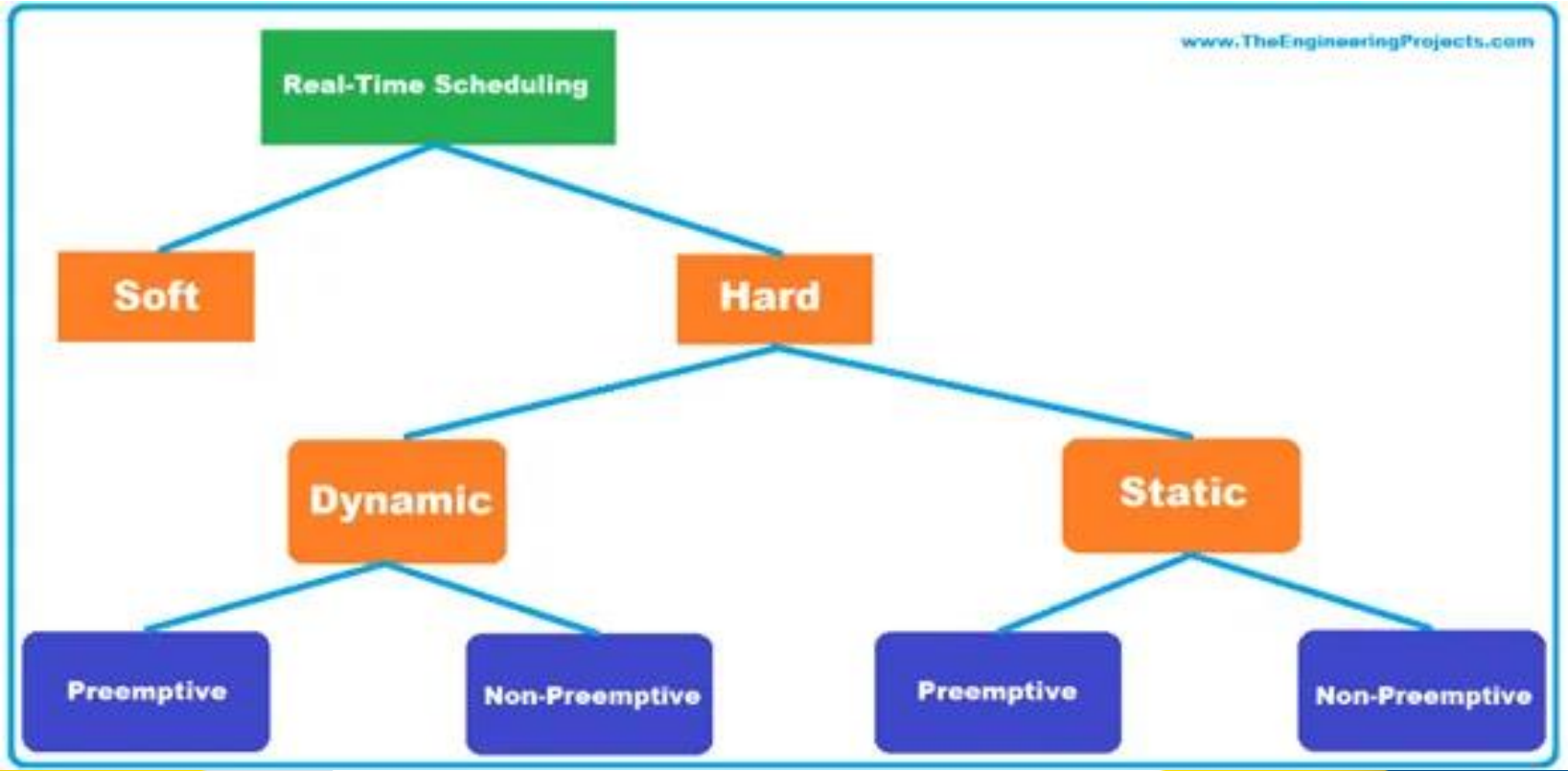
GIỚI THIỆU VỀ RTS

GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

- **Định nghĩa:** RTS yêu cầu phản hồi nhanh và chính xác theo thời gian thực.
- **Tầm quan trọng:** Được ứng dụng rộng rãi trong y tế, giao thông, công nghiệp, viễn thông...
- **Phân loại hệ thống thời gian thực**
  - **Hard Real-Time:** Phải đúng hạn, sai lệch gây hậu quả nghiêm trọng.
  - **Soft Real-Time:** Có thể sai lệch nhẹ.
  - **Firm Real-Time:** Lỗi thời gian làm giảm giá trị dữ liệu.

GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

Sự khác biệt giữa hệ thống thời gian thực cứng và mềm?



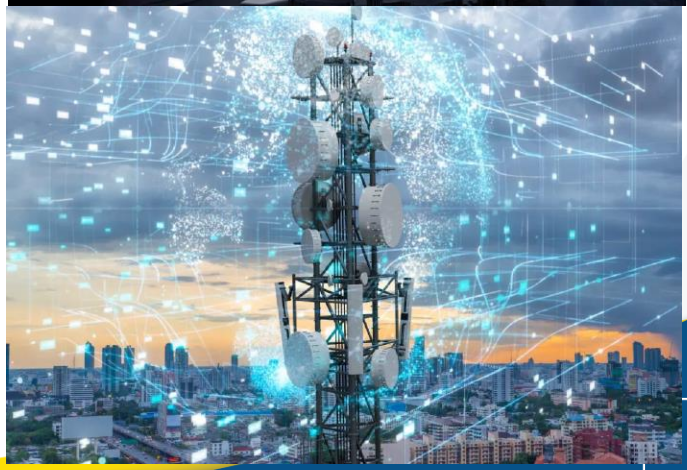
GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

Tiêu chí	Hard Real-Time System	Soft Real-Time System
File size	Kích thước tệp dữ liệu nhỏ hoặc trung bình.	Kích thước tệp dữ liệu lớn.
Response time	Thời gian phản hồi được xác định trước, thường tính bằng mili-giây.	Thời gian phản hồi cao hơn.
Database	Có cơ sở dữ liệu nhỏ.	Có cơ sở dữ liệu mở rộng.
Performance	Hiệu suất tải đỉnh phải có thể dự đoán được.	Tải đỉnh có thể được chấp nhận.
Safety	An toàn là yếu tố quan trọng.	An toàn không quá quan trọng.
Integrity	Có tính toàn vẹn dữ liệu ngắn hạn.	Có tính toàn vẹn dữ liệu dài hạn.
Flexibility & Laxity	Không linh hoạt, tuân thủ nghiêm ngặt thời hạn.	Linh hoạt hơn, có độ lỏng lẻo cao hơn và có thể chịu được một số lần trễ hạn.

GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

❖ Ứng dụng của RTS

- **Công nghiệp:** Robot, điều khiển sản xuất, quản lý kho bãi, tự động hóa.
- **Y tế:** Theo dõi bệnh nhân, hỗ trợ phẫu thuật, thiết bị y tế thông minh.
- **Viễn thông:** Chuyển mạch mạng, xử lý tín hiệu, dịch vụ truyền thông.
- **Giao thông:** Xe tự hành, điều khiển tín hiệu giao thông, quản lý luồng xe.



GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

❖ Tính năng chính của RTS

Predictability
Hệ thống có khả năng dự đoán và đảm bảo đáp ứng đúng thời gian.

Reliability
Hệ thống hoạt động ổn định, đáng tin cậy trong mọi điều kiện.

Low Latency
Độ trễ thấp, thời gian phản hồi nhanh chóng.

Scalability
Dễ dàng mở rộng quy mô, đáp ứng nhu cầu ngày càng tăng.

GIỚI THIỆU TỔNG QUAN HỆ THỐNG THỜI GIAN THỰC

❖ Phần mềm hỗ trợ RTS

- **RTOS:** VxWorks, FreeRTOS, QNX, RT-Linux.
- **Công cụ:** Simulink, LabVIEW, Eclipse, Visual Studio.
- **Ngôn ngữ lập trình:** C, Ada, Rust, C++, Java.
- **Thư viện hỗ trợ:** POSIX, Real-Time Java, HAL (Hardware Abstraction Layer).

CHI TIẾT KỸ THUẬT

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

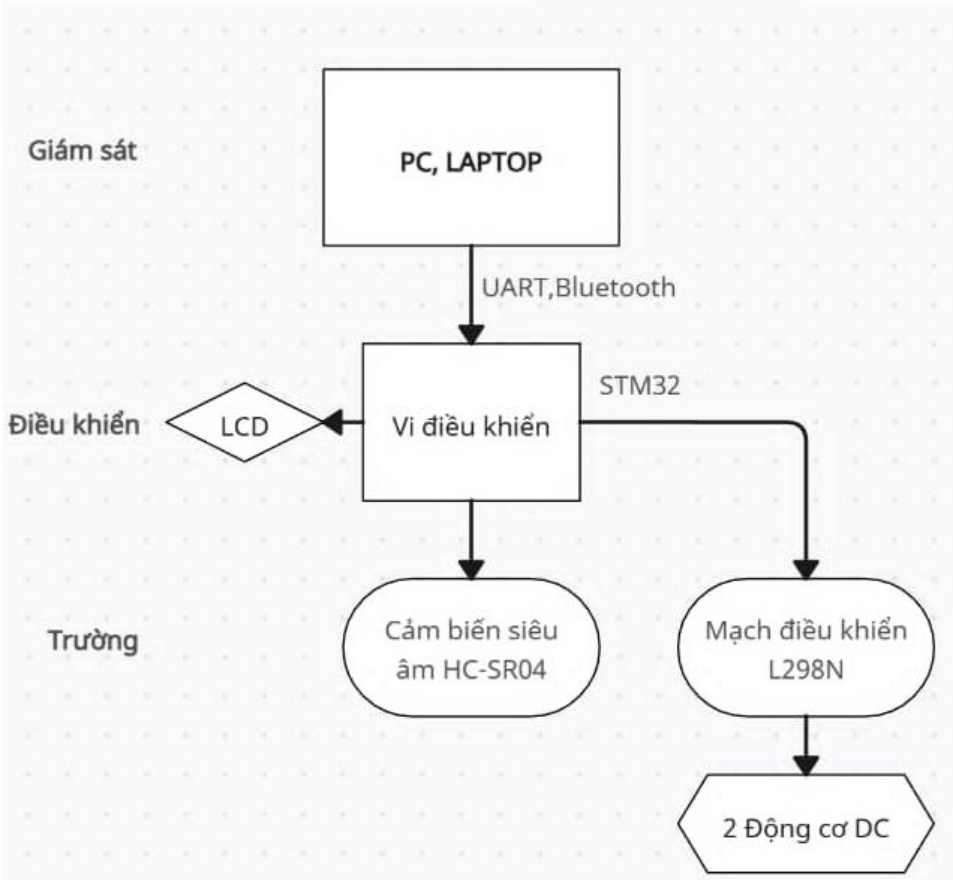
1. Mô tả mô hình

Mô hình là một robot di chuyển tự động lau nhà, có thể tránh vật cản nhờ cảm biến siêu âm/hồng ngoại. Hệ thống sử dụng RTOS để quản lý và lập lịch các nhiệm vụ như:

- Đọc dữ liệu từ cảm biến
- Điều khiển động cơ
- Gửi thông tin trạng thái về máy tính qua wifi

2. Phần cứng đề xuất

- Vi điều khiển: ESP32
- Cảm biến: Cảm biến siêu âm HC-SR04
- Động cơ: Động cơ DC hoặc Servo với mạch điều khiển L298N
- RTOS: FreeRTOS (hoặc Zephyr OS nếu muốn thử hệ điều hành mạnh hơn)

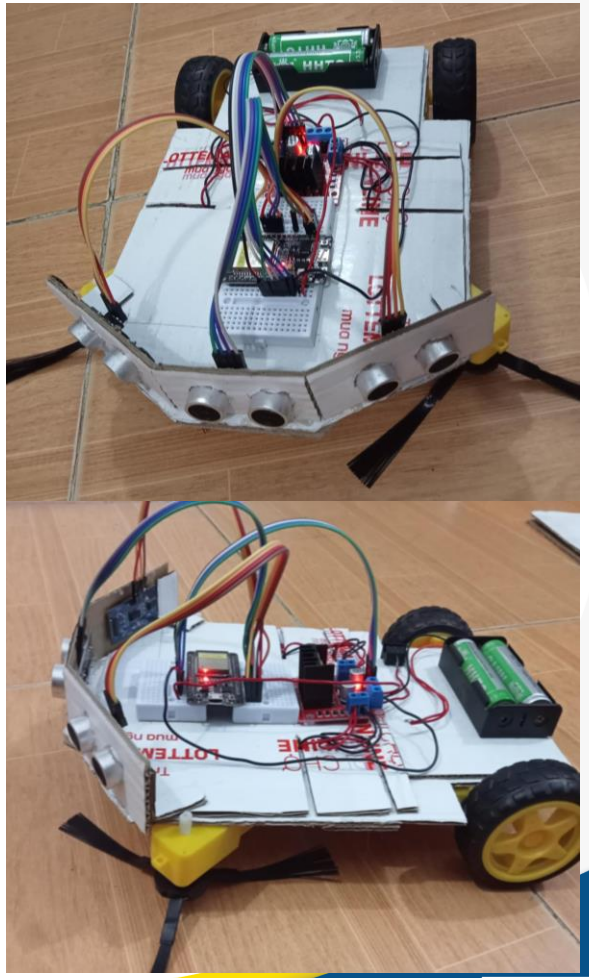


CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

3. Ứng dụng của RTOS

- ❑ RTOS sẽ giúp bạn lập lịch các tác vụ chạy đồng thời, như:
  - ➢ Task 1: Đọc dữ liệu từ cảm biến siêu âm/hồng ngoại
  - ➢ Task 2: Xử lý dữ liệu và đưa ra quyết định điều khiển
  - ➢ Task 3: Điều khiển động cơ di chuyển hoặc quay đầu khi phát hiện vật cản
  - ➢ Task 4: Gửi dữ liệu trạng thái đến máy tính qua Wifi

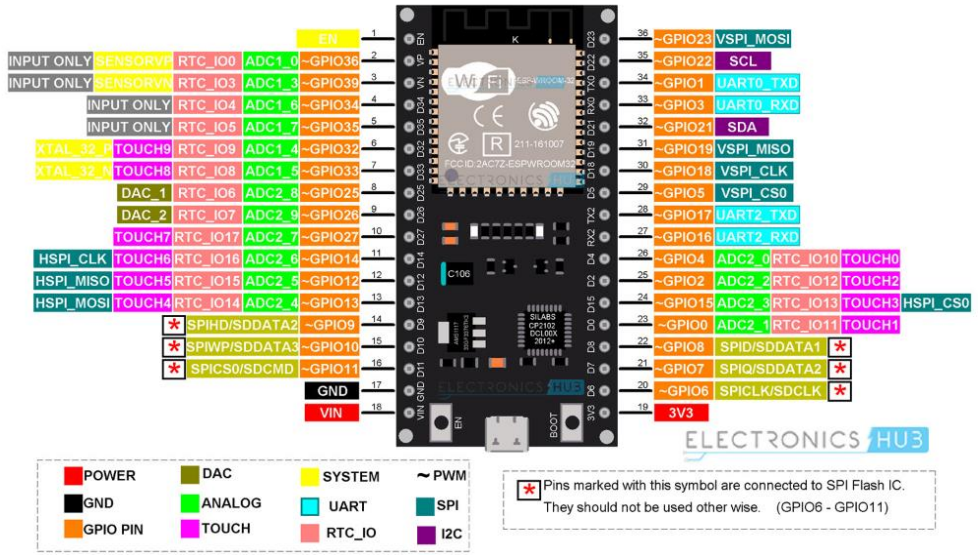


CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

I. Phần cứng/mạch (Hardware & Circuitry)

- Hệ thống thời gian thực đòi hỏi phần cứng có độ tin cậy cao và khả năng xử lý nhanh chóng.
- Các vi điều khiển (MCU) như STM32, ESP32 hoặc các bộ vi xử lý nhúng được sử dụng phổ biến.
- Mạch điện cần thiết kế tối ưu để giảm thiểu độ trễ, có các thành phần như bộ nhớ flash, RAM tốc độ cao và bus truyền dữ liệu nhanh (SPI, I2C, CAN,TCP IP, HTTP).
- FPGA (Field-Programmable Gate Array) cũng được ứng dụng để xử lý tín hiệu thời gian thực với độ trễ thấp.



CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

II. Hệ điều hành (Real-Time Operating System - RTOS)

- ❑ RTOS như FreeRTOS, VxWorks, QNX hoặc RTEMS giúp quản lý tài nguyên hệ thống hiệu quả.
- ❑ RTOS phân chia các tác vụ theo mức ưu tiên, đảm bảo tác vụ quan trọng được thực thi trước.
- ❑ Hệ điều hành có cơ chế **lập lịch thời gian thực (Real-Time Scheduling)** như:
  - ➢ **Lập lịch ưu tiên tĩnh (Static Priority Scheduling)**: Ưu tiên cố định theo mức quan trọng của tác vụ.
  - ➢ **Lập lịch ưu tiên động (Dynamic Priority Scheduling)**: Điều chỉnh mức ưu tiên theo tình trạng hệ thống.
  - ➢ RTOS hỗ trợ **task synchronization** để tránh xung đột khi nhiều tiến trình truy cập cùng một tài nguyên.

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

III. Cảm biến/linh kiện và lập lịch (Sensors, Components & Scheduling)

- ❑ Hệ thống thời gian thực thường thu thập dữ liệu từ nhiều cảm biến như cảm biến nhiệt độ, áp suất, gia tốc, LIDAR, v.v.
- ❑ Cảm biến phải có tốc độ phản hồi nhanh và kết nối ổn định qua giao thức như I2C, SPI, UART.
- ❑ **Lập lịch tác vụ (Task Scheduling)** giúp hệ thống hoạt động chính xác, tránh trễ hạn:
  - ➢ **Lập lịch theo vòng (Cyclic Scheduling)**: Các nhiệm vụ chạy theo chu kỳ cố định.
  - ➢ **EDF (Earliest Deadline First)**: Nhiệm vụ có thời hạn gần nhất được ưu tiên.
  - ➢ **Rate Monotonic Scheduling (RMS)**: Nhiệm vụ có chu kỳ ngắn hơn được ưu tiên cao hơn.

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

III. Cảm biến/linh kiện và lập lịch (Sensors, Components & Scheduling)

Task	Mô tả	Tần suất chạy
Task 1: SensorTask	Đọc dữ liệu từ cảm biến siêu âm/hồng ngoại để phát hiện vật cản	100ms
Task 2: ControlTask	Xử lý dữ liệu cảm biến và quyết định hướng di chuyển	50ms
Task 3: MotorTask	Điều khiển động cơ dựa trên quyết định từ ControlTask	50ms
Task 4: CommTask	Gửi trạng thái robot đến máy tính qua **WiFi**	500ms

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

III. Cảm biến/linh kiện và lập lịch (Sensors, Components & Scheduling)

Thời gian (ms)	SensorTask (100ms)	ControlTask (50ms)	MotorTask (50ms)	CommTask (500ms)
0	✅ (Đọc cảm biến)	✅ (Xử lý lệnh)	✅ (Điều khiển động cơ)	✅ (Gửi trạng thái)
50	❌	✅	✅	❌
100	✅ (Đọc cảm biến)	✅	✅	❌
150	❌	✅	✅	❌
200	✅ (Đọc cảm biến)	✅	✅	❌
250	❌	✅	✅	❌
300	✅ (Đọc cảm biến)	✅	✅	❌
350	❌	✅	✅	❌
400	✅ (Đọc cảm biến)	✅	✅	❌
450	❌	✅	✅	❌
500	✅ (Đọc cảm biến)	✅	✅	✅ (Gửi trạng thái)

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

Kỹ thuật yêu cầu cho hệ thông thời gian thực

Loại Yêu Cầu	Mô Tả
Yêu cầu chức năng	Mô tả hành vi hệ thống, trình tự hoạt động của từng đầu vào, bao gồm cả tình huống bình thường và bất thường.
Yêu cầu giao diện	Xác định đầu vào và đầu ra của hệ thống.
Yêu cầu hiệu suất	Chỉ định ràng buộc về thời gian và số lượng để đảm bảo đáp ứng thời gian thực.
Yêu cầu cơ sở dữ liệu logic	Xác định cấu trúc dữ liệu và cách lưu trữ.
Ràng buộc thiết kế	Liên quan đến các tiêu chuẩn phần cứng và giới hạn hệ thống.
Thuộc tính hệ thống phần mềm	Bao gồm tính khả dụng, độ tin cậy, khả năng bảo trì, v.v.
Yêu cầu phi chức năng	Định lượng & định tính, đảm bảo khả năng sử dụng, bảo trì, và độ tin cậy của hệ thống.

CHI TIẾT KỸ THUẬT

❖ Mô hình: Robot lau nhà với RTOS

Thiết kế Hệ Điều Hành Thời Gian Thực (RTOS)

Lập lịch & Đồng bộ hóa

- **Lập lịch & điều phối cục bộ:**
  - Mỗi tác vụ ứng dụng tự lập lịch dựa trên ngắt phần cứng & sự kiện hệ thống.
- **Khóa Mutex để đồng bộ hóa:**
  - Bảo vệ tài nguyên quan trọng.
  - Hệ thống quản lý yêu cầu & giải phóng Mutex qua kênh tốc độ cao.

Giao tiếp & Truy cập tài nguyên

- **Giao tiếp ưu tiên / vòng tròn:** Xử lý đồng thời yêu cầu từ ứng dụng.
- **Bộ đệm tin nhắn Task-Local:** Gửi tin nhắn qua kênh tốc độ cao dựa trên mức độ sẵn sàng.
- **Quyền truy cập tài nguyên giới hạn thời gian:** Đảm bảo phân tích thời gian thực hiện trong trường hợp xấu nhất.
- **Dịch vụ tiện ích bổ sung:** Đồng hồ thời gian thực, lịch hệ thống.

ỨNG DỤNG RTS TRONG JAVA

ỨNG DỤNG RTS TRONG JAVA

❖ Thread và cách tạo thread trong java

- **Thread** (luồng) là một đơn vị xử lý nhỏ nhất trong một chương trình, cho phép thực thi đồng thời nhiều tác vụ trong cùng một tiến trình (process). **Thread** giúp tận dụng tối đa tài nguyên CPU, đặc biệt trong các ứng dụng đa nhiệm hoặc cần xử lý song song.

- Một **thread** là một chuỗi lệnh (sequence of instructions) được thực thi độc lập trong chương trình.

- Trong Java, mọi chương trình đều có ít nhất một **thread** chính (main thread), được tạo tự động khi chương trình khởi chạy.

Kế thừa **Thread**: Tạo một lớp mới kế thừa từ lớp **Thread** và ghi đè phương thức run(). Sau đó, gọi phương thức start() để chạy Thread.

```
class MyThread extends Thread {
    public void run() {
    System.out.println(x:"Thread đang chạy...");
    }
    }
    public class Main {
    Run | Debug
    public static void main(String[] args) {
    MyThread thread = new MyThread();
    thread.start();
    }
    }
```

ỨNG DỤNG RTS TRONG JAVA

❖ Ví dụ chạy nhiều Thread

```java
public class thread extends Thread {
    private int id;

    public thread(int id) {
        this.id = id;
        this.setName("Luong " + id);
    }

    @Override
    public void run() {
        System.out.println(getName() + " bat dau xu ly...");
        try {
            Thread.sleep(millis:1000); // Giả lập xử lý trong 1 giây
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(getName() + " da xu ly xong.");
    }
```

```java
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            thread thread = new thread(i);
            thread.start();
        }
    }
}
```

```
Luong 4 bat dau xu ly...
Luong 4 bat dau xu ly...
Luong 1 bat dau xu ly...
Luong 1 bat dau xu ly...
Luong 3 bat dau xu ly...
Luong 2 bat dau xu ly...
Luong 2 bat dau xu ly...
Luong 5 bat dau xu ly...
Luong 4 da xu ly xong.
Luong 1 da xu ly xong.
Luong 2 da xu ly xong.
Luong 3 da xu ly xong.
Luong 5 da xu ly xong.
```

ỨNG DỤNG RTS TRONG JAVA

❖ Runnable và cách tạo runnable trong java

- **Runnable** là một interface (giao diện) được sử dụng để định nghĩa một tác vụ (task) mà bạn muốn thực thi trong một thread. Nó là một cách phổ biến và linh hoạt để tạo các thread trong lập trình đa luồng (multithreading)

- **Runnable** cho phép bạn định nghĩa công việc (task) riêng biệt, sau đó truyền nó vào một đối tượng Thread để thực thi.

➢ Triển khai interface **Runnable**: Tạo một lớp triển khai interface Runnable và định nghĩa phương thức run(). Truyền đối tượng này vào một đối tượng Thread và gọi start().

```
class MyRunnable implements Runnable {
    public void run() {
    System.out.println(x:"Thread đang chạy...");
    }
    }
    public class Main {
    Run | Debug
    public static void main(String[] args) {
    MyRunnable runnable = new MyRunnable();
    Thread thread = new Thread(runnable);
    thread.start();
    }
    }
```

ỨNG DỤNG RTS TRONG JAVA

❖ Ví dụ chạy nhiều Runnable

```java
public class runnable {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            int id = i; // phải là biến final hoặc effectively final để dùng trong lambda

            Runnable task = new Runnable() {
                @Override
                public void run() {
                    String threadName = Thread.currentThread().getName();
                    System.out.println(threadName + " (ID: " + id + ") bat dau xu ly...");
                    try {
                        Thread.sleep(millis:1000); // Giả lập xử lý
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println(threadName + " (ID: " + id + ") da xu ly xong.");
                }
            };

            Thread thread = new Thread(task, "Luong " + id);
            thread.start();
        }
    }
}
```

```
Luong 2 (ID: 2) bat dau xu ly...
Luong 4 (ID: 4) bat dau xu ly...
Luong 3 (ID: 3) bat dau xu ly...
Luong 5 (ID: 5) bat dau xu ly...
Luong 1 (ID: 1) bat dau xu ly...
Luong 5 (ID: 5) da xu ly xong.
Luong 2 (ID: 2) da xu ly xong.
Luong 4 (ID: 4) da xu ly xong.
Luong 3 (ID: 3) da xu ly xong.
Luong 1 (ID: 1) da xu ly xong.
```

ỨNG DỤNG RTS TRONG JAVA

❖ Đánh giá

❖ Sử dụng nhiều Thread (kế thừa Thread)

- ➢ Ưu điểm:
  - Đơn giản, dễ hiểu, phù hợp với tác vụ nhỏ và độc lập.
  - Logic luồng nằm ngay trong lớp Thread, không cần đối tượng phụ.
  - Dễ dùng các phương thức sẵn có như getName(), setPriority().
- ➢ Nhược điểm:
  - Bị giới hạn kế thừa do Java không hỗ trợ đa kế thừa.
  - Khó tái sử dụng vì logic run() gắn chặt với lớp.
  - Không dùng được trực tiếp với ExecutorService.

❖ Sử dụng nhiều Runable:

- ➢ Ưu điểm:
  - Linh hoạt: Không bị ràng buộc bởi kế thừa.
  - Tái sử dụng: Có thể truyền cho nhiều Thread hoặc ExecutorService.
  - Tách biệt rõ giữa logic xử lý và cơ chế thực thi.
  - Hỗ trợ tốt cho các công cụ đa luồng hiện đại như ExecutorService.
- ➢ Nhược điểm:
  - Mã dài hơn một chút: Cần tạo Thread để chạy Runnable.
  - Không truy cập trực tiếp các phương thức của Thread.

ỨNG DỤNG RTS TRONG JAVA

❖ Sự khác nhau giữa Thread và Runnable

Tính năng	Extending Thread	Implementing Runnable
Tính kế thừa	Không thể kế thừa lớp khác	Có thể kế thừa lớp khác
Tách biệt mã nguồn	Logic và luồng gắn chặt với nhau	Logic và luồng được tách biệt
Tính linh hoạt	Ít linh hoạt hơn	Linh hoạt hơn (tách rời logic)
Đa luồng	Tốn nhiều bộ nhớ hơn	Tiết kiệm bộ nhớ hơn

ỨNG DỤNG RTS TRONG JAVA

❖ Client Socket

❑ **Khái niệm:**

Client socket là điểm cuối (endpoint) đại diện cho phía client trong mô hình client-server. Nó thiết lập kết nối với server qua giao thức TCP/IP, cho phép gửi yêu cầu và nhận phản hồi.

❑ **Chức năng chính:**

- **Kết nối** đến server (IP + port).
- **Gửi dữ liệu** bằng OutputStream.
- **Nhận dữ liệu** bằng InputStream.

❑ **Đặc điểm:**

- **Địa chỉ & cổng:** Cần biết IP/hostname và port server (VD: localhost:5000).
- **Hai chiều:** Client chủ động kết nối, sau đó dữ liệu trao đổi hai chiều.
- **Tài nguyên:** Mỗi socket dùng tài nguyên hệ thống → cần close() sau khi dùng.

❑ **Quy trình hoạt động:**

1. Client tạo Socket với IP & cổng server.
2. Server chấp nhận → kết nối được thiết lập.
3. Dữ liệu trao đổi qua luồng (Input/Output).
4. Kết thúc → client đóng socket.

ỨNG DỤNG RTS TRONG JAVA

❖ Client Socket

```java
import java.io.*;
import java.net.*;

public class SimpleClient {
    public static void main(String[] args) {
        final String SERVER_IP = "localhost"; // hoặc IP máy chủ
        final int SERVER_PORT = 5000;

        try {
            // 1. Kết nối đến Server
            Socket socket = new Socket(SERVER_IP, SERVER_PORT);
            System.out.println("Đã kết nối đến server tại " + SERVER_IP + ":" + SERVER_PORT);

            // 2. Tạo luồng đầu ra để gửi dữ liệu
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);
            writer.println("Xin chào server!");

            // 3. Tạo luồng đầu vào để nhận dữ liệu
            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            String response = reader.readLine();
            System.out.println("Phản hồi từ server: " + response);

            // 4. Đóng kết nối
            socket.close();
            System.out.println("Đã đóng kết nối.");
        } catch (IOException e) {
            System.out.println("Lỗi: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.*;
import java.net.*;

public class SimpleServer {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        final int PORT = 5000;

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server đang lắng nghe ở cổng " + PORT);

            Socket socket = serverSocket.accept(); // Chấp nhận kết nối từ client
            System.out.println("Client đã kết nối: " + socket.getInetAddress());

            // Nhận dữ liệu từ client
            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            String clientMessage = reader.readLine();
            System.out.println("Client gửi: " + clientMessage);

            // Gửi phản hồi lại cho client
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), autoFlush:true);
            writer.println(x:"Chào bạn, client!");

            // Đóng kết nối
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ỨNG DỤNG RTS TRONG JAVA

❖ Server Socket

❑ **Khái niệm:**

- Server socket là điểm cuối phía máy chủ trong mô hình client-server. Được triển khai qua lớp java.net.ServerSocket, nó cho phép lắng nghe cổng và thiết lập kết nối TCP với các client.

❑ **Chức năng chính:**

- Lắng nghe yêu cầu từ client.
- Chấp nhận kết nối và tạo Socket mới cho từng client.
- Quản lý luồng dữ liệu hai chiều giữa server và client.

❑ **Đặc điểm:**

- **Cổng (port):** Lắng nghe trên cổng cụ thể (VD: 5000); cổng < 1024 cần quyền admin.
- **Đa kết nối:** Hỗ trợ nhiều client bằng đa luồng hoặc thread pool.
- **Tài nguyên:** Tiêu tốn tài nguyên → cần quản lý và đóng kết nối khi xong.

❑ **Quy trình hoạt động:**

1. Tạo ServerSocket gắn với cổng (VD: 5000).
2. Gọi accept() để chờ client kết nối.
3. Khi có kết nối, trả về Socket để trao đổi dữ liệu.
4. Server xử lý và phản hồi.
5. Đóng kết nối sau khi hoàn tất.

ỨNG DỤNG RTS TRONG JAVA

❖ Server Socket

```java
import java.io.*;
import java.net.*;

public class MultiThreadedServer {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        final int PORT = 5000;

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("🚀 Server đang lắng nghe tại cổng " + PORT);

            while (true) {
                Socket socket = serverSocket.accept(); // Chờ client kết nối
                System.out.println("✅ Kết nối mới từ " + socket.getInetAddress());

                // Tạo luồng riêng xử lý client
                new ClientHandler(socket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```java
import java.io.*;
import java.net.*;

public class ClientHandler extends Thread {
    private Socket socket;

    public ClientHandler(Socket socket) {
        this.socket = socket;
    }

    public void run() {
        try {
            // Nhận dữ liệu từ client
            BufferedReader reader = new BufferedReader(
                new InputStreamReader(socket.getInputStream())
            );
            String message = reader.readLine();
            System.out.println("📩 Client gửi: " + message);

            // Gửi phản hồi
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), autoFlush:true);
            writer.println("📢 Server nhận: " + message);

            // Đóng kết nối
            socket.close();
            System.out.println(x:"🔚 Kết thúc kết nối với client.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ỨNG DỤNG RTS TRONG JAVA

❖ Đánh giá

❑ Server:

- ➢ ServerSocket lắng nghe trên cổng 5000 và chấp nhận kết nối từ client.
- ➢ Mỗi khi có client kết nối, một thread mới được tạo từ ClientHandler để xử lý độc lập.
- ➢ while (true) đảm bảo server luôn sẵn sàng nhận kết nối mới.

❑ ClientHandler (Runnable):

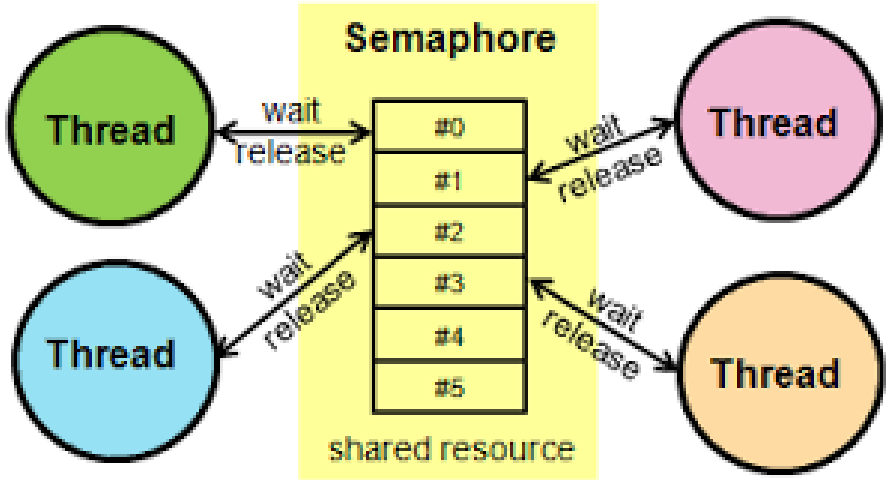
- ➢ Là một lớp triển khai Runnable, xử lý giao tiếp với từng client.
- ➢ Đọc tin nhắn từ client, gửi phản hồi, rồi đóng kết nối khi hoàn tất.

ỨNG DỤNG RTS TRONG JAVA

❖ Semaphore và synchronize

❑ **Semaphore**:

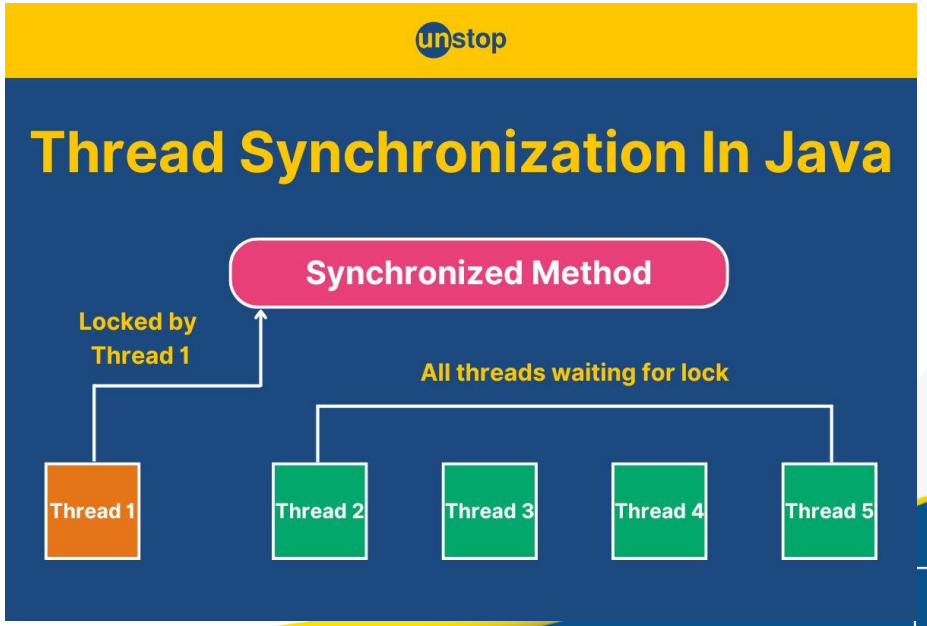
- Là một cơ chế đồng bộ hóa trong hệ điều hành, dùng để quản lý truy cập tài nguyên chung giữa các tiến trình/luồng.
- Là một biến nguyên (integer) với hai thao tác chính:
  - **Wait (P)**: Giảm giá trị semaphore, nếu < 0 thì luồng bị chặn.
  - **Signal (V)**: Tăng giá trị semaphore, đánh thức luồng đang chờ nếu có.
- Có 2 loại:
  - *Counting Semaphore*: Đếm số lượng tài nguyên.
  - *Binary Semaphore*: Tương đương khóa mutex (0 hoặc 1).



ỨNG DỤNG RTS TRONG JAVA

❖ Semaphore và synchronize

- **Synchronize (đồng bộ hóa)**:
  - Kỹ thuật đảm bảo **chỉ một tiến trình/luồng** được truy cập vào vùng tài nguyên quan trọng (critical section) tại một thời điểm. Ngăn **race condition** khi nhiều luồng truy cập dữ liệu chung.
- **Trong HĐH**:
  - Bảo vệ tài nguyên (RAM, file, thiết bị)
  - Cơ chế: Mutex, Semaphore...
- **Trong Java**:
  - Dùng từ khóa synchronized
  - Áp dụng cho phương thức hoặc khối mã
- **Ứng dụng**:
  - Cập nhật biến toàn cục, danh sách dùng chung
  - Điều khiển robot, tránh va chạm luồng



ỨNG DỤNG RTS TRONG JAVA

❖ Semaphore và synchronize

Ví dụ bài toán áp dụng Semaphore

```java
import java.util.concurrent.Semaphore;

public class SemaphoreExample {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        Semaphore sem = new Semaphore(permits:2); // Giới hạn 2 luồng truy cập

        Runnable task = () -> {
            String threadName = Thread.currentThread().getName();
            System.out.println(threadName + " dang *cho* quyen truy cap...");

            try {
                sem.acquire(); // Xin quyền truy cập
                System.out.println(threadName + " duoc cap quyen -> dang xu ly...");
                Thread.sleep(millis:1000); // Giả lập xử lý
            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                System.out.println(threadName + " da xong, tra quyen.");
                sem.release(); // Trả quyền
            }
        };

        for (int i = 0; i < 5; i++) {
            new Thread(task, "Luong " + i).start();
        }
    }
}
```

```
Luong 0 dang *cho* quyen truy cap...
Luong 1 dang *cho* quyen truy cap...
Luong 3 dang *cho* quyen truy cap...
Luong 2 dang *cho* quyen truy cap...
Luong 1 duoc cap quyen -> dang xu ly...
Luong 0 duoc cap quyen -> dang xu ly...
Luong 4 dang *cho* quyen truy cap...
Luong 1 da xong, tra quyen.
Luong 0 da xong, tra quyen.
Luong 3 duoc cap quyen -> dang xu ly...
Luong 2 duoc cap quyen -> dang xu ly...
Luong 2 da xong, tra quyen.
Luong 3 da xong, tra quyen.
Luong 4 duoc cap quyen -> dang xu ly...
Luong 4 da xong, tra quyen.
```

✓ Kết quả: Mỗi lần chỉ 2 luồng được xử lý, các luồng khác đợi.

ỨNG DỤNG RTS TRONG JAVA

❖ Semaphore và synchronize

Các phần cần đồng bộ hóa hoặc Semaphore:

Thành phần	Kỹ thuật sử dụng	Mục đích
Gửi dữ liệu từ sensor client → server	synchronized (out)	Tránh ghi đè khi nhiều sensor gửi cùng lúc
Task ControlTask chờ tín hiệu từ client	ReentrantLock + Condition.await/signal	Điều khiển đúng thời điểm
Task MotorTask chờ tín hiệu từ ControlTask	Tương tự trên	Thực thi sau khi xử lý xong
Biến chia sẻ sensorValues, motorCommand, mode	ConcurrentHashMap, AtomicInteger	Đảm bảo thread-safe
Giao tiếp mạng client-server	Socket độc lập mỗi client	Tránh xung đột dữ liệu mạng

ỨNG DỤNG RTS TRONG JAVA

❖ Semaphore và synchronize

Khi nào cần Semaphore/ Synchronize ?

- ❑ **Semaphore**:
  - ➢ Quản lý tài nguyên giới hạn (kết nối DB, ghế phòng).
  - ➢ Điều phối luồng (Producer-Consumer, hàng đợi).
- ❑ **Synchronize**:
  - ➢ Bảo vệ dữ liệu chung (biến, danh sách).
  - ➢ Ngăn race condition (giao dịch ngân hàng, log).
- ❑ **Bài tập nhóm**:
  - ➢ Semaphore: Mô phỏng tài nguyên giới hạn.
  - ➢ Synchronize: Đảm bảo tính toàn vẹn dữ liệu.

MÔ PHỎNG

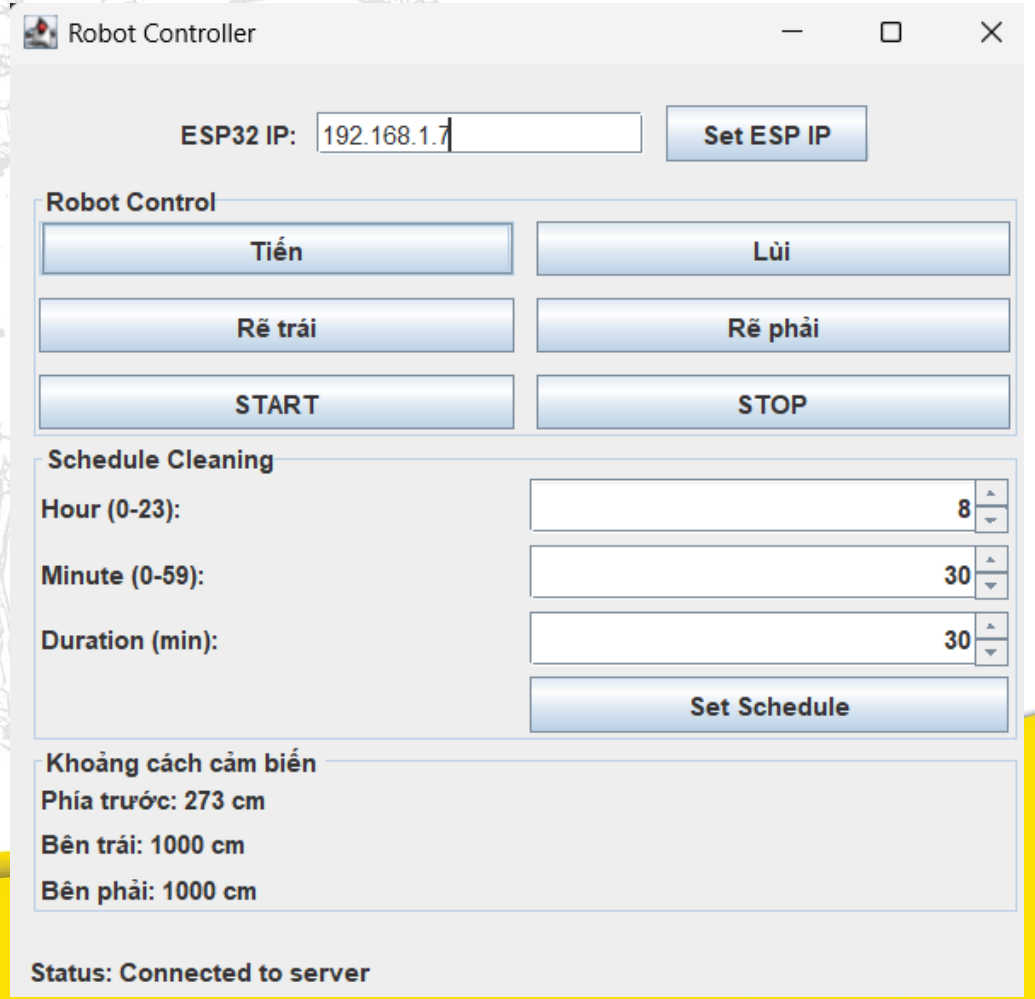
❖ SỬ DỤNG JAVA ĐỂ LẬP TRÌNH HỆ THỐNG THỜI GIAN THỰC

❑ Hệ thống thời gian thực:

- ➢ Đa luồng: Server xử lý nhiều client và ESP32 đồng thời.
- ➢ Socket: Giao tiếp giữa Client GUI, Server, và ESP32 qua socket.
- ➢ Semaphore: Giới hạn số client và lệnh đồng thời.
- ➢ Synchronize: Đồng bộ hóa truy cập dữ liệu và gửi lệnh.

❑ Điều khiển robot:

- ➢ Các nút trên Java Client GUI hoạt động giống web server, điều khiển robot thành công.



❖ SỬ DỤNG JAVA ĐỂ LẬP TRÌNH HỆ THỐNG THỜI GIAN THỰC

- Server có thể nhận dữ liệu từ ESP32 và xử lý nhiều yêu cầu từ client cùng lúc, tăng hiệu suất thời gian thực.
- Server giao tiếp với client (cổng 8888) và ESP32 (cổng 8889) thông qua socket, đảm bảo truyền dữ liệu thời gian thực giữa các thành phần.
- Ngăn chặn quá tải server bằng cách giới hạn số client, đảm bảo hiệu suất ổn định trong hệ thống thời gian thực.
- Tránh xung đột dữ liệu khi nhiều luồng truy cập cùng lúc, đảm bảo tính toàn vẹn dữ liệu trong hệ thống thời gian thực.

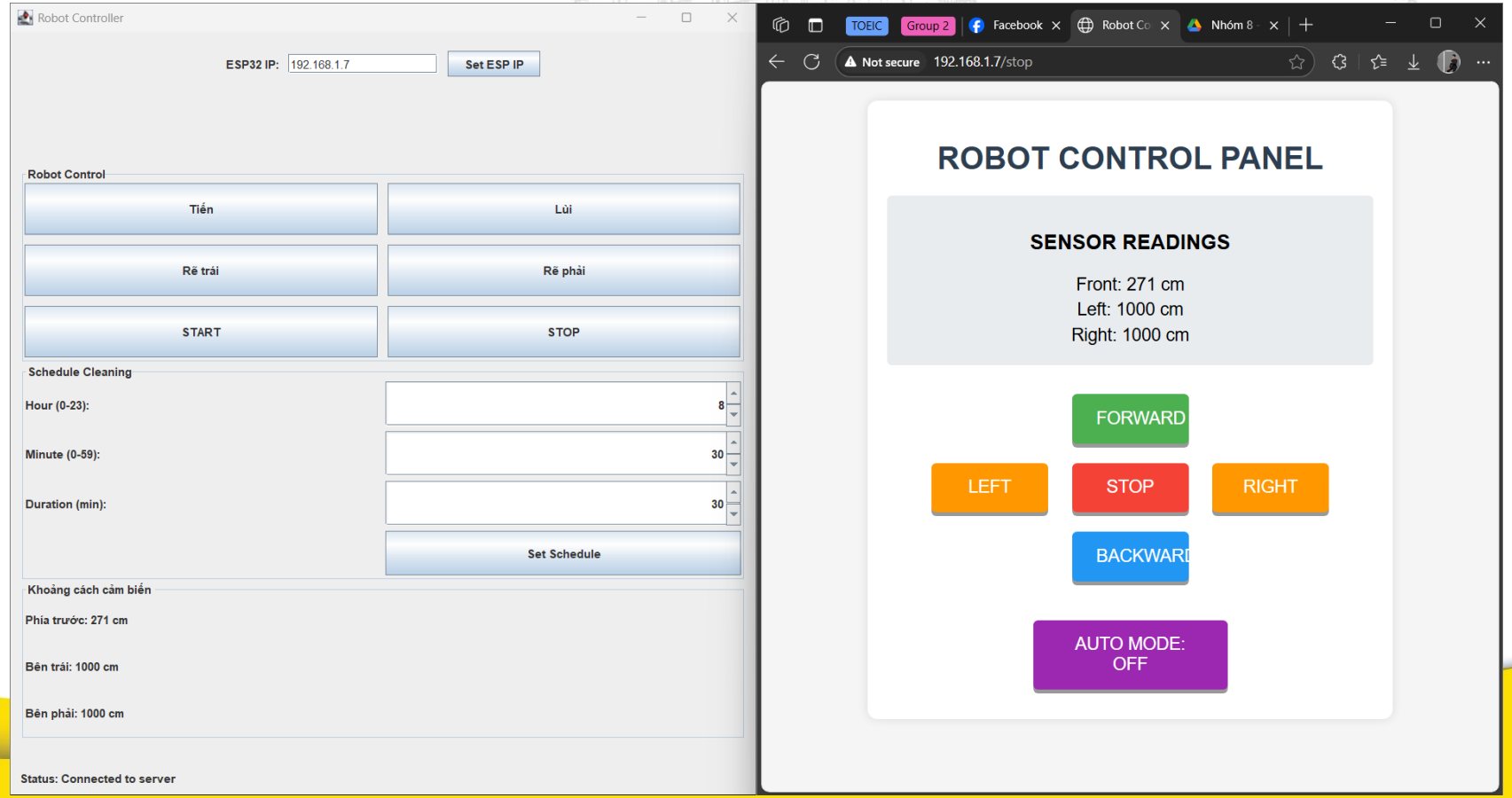
```java
// Luồng nhận dữ liệu từ ESP32
new Thread(() -> {
    try {
        ServerSocket espServerSocket = new ServerSocket(8889);
        System.out.println("Waiting for ESP32 data on port 8889...");

        while (true) {
            Socket espSocket = espServerSocket.accept();
            BufferedReader espIn = new BufferedReader(new InputStreamReader(espSocket.getInputStream()));
            String espData;
            while ((espData = espIn.readLine()) != null) {
                if (espData.startsWith("SENSOR")) {
                    System.out.println("ESP32 Data: " + espData);
                    synchronized (Server.class) { // Đồng bộ hóa truy cập latestSensorData
                        latestSensorData = espData;
                    }
                }
            }
            espSocket.close();
        }
    } catch (IOException e) {
        System.err.println("Error receiving ESP32 data: " + e.getMessage());
    }
}).start();

// Xử lý client
while (true) {
    Socket clientSocket = serverSocket.accept();
    if (clientSemaphore.tryAcquire()) {
        System.out.println("Client connected: " + clientSocket.getInetAddress());
        new Thread(() -> handleClient(clientSocket)).start();
```

❖ SỬ DỤNG JAVA ĐỂ LẬP TRÌNH HỆ THỐNG THỜI GIAN THỰC

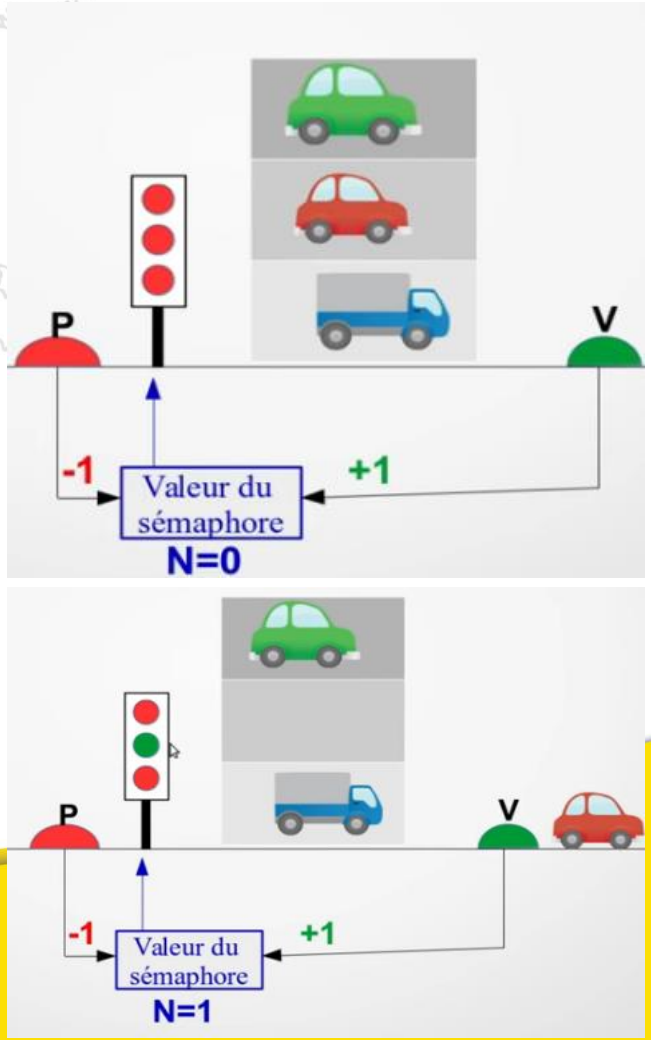
➢ So sánh synchronize đồng bộ dữ liệu và gửi lệnh điều khiển giữa webserver và java



❖ SỬ DỤNG JAVA ĐỂ LẬP TRÌNH HỆ THỐNG THỜI GIAN THỰC

❑ **Kết quả khi áp dụng: Multithread, Socket, Semaphore, Synchronize vào bài thực mô hình thực tế**

- **Hiệu quả tổng thể**: Các cơ chế trên kết hợp tạo ra một hệ thống thời gian thực ổn định, với khả năng xử lý đồng thời, giao tiếp mạng hiệu quả, quản lý tài nguyên hợp lý, và bảo vệ dữ liệu khỏi xung đột.
- **Ưu điểm**: Hệ thống phản hồi nhanh, hỗ trợ nhiều client và cập nhật dữ liệu cảm biến liên tục.
- **Điểm cần cải thiện**: Có thể tối ưu bằng cách sử dụng thread pool thay vì tạo thread mới cho mỗi client, hoặc thêm cơ chế retry cho socket khi kết nối thất bại.



TRÂN TRỌNG
CẢM ƠN

MỜI CÁC BẠN ĐẶT CÂU HỎI