RACKBOX SOFTWARE

Versão 1.0 **06/mar/2025**

Índice

Especificação do software:

1 - O fan foi acionado e devolveu sinal que está funcionando.
2 - O fan foi acionado e o sinal de funcionamento não apareceu, situação crítica, pois a cpu está aquecendo acima da temperatura normal de trabalho, o rust poderia enviar um email via gmail.
3 - O fan foi delisgado e o sinal de funcionamento do fan enviou sinal que desligou
4 - O fan foi desligado e o sinal de funcionamento continua dizendo que o fan está rodando, situação crítica pois os fans para rpi4 que tenho não devem trabalhar 100% do tempo, nesse caso também cabe enviar um email via gmail para um email configurado.
5 - O sinal de fan funcionando é via gpio e valor lógico 1 para funcionando valor lógico 0(zero) para não funcionando.

1. Requisitos da Versão 2:

- Entrada GPIO:

Um pino GPIO para ler o sinal de feedback do fan (1 = funcionando, 0 = não funcionando).

- Envio de E-mails:

Usar a crate lettre
para enviar e-mails via Gmail em situações críticas.

- Lógica de Controle:

1. Verificar se o fan está funcionando após ser ligado.
2. Verificar se o fan parou após ser desligado.
3. Enviar e-mails em situações críticas (fan não ligou ou não desligou).

2. Código Rust para a Versão 2:

Código completo:

rust

```
use rppal::gpio::Gpio;
 use std::fs;
 use std::thread;
 use std::time::Duration;
 use syslog::{Facility, Formatter3164};
 use lettre::{Message, SmtpTransport, Transport};
 use lettre::transport::smtp::authentication::Credentials;

 // Constantes de configuração
 const GPIO_FAN_PIN: u8 = 17; // Pino GPIO para controlar o fan
 const GPIO_FAN_FEEDBACK_PIN: u8 = 27; // Pino GPIO para ler o feedback do fan
 const TEMP_MIN: f32 = 40.0;  // Temperatura mínima para desligar o fan
 const TEMP_MAX: f32 = 50.0;  // Temperatura máxima para ligar o fan
 const POLL_INTERVAL: u64 = 10; // Intervalo de verificação da temperatura em
segundos
 const FAN_CHECK_DELAY: u64 = 2; // Tempo para verificar o feedback do fan
após ligar/desligar

 // Configurações do Gmail
 const GMAIL_USER: &str = "seuemail@gmail.com";
 const GMAIL_PASSWORD: &str = "suasenha";
 const ALERT_EMAIL: &str = "destinatario@example.com";

 fn get_cpu_temperature() -> f32 {
     // Lê a temperatura da CPU do arquivo do sistema
     let temp_str =
fs::read_to_string("/sys/class/thermal/thermal_zone0/temp")
         .expect("Falha ao ler a temperatura da CPU");
     let temp_millicelsius: f32 = temp_str.trim().parse().expect("Temperatura
inválida");
```

```
        temp_millicelsius / 1000.0
    }

    fn log_to_syslog(message: &str) {
        // Configura o logger para syslog
        let formatter = Formatter3164 {
            facility: Facility::LOG_USER,
            hostname: None,
            process: "rackbox-fancontroller".into(),
            pid: 0,
        };

        match syslog::unix(formatter) {
            Ok(mut logger) => {
                logger.err(message).expect("Falha ao escrever no syslog");
            }
            Err(e) => eprintln!("Falha ao configurar o syslog: {}", e),
        }
    }

    fn send_email(subject: &str, body: &str) {
        let email = Message::builder()
            .from(GMAIL_USER.parse().unwrap())
            .to(ALERT_EMAIL.parse().unwrap())
            .subject(subject)
            .body(body.to_string())
            .unwrap();

        let creds = Credentials::new(GMAIL_USER.to_string(),
GMAIL_PASSWORD.to_string());

        let mailer = SmtpTransport::relay("smtp.gmail.com")
            .unwrap()
            .credentials(creds)
            .build();

        match mailer.send(&email) {
            Ok(_) => log_to_syslog(&format!("E-mail enviado: {}", subject)),
            Err(e) => log_to_syslog(&format!("Falha ao enviar e-mail: {}", e)),
        }
    }

    fn main() {
        let gpio = Gpio::new().expect("Falha ao inicializar o GPIO");
        let mut fan_pin = gpio.get(GPIO_FAN_PIN).unwrap().into_output();
        let fan_feedback_pin =
gpio.get(GPIO_FAN_FEEDBACK_PIN).unwrap().into_input();

        log_to_syslog("Serviço de controle do fan iniciado.");

        loop {
            let temp = get_cpu_temperature();
            println!("Temperatura da CPU: {:.1}°C", temp);

            if temp >= TEMP_MAX && !fan_pin.is_set_high() {
                fan_pin.set_high(); // Liga o fan
                log_to_syslog(&format!("Fan ligado. Temperatura: {:.1}°C",
temp));

                // Verifica se o fan está funcionando após ser ligado
                thread::sleep(Duration::from_secs(FAN_CHECK_DELAY));
                if fan_feedback_pin.is_low() {
                    log_to_syslog("Erro crítico: Fan não ligou.");
                    send_email(
                        "Erro Crítico: Fan não ligou",
                        &format!("Temperatura da CPU: {:.1}°C. O fan não ligou.",
temp),
                    );
                }
            } else if temp <= TEMP_MIN && fan_pin.is_set_high() {
```

```
                fan_pin.set_low(); // Desliga o fan
                log_to_syslog(&format!("Fan desligado. Temperatura: {:.1}°C",
temp));

                // Verifica se o fan parou após ser desligado
                thread::sleep(Duration::from_secs(FAN_CHECK_DELAY));
                if fan_feedback_pin.is_high() {
                    log_to_syslog("Erro crítico: Fan não desligou.");
                    send_email(
                        "Erro Crítico: Fan não desligou",
                        &format!("Temperatura da CPU: {:.1}°C. O fan não
desligou.", temp),
                    );
                }
            }

            thread::sleep(Duration::from_secs(POLL_INTERVAL));
        }
    }
```

3. Explicação do Código:

- Novas Constantes:

- `GPIO_FAN_FEEDBACK_PIN`: Pino GPIO para ler o sinal de feedback do fan.
- `FAN_CHECK_DELAY`: Tempo para verificar o feedback do fan após ligar/desligar.
- `GMAIL_USER`, `GMAIL_PASSWORD`, `ALERT_EMAIL`: Configurações do Gmail.

- Funções Adicionais:
  - `send_email`: Envia e-mails via Gmail usando a crate lettre.
  - Verificação do feedback do fan após ligar/desligar.

- Lógica de Controle:
  1. Liga o fan se a temperatura atingir `TEMP_MAX` e verifica se o feedback é `1`.
  2. Desliga o fan se a temperatura cair para `TEMP_MIN` e verifica se o feedback é `0`.
  3. Envia e-mails em situações críticas (fan não ligou ou não desligou).

4. Dependências:

Adicione as crates lettre e syslog ao `Cargo.toml`:
toml

```
[dependencies]
 rppal = "0.14"
 syslog = "5.0.0"
 lettre = "0.10"
```

5. Compilação e Execução:

Para compilar e executar o código:
bash
cargo build --release
sudo ./target/release/rackbox-fancontroller

6. Exemplo de Log no Syslog e E-mails:

- Syslog:

Oct 10 12:34:56 raspberrypi rackbox-fancontroller: Serviço de controle do fan iniciado.
Oct 10 12:35:10 raspberrypi rackbox-fancontroller: Fan ligado. Temperatura: 50.5°C
Oct 10 12:35:12 raspberrypi rackbox-fancontroller: Erro crítico: Fan não ligou.
Oct 10 12:35:50 raspberrypi rackbox-fancontroller: Fan desligado. Temperatura: 39.8°C
Oct 10 12:35:52 raspberrypi rackbox-fancontroller: Erro crítico: Fan não desligou.

- E-mails:

- Assunto: "Erro Crítico: Fan não ligou"
- Corpo: "Temperatura da CPU: 50.5°C. O fan não ligou."

7. Considerações Finais:

- Segurança do Gmail:

Certifique-se de usar uma senha de aplicativo no Gmail (não use sua senha principal).

- Testes:

Teste o código em cada Raspberry Pi 4 para garantir que o feedback do fan e o envio de e-mails funcionem corretamente.

- Escalabilidade:

O código é modular e pode ser adaptado para mais fans ou situações complexas.