https://doi.org/10.47734/regrasp.vxxx.xxxx

INSTITUTO FEDERAL
São Paulo
Campus São Paulo
REGRASP

IMPLEMENTAÇÃO DE UM SISTEMA OPERACIONAL EM TEMPO REAL EM SISTEMA EMBARCADO PARA CONTROLE DE TEMPERATURA

IMPLEMENTATION OF A REAL-TIME OPERATING SYSTEM IN AN EMBEDDED SYSTEM FOR TEMPERATURE CONTROL

Andrew Ral Sen Wong
Graduando em Engenharia de Controle e Automação
IFSP/ Campus São Paulo
andrew.wong@aluno.ifsp.edu.br

Arthur Augusto de Lima Maximiano
Graduando em Engenharia de Controle e Automação
IFSP/ Campus São Paulo
arthur.maximiano@aluno.ifsp.edu.br

Rodrigo Rech
Mestre em Controle e Automação
IFSP/ Campus São Paulo
rodrigo.rech@ifsp.edu.br

ARTIGO INFO. Recebido: 10.05.2025 Aprovado: 11.10.2025 Disponibilizado: 22.10.2025

RESUMO
Este trabalho apresenta uma aplicação de Sistema Operacional de Tempo Real (RTOS / Real Time Operating System) em um sistema embarcado de controle de temperatura. Durante o desenvolvimento, foram analisadas a programação de um RTOS em comparação a uma programação convencional de embarcados (Bare Metal), a sua complexidade no desenvolvimento, a aplicação da gestão dos recursos das tarefas em tempo real e a sua sincronização, além de avaliar os benefícios da utilização do RTOS, como a simplificação do gerenciamento de recursos, determinação das tarefas mais prioritárias até o desempenho do controle térmico da aplicação proposta. Os resultados demonstraram a eficiência da aplicação de um RTOS num sistema embarcado de controle de temperatura, utilizando como base o FreeRTOS em um microcontrolador ARM (STM32F401), disponibilizando uma base para apoiar o desenvolvimento de novos projetos embarcados.
Palavras-chave: Controle de temperatura; Microcontrolador; PID; RTOS.

ABSTRACT
This work presents an application of Real Time Operating System (RTOS) in a embedded system of temperature control, throughout the implementation, it has been analyzed how to program a RTOS in comparison to a standard methods (bare metal) in embedded system, the complexity on the development, the application of resource management for the tasks in real time and synchronization. Besides, it has been evaluated the benefits of using a RTOS, like the simplification of the resource management, determining which task has the highest priority or the performance on the temperature control of the proposed application. The results show the efficiency of the RTOS in an embedded system for temperature control, using the FreeRTOS as base for development in a microcontroller ARM (STM32F401), making it available as a base of development for new projects in the embedded systems.
Keywords: Temperature Control; Microcontroller; PID; RTOS.

INTRODUÇÃO

Com a demanda crescente sobre a Internet das Coisas (IoT), como nas áreas de instrumentação, controle de processos, dentre outros em uma escala global, traz conjuntamente desafios significativos e na complexidade no desenvolvimento de hardware e firmware, já que necessitam ser capazes de ser integrados em um sistema maior.

Para isso, os microcontroladores e microprocessadores têm tido um papel muito importante nesse assunto, onde combinados com hardware e software, obtém-se os sistemas embarcados, que, segundo Bertoleti (2019), esses sistemas são capazes de atender a aplicações específicas, com um baixo custo, eficiência energética, robustez e a compacidade, comparados a um sistema computacional genérico.

No entanto, para conseguir essa robustez e eficiência energética, não requer somente do seu hardware, mas também do seu software, para isso, métodos de programação não convencionais podem ser extremamente úteis nessas aplicações, como o Sistema Operacional de Tempo Real (RTOS). Segundo Leroux (2005), um RTOS possui um ambiente multitarefa que facilita o gerenciamento de recursos internos para o processamento de partes críticas, por meio da determinação de quais tarefas devem-se priorizar, garantindo um tempo de resposta rápida.

Com isso, o objetivo deste trabalho é a aplicação de um Sistema Operacional de Tempo Real (RTOS) embarcado no microcontrolador, ao invés de utilizar um sistema genérico, visando a gerar referência e materiais para futuros desenvolvimentos de projetos na área de sistemas embarcados que necessitem um gerenciamento dos recursos internos.

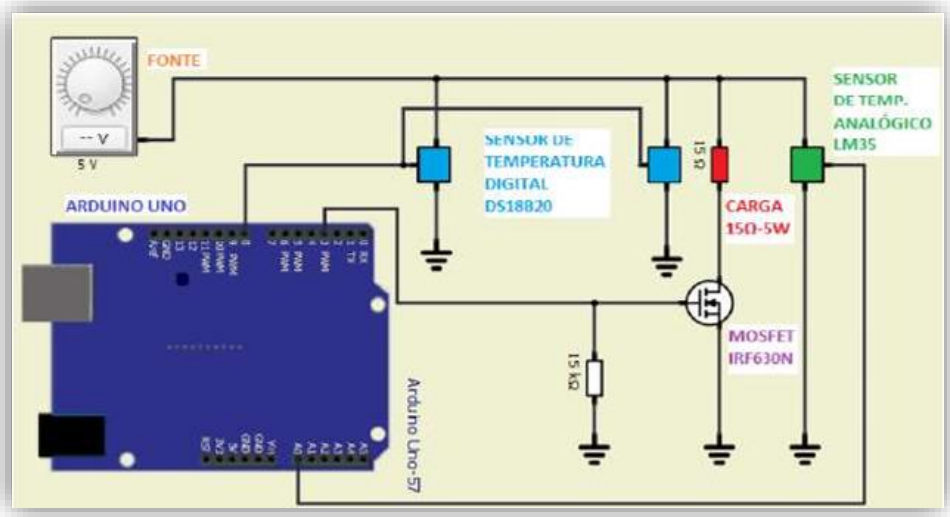
O projeto visa a utilização do RTOS num sistema embarcado, para isso foi pesquisada a arquitetura ARM (Advanced RISC Machine) do microcontrolador STM32F401, visando entender os seus periféricos ADC (Analog to Digital Converter / Conversor Análógico Digital) e PWM (Pulse Width Modulation / Modulação por Largura de Pulso) que são essenciais para o controle da planta. Em conjunto, foram estudadas as suas programações, utilizando o STM32CubeIDE como o meio de desenvolvimento, e o RTOS através da biblioteca RTOS já incluída no Middleware do STM32CubeIDE, no qual é possível utilizar a programação do FreeRTOS.

MATERIAIS E MÉTODOS

Essa pesquisa se baseou na proposta de Bastos (2021), conforme a Figura 01, sendo ele um controle de temperatura por meio de uma resistência com um sensor térmico como realimentação. Com isso foi montado um circuito conforme a figura 02 utilizando:

- Um resistor PTH (Pin Through Hole) de 15Ω/2W;
- Um LM35 como sensor de temperatura;
- Um MOSFET para controlar a carga;
- Um microcontrolador STM32F401 para receber os dados do sensor, processar e realizar o controle da carga por meio do acionamento do MOSFET.

Figura 01. Circuito do Projeto

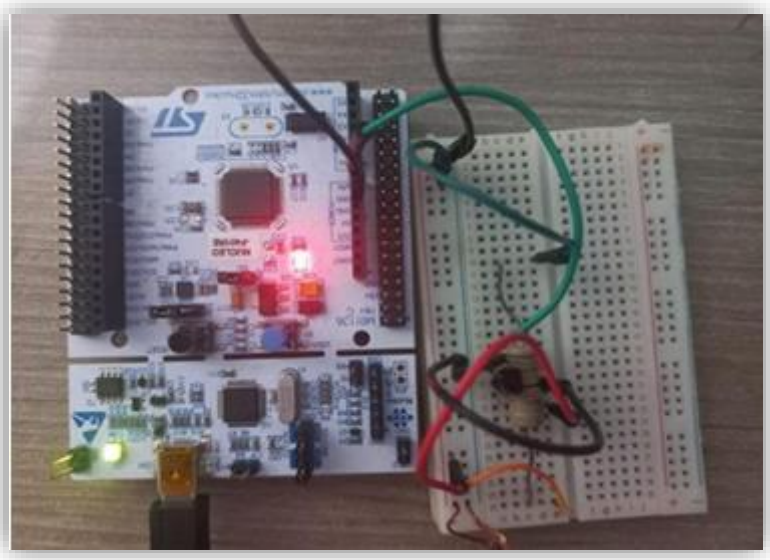


Fonte: BASTOS (2021).

Segundo Cardoso (2020), o microcontrolador é o componente mais importante para aplicações embarcadas, pois no ele é um circuito integrado que possui todos os componentes necessários de um sistema microprocessado, tais como: a CPU (Central Processing Unit), memória e periféricos, todos em um único pacote ou chip. Com isso, são computadores extremamente compactos capazes de realizar tarefas com alta eficiência energética e baixo custo.

No projeto de Bastos foi utilizado um Arduino, no entanto nesse escopo do projeto foi utilizado um STM32, que utiliza a arquitetura ARM, essa arquitetura realiza melhorias na questão da confiabilidade e desempenho (Schneider e Smalley. 2024)

Figura 02. Montagem do protótipo



Fonte: Autores

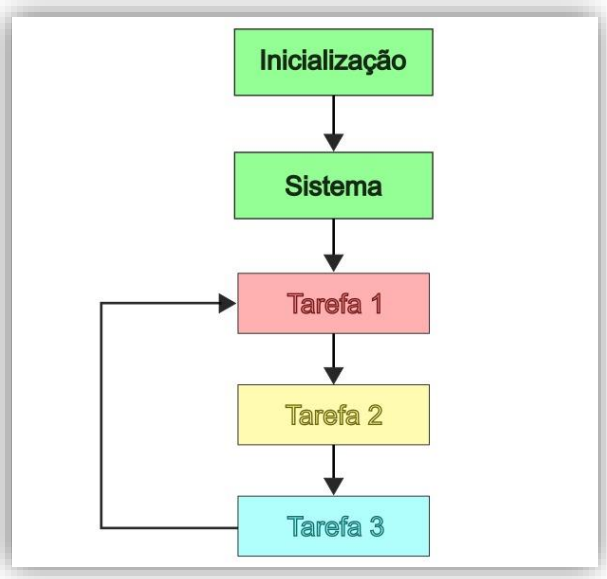
Ao ligar o sistema, o microcontrolador realiza o acionamento do MOSFET para permitir a passagem de corrente elétrica na carga resistiva, que consequentemente irá esquentar devido ao efeito joule. Ao mesmo tempo, o sensor LM35 fornecerá uma tensão elétrica proporcional à temperatura medida em seu pino de saída, que estará conectada ao microcontrolador. Este

valor de tensão será convertido em um sinal digital e é utilizado para ajustar a saída PWM do acionamento do resistor para alcançar a temperatura desejada.
Para o controle do sistema térmico, foi criado um algoritmo de PID (Proporcional, Integral e Derivativo) através do RTOS, pois esse sistema operacional possui o benefício de simplificação de certas partes da programação com uma resposta mais rápida e robusta.

Visão geral do RTOS

O sistema RTOS possui uma aplicação diferente aos dos GPOS (General Purpose Operating System), onde Leroux (2005) comenta que uma das principais vantagens é a execução de tarefas críticas de um sistema, realizando num tempo pré-definido pelo programador ao ser acionado.
Os GPOS (normalmente utilizados em computadores) geralmente realizam tarefas no plano de fundo (Background), o que pode fazer com que o sistema seja mais lento, mas essas tarefas são feitas para uma melhor interação com o usuário e ser capaz de atender aos requerimentos (Tpointtech, 2025).
Enquanto em sistemas embarcados, que utilizam a programação “Superloop” ou “Bare metal”, permitem uma programação simplificada, possui problemas semelhantes ao GPOS (Vaibhav, 2024), em que uma tarefa crítica para o funcionamento correto poderá esperar o processamento de tarefas menos críticas, possibilitando falhas no sistema implementado devido a lentidão da resposta conforme mostra a Figura 03 (Zhao, 2025).

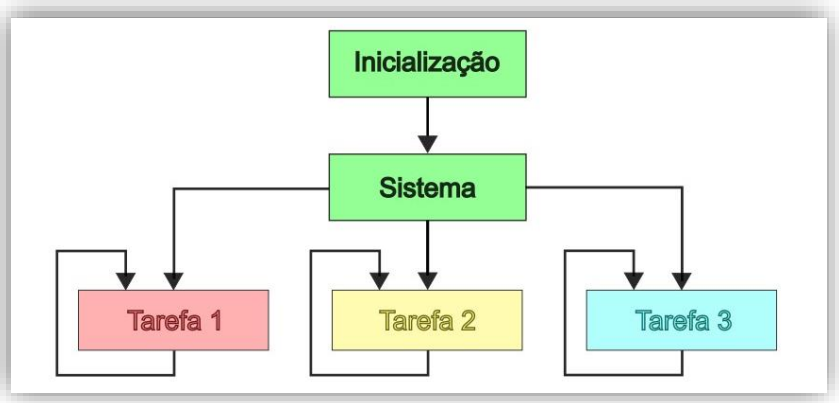
Figura 03. Funcionamento de um “Bare metal”



Fonte: Autores.

O RTOS, que funciona conforme a Figura 04, é capaz de executar as tarefas com maior prioridade (críticas) no menor tempo possível, além de ser de forma geral mais leve, podendo ser usado em sistemas menores (sistemas que realizam funções específicas) ou embarcados, em que a sua especialização é a sua velocidade, redução do seu tempo de inatividade, robustez contra falhas e melhor eficiência energética.

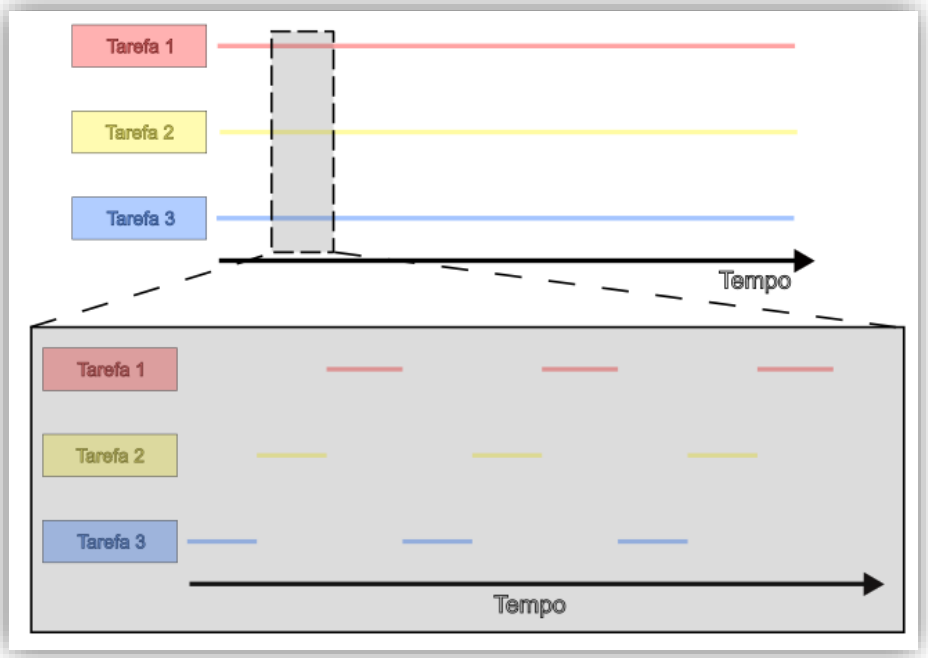
Figura 04. Funcionamento de um RTOS



Fonte: Autores.

O sistema RTOS é programado, segundo Walls (2021), de uma forma que há múltiplas tarefas ocorrendo ao mesmo tempo, com a distinção entre eles sendo a sua prioridade, assim esse sistema é capaz de alocar recursos de forma mais inteligente.
Embora o RTOS aparenta realizar múltiplas tarefas simultâneas num microcontrolador, na realidade as tarefas estão sendo realizadas de forma sequencial, de acordo com a prioridade das tarefas e o seu tempo requerido, como ilustrado na Figura 05 (Walls, 2021).

Figura 05. Funcionamento do Hardware para as tarefas



Fonte: Autores.

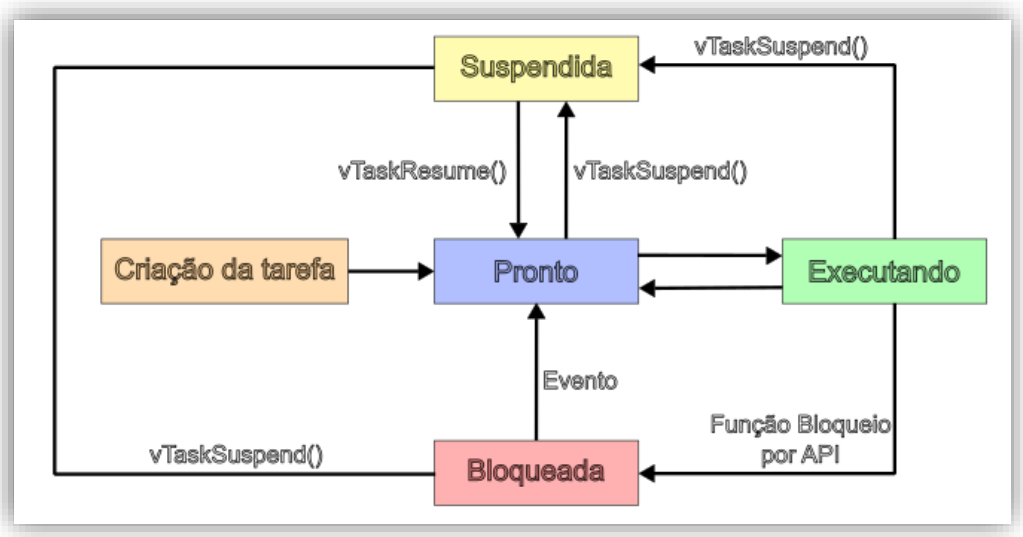
Utilização do RTOS

Para o seu funcionamento, é necessário configurar alguns recursos embutidos nesse sistema, sendo os principais:

- Definição das Prioridades de Tarefas;
- Tempo do Ciclo (geralmente 1 ms);
- Fila (controle de acesso de dados);
- MUTEX (Controle de acesso aos dados);
- Semáforo (Controle de Acesso aos dados).

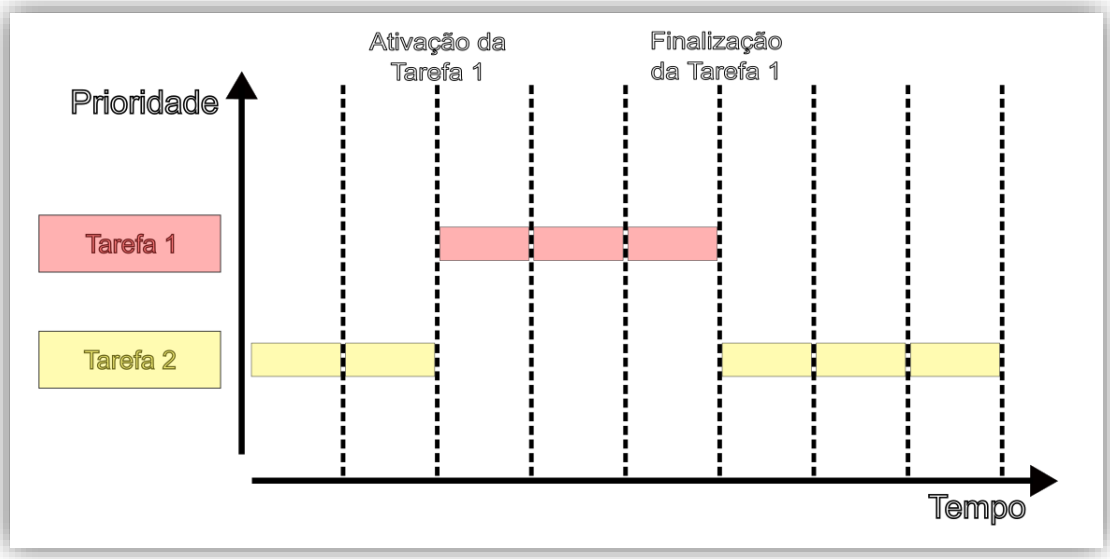
A Definição das Prioridades determina as tarefas críticas, pois as tarefas, de forma simplificada, possuem 4 estados definidos (Bloqueado, Aguardando, Ativo, Pausado), conforme a Figura 06, e caso uma tarefa de baixa prioridade esteja em execução quando uma nova tarefa de maior prioridade for requisitada, a tarefa de baixa prioridade vai entrar em suspensão quando entrar num novo ciclo e a tarefa de maior prioridade ficará ativa, com ambas as tarefas permanecendo nesse estado em todos os ciclos até que a tarefa prioritária finalize, ilustrado na Figura 07.

Figura 06. Estados das tarefas do RTOS



Fonte: Autores.

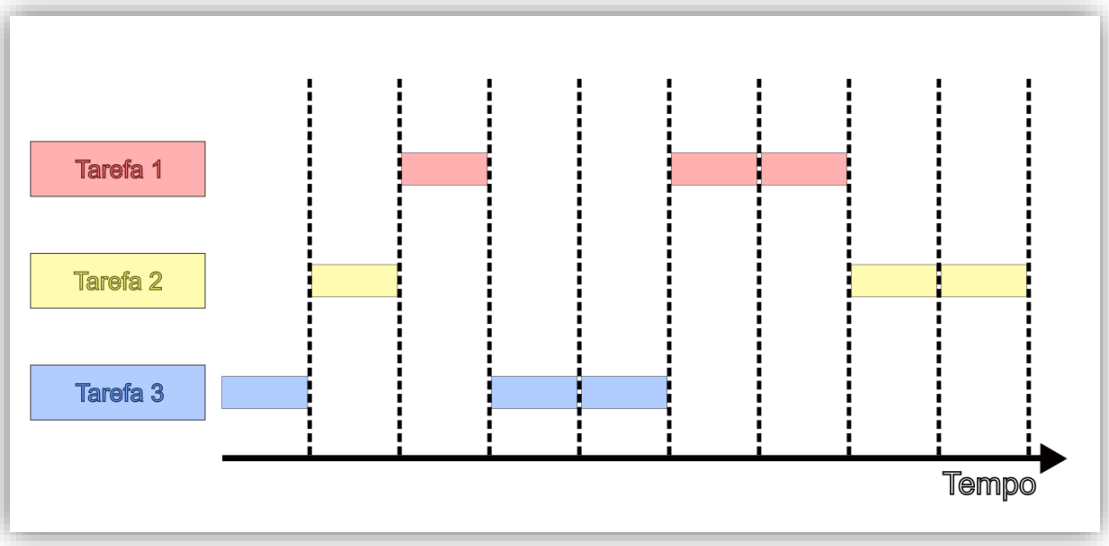
Figura 07. Funcionamento do Ciclo



Fonte: Autores.

O tempo de ciclo comentado anteriormente é um período em que o sistema irá realizar uma verificação se possui uma tarefa em espera, podendo estar pronta ou suspensa, além de verificar entre as tarefas qual é a mais prioritária, assim alocando os recursos e o processamento para a tarefa mais prioritária no momento, representado na Figura 08.

Figura 08. Processamento das tarefas

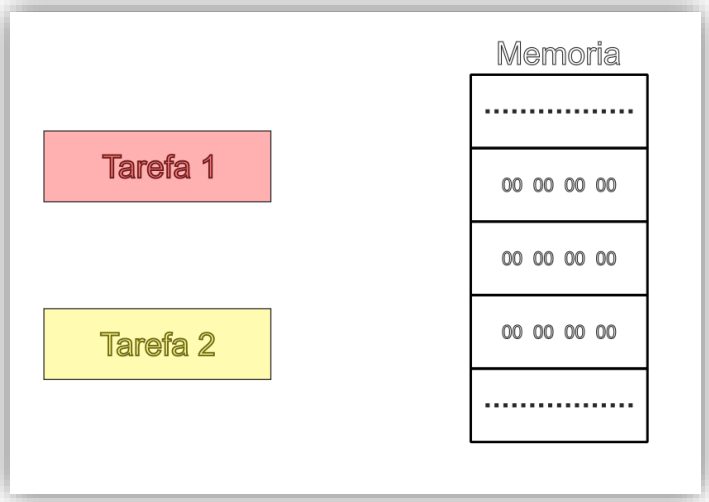


Fonte: Autores.

No entanto, ao utilizar o RTOS, é gerado um problema com o manuseio desses recursos entre as tarefas, podendo gerar dados errôneos, como exemplificado a seguir:

1. No estado inicial, nenhuma das tarefas está acessando o espaço determinado pela variável criada na memória, conforme a Figura 09.

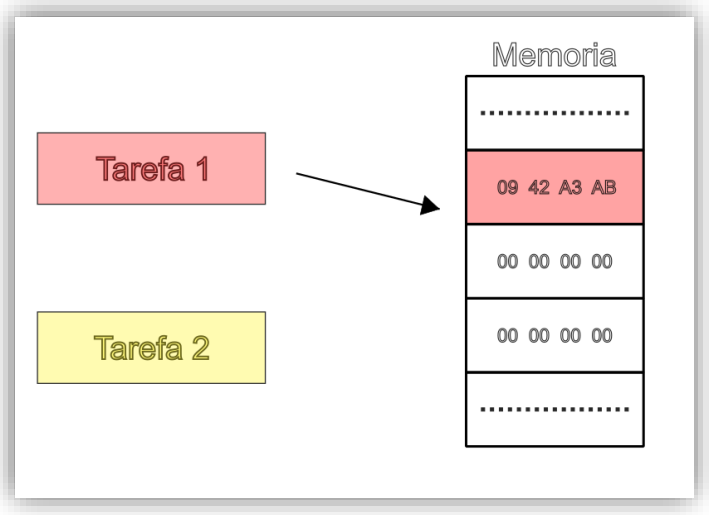
Figura 09. Estado inicial da memória



Fonte: Autores.

2. A Tarefa 1 é iniciada e começa a escrever na variável os seus dados, conforme a Figura 10.

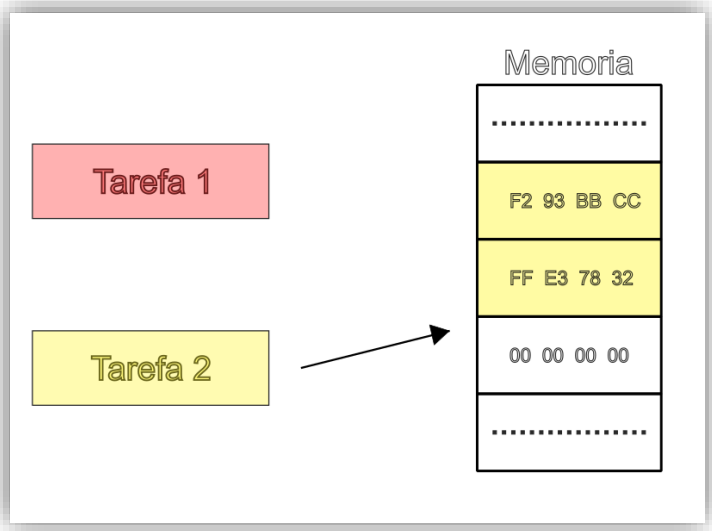
Figura 10. Acesso à memória pela tarefa 1



Fonte: Autores.

3. Durante a escrita da Tarefa 1, a Tarefa 2 é iniciada também, e ela possui uma prioridade maior que a tarefa 1, além de acessar o mesmo dado, interrompendo a escrita da Tarefa 1 e a sobrescrevendo (Figura 11).

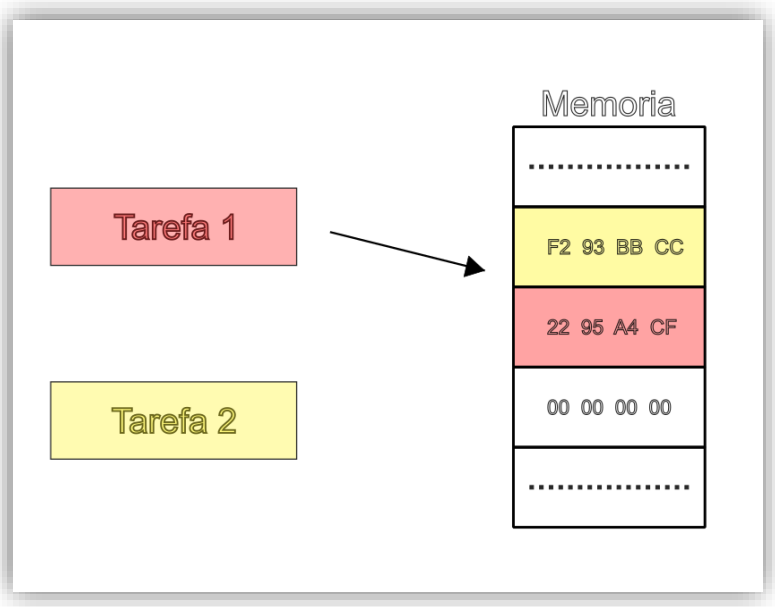
Figura 11. Interrupção da tarefa 1 pela tarefa 2



Fonte: Autores.

4. Quando a Tarefa 2 finaliza a sua execução, a Tarefa 1 retorna a escrita, mas ao finalizar, o dado está corrompido/errôneo, como demonstrado pela Figura 12.

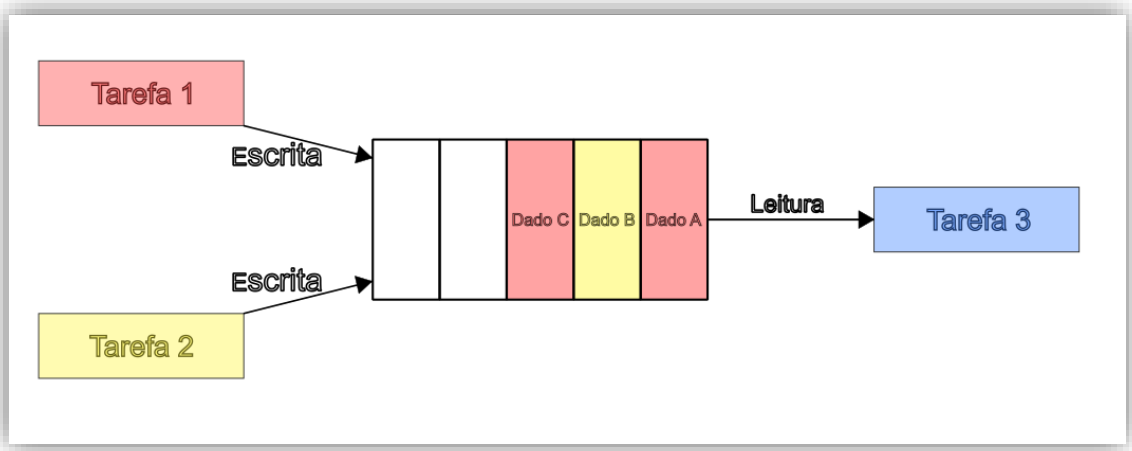
Figura 12. Retorno para a tarefa 1



Fonte: Autores.

Para isso, o RTOS possui recursos como a fila (*queue*), servindo como a primeira forma de transmissão de dados entre tarefas ou sincronização, fazendo que armazene dados num *Buffer*, como mostrado na Figura 13, para transmitir para as outras tarefas. Nele, pode-se realizar a escrita por múltiplas tarefas, pois a fila garante que os dados não serão corrompidos, com outra tarefa executando a leitura e removendo o dado da fila.

Figura 13. Funcionamento da fila (queue)



Fonte: Autores.

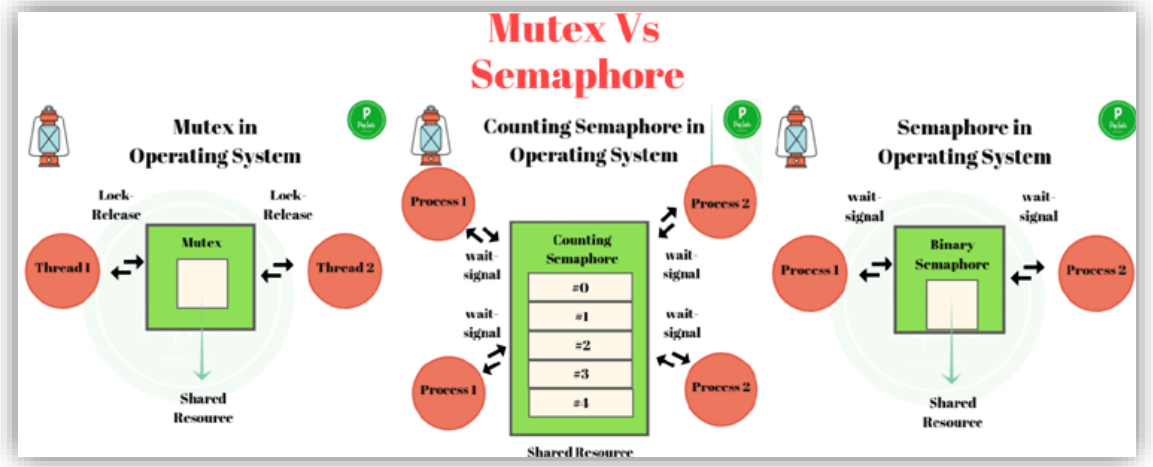
No trabalho, foi criada a fila para armazenar até 10 valores de 16 bits, possibilitando a transferência segura dos dados do ADC, adicionalmente no programa foi implementado um filtro de média móvel, que tem como objetivo reduzir os ruídos do sinal para que, em seguida, o sistema realize o processamento desses dados.

Outros recursos para controlar o acesso de dados são o Semáforo e o MUTEX (Mutual Exclusion), que sinalizam e bloqueiam o acesso a esses dados respectivamente para, assim, garantir o funcionamento correto da escrita e leitura de variáveis entre as distintas tarefas. Segundo Jain (2020), essas funções podem ser simplificadas em:

- Semáforo Binário: Ele sinaliza entre duas tarefas, podendo fazer com que a tarefa entre no estado de espera, até que seja liberado.
- Semáforo Counting: Ao invés de sinalizar apenas para duas tarefas, ele é capaz de sinalizar para múltiplas tarefas ou a quantidade de recursos disponíveis para utilizar.
- MUTEX: Ele funciona de forma similar ao semáforo binário, no entanto, tem a função de evitar a inversão de prioridades através da realização de uma prioridade dinâmica, mas consequentemente é mais lento do que o semáforo. (Jain, 2020).

  Na Figura 14 é mostrada uma simplificação do conteúdo sobre o semáforo e o MUTEX.

Figura 14. Funções de controle de acesso a dados



Fonte: Jain (2020).

CONTROLE PID

Para o controle de temperatura do sistema, foi desenvolvido um controle PID, que envolve três componentes principais: proporcional, integral e derivativo. Esse PID controlará uma saída PWM por meio da variação do seu duty cicle de forma proporcional ao valor calculado pelo controlador.

Esse sistema de controle PID, de acordo com Ogata (2011), funciona com uma realimentação pelo erro, que é a diferença entre o valor medido e o valor desejado (Setpoint), e o utiliza como base para o cálculo de cada componente. Que, resumidamente:

- O proporcional é calculado diretamente com base no erro atual, oferecendo uma resposta rápida, reduzindo drasticamente o erro inicial do sistema.
- O integral acumula o erro ao longo do tempo, assim eliminando os erros residuais que poderiam persistir.
- O derivativo responde à taxa de variação do erro, contribuindo para suavizar a resposta do sistema em mudanças abruptas.

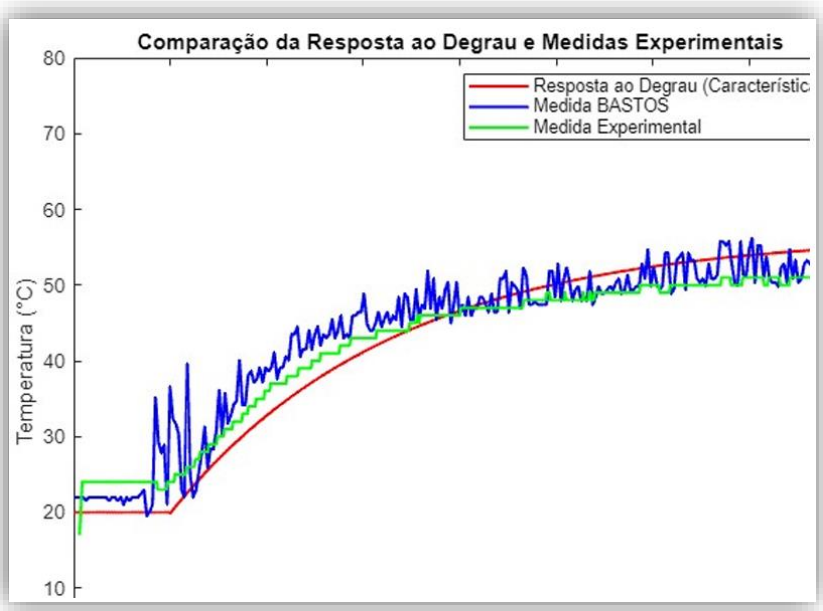
No trabalho, os parâmetros foram ajustados com: 60 no proporcional, 10 no integral e 5 no derivativo, devido a apresentar uma resposta rápida, com poucas oscilações e boa

estabilidade no circuito montado. Esses valores foram encontrados de forma experimental, abordando em múltiplos testes com distintas combinações dos ganhos até que chegasse a uma resposta rápida e estável. Os valores encontrados foram suficientes para minimizar o tempo necessário para atingir o Setpoint, reduzir oscilações e eliminar os erros de estado estacionário. Adicionalmente, foi implementada uma medida de proteção no sinal de saída, sendo ele o PWM, foi limitado entre 0% e 100%, evitando, assim, valores fora do intervalo aceitável.

RESULTADOS

No trabalho realizado por Bastos (2021), foi realizada a medição de temperatura de cinco resistores de valores distintos, mas nesse projeto foi somente considerado o resistor de 15Ω-2W, visando somente na validação do controle de temperatura em um RTOS embarcado. A Figura 15 compila os dados da resposta característica do sistema ao degrau, os valores encontrados por Bastos, e os dados obtidos experimentalmente, com os valores iniciais ajustados para 20°C e o setpoint para 50°C. Essa compilação foi realizada com o objetivo de traçar a curva de comportamento do circuito montado num RTOS embarcado no microcontrolador ARM, sem a implementação de um sistema de controle. Pode-se observar na imagem uma semelhança entre os valores teóricos e experimentais.

Figura 15. Comparação entre as respostas do sistema



Fonte: Autores.

Após a validação, foi implementado o controle PID explicado anteriormente no RTOS embarcado, com a função de realizar um ajuste automático de acordo com as necessidades do sistema para manter a saída de temperatura próxima ao setpoint. Para validar esse controle, foram propostos dois setpoints distintos (30 °C e 35 °C), simulando cenários de aquecimento e resfriamento. Durante os testes, o controlador PID realizou ajustes de forma dinâmica na potência dissipada pela carga para estabilizar a temperatura no valor desejado, cujo desempenho foi mais próximo da curva característica em relação ao trabalho de Bastos, tendo uma redução da diferença média percentual em 10,68%.

A aplicação do FreeRTOS foi fundamental para a estruturação e o desempenho eficiente do sistema embarcado. O firmware foi organizado em tarefas independentes, cada uma com uma

responsabilidade específica. Uma tarefa foi responsável pela leitura da temperatura, utilizando o conversor analógico-digital para obter os valores do sensor, outra tarefa responsável pela filtragem digital por média móvel, suavizando os dados para eliminar ruídos e, por fim, uma terceira tarefa para o controle PID, utilizando os dados processados para calcular o erro, aplicar os ganhos e ajustar o sinal PWM responsável por controlar a potência do atuador.

Essas tarefas operam de forma assíncrona, mas coordenada, graças ao uso de filas. As filas garantiram que os dados fluíssem corretamente entre as etapas do sistema, da leitura do sensor até o controle final da temperatura. Isso evitou a perda de dados e possíveis interferências entre as tarefas, permitindo que cada uma operasse em seu próprio ritmo, sem travamentos e bloqueios.

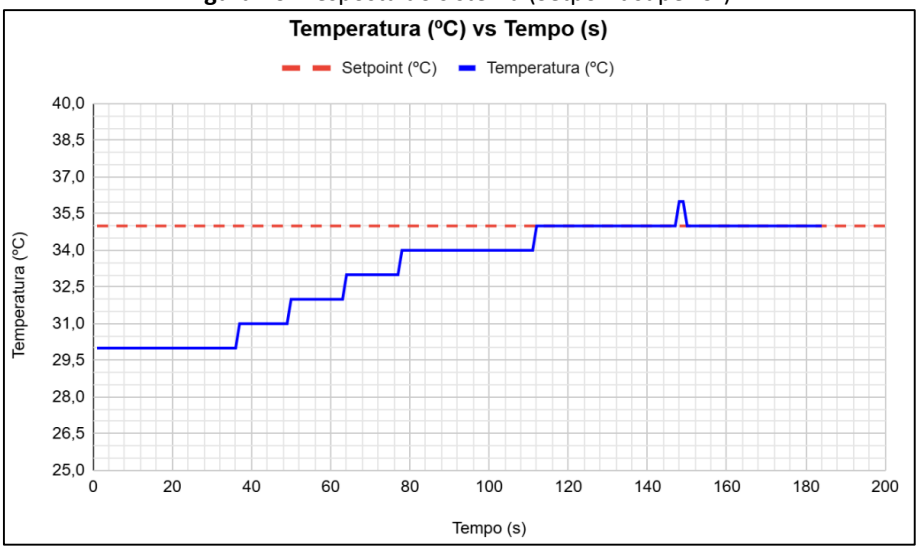
O sistema também se beneficiou do uso do escalonador preemptivo do FreeRTOS, que desempenhou um papel fundamental ao garantir que as tarefas mais críticas, como o controle PID, fossem executadas com prioridade e no momento exato necessário. Isso assegurou que o sistema mantivesse sua capacidade de resposta, mesmo com múltiplas tarefas em execução simultânea, evitando atrasos indesejados no controle da temperatura.

Essa arquitetura baseada em RTOS proporcionou diversos benefícios ao projeto. A modularidade permitiu que cada parte do sistema fosse isolada em uma tarefa, facilitando o desenvolvimento, teste e manutenção. A previsibilidade temporal garantiu que o sistema respondesse em tempo real, com execuções programadas de forma determinística. A escalabilidade foi favorecida, já que novas funcionalidades podem ser facilmente adicionadas por meio de novas tarefas. E, por fim, a robustez foi fortalecida pela separação de responsabilidades e pela comunicação estruturada, que evitaram travamentos e conflitos.

Adicionalmente, o erro residual se manteve próximo de zero após o período transitório, demonstrando a eficácia do controle implementado. Os gráficos ilustrados pelas figuras 16 e 17 apresentam os resultados adquiridos, mostrando a evolução da temperatura em função do tempo para diferentes setpoints, com um estando num valor superior ao estado inicial do sistema e outro inferior. Cada curva demonstra a capacidade do sistema de atingir o valor desejado com estabilidade, sem oscilações prolongadas.

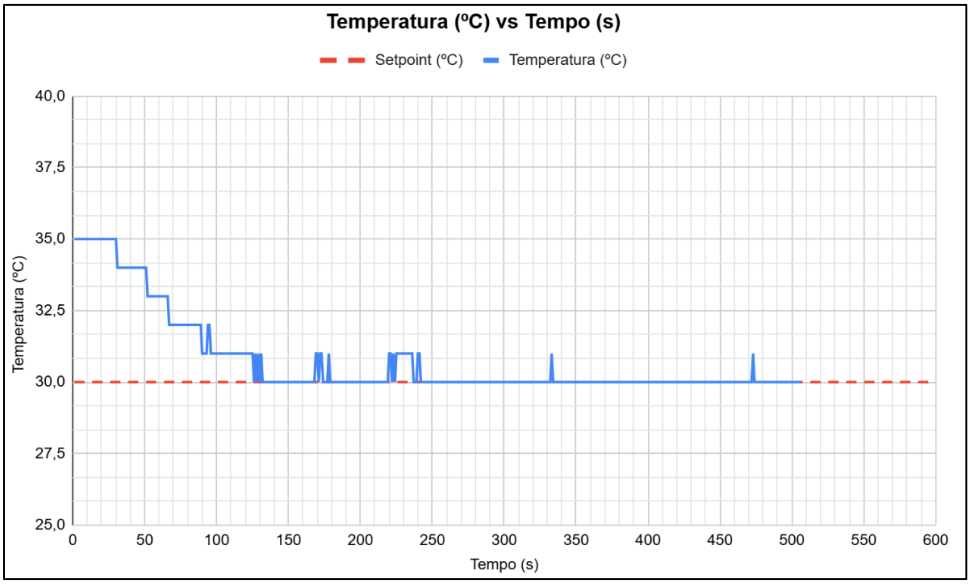
Portanto, mais do que apenas um sistema de controle de temperatura, o trabalho validou, na prática, o uso de um RTOS como ferramenta poderosa para o desenvolvimento de sistemas embarcados, promovendo organização, desempenho e confiabilidade, características essenciais em aplicações críticas.

Figura 16. Resposta do sistema (Setpoint superior)



Fonte: Autores.

Figura 17. Resposta do Sistema (Setpoint Inferior)



Fonte: Autores.

Após a análise dos dados, pode-se concluir que o sistema de controle de temperatura apresentou resultados satisfatórios, destacando-se pela eficiência em atingir e manter diferentes setpoints com precisão e estabilidade. Dentro dessa análise, se destacam os seguintes pontos:

- Estabilidade do controle: O sistema foi capaz de alcançar os setpoints definidos, que mesmo em condições que simulavam as variações ambientais, possuía oscilações mínimas, demonstrando uma robustez;
- Organização do RTOS: A aplicação do FreeRTOS organizou as tarefas críticas de forma eficiente e simples, permitindo assim execução ordenada e previsível das tarefas;
- Precisão do PID: O PID corrigiu o erro em 112 segundos, já alcançando o setpoint nesse tempo, além de apresentar uma resposta suave, sem possuir overshoots significativos.

CONCLUSÃO

Os resultados preliminares deste trabalho demonstram que a aplicação de um RTOS para a medição de temperatura apresenta semelhanças com os resultados de estudos anteriores, mesmo um método de programação não convencional. Ao analisar os gráficos obtidos com os do trabalho de Bastos (2021), indicam que, apesar das diferenças nas abordagens e ferramentas utilizadas, o comportamento térmico do sistema foi similar mesmo com a utilização do RTOS.

Na parte do controle de temperatura, os dados comprovam que a utilização do RTOS traz benefícios para o desenvolvimento de sistemas embarcados, já que trazem benefícios como a simplificação do gerenciamento das tarefas e de seus recursos, a determinação das prioridades e a sua velocidade de alcançar os tempos críticos. Com os resultados mostrando um controle preciso e rápido, mesmo utilizando uma programação de um simples controlador PID.

Dessa forma, este trabalho contribui para uma compreensão mais aprofundada dos benefícios do uso do RTOS em sistemas embarcados, disponibilizando uma base para futuros projetos na área de sistemas embarcados em controle de processos

REFERÊNCIAS

Bastos, C. A.; Rech, R. Estudo de métodos de controle para um sistema térmico. *In:* 12o Congresso de Inovação, Ciência e Tecnologia do IFSP (CONICT). São Paulo: Instituto Federal de São Paulo (IFSP), 2021. p. 1–5. Disponível em: https://ocs.ifsp.edu.br/conict/xiiconict/paper/view/7375/2538 . Acesso em: 16 set. 2024.

Bertoleti, P. *Principais conceitos de RTOS para iniciantes com Arduino e FreeRTOS Brasil*: Embarcados, 2019. Disponível em: https://embarcados.com.br/rtos-para-iniciantes-com-arduino-e-freertos/. Acesso em 8 dez. 2024.

Cardoso, M. O Que É Um Microcontrolador? Brasil: *IEEE RAS UFCG*, 2020. Disponível em: https://edu.ieee.org/br-ufcgras/o-que-e-um-microcontrolador/. Acesso em 30 jun. 2025.

Jain, R. Arduino FreeRTOS Tutorial 3- Using Semaphore and Mutex in FreeRTOS with Arduino India: *Circuit Digest*, 2020. Disponível em: https://circuitdigest.com/microcontroller-projects/arduino-freertos-tutorial-using-semaphore-and-mutex-in-freertos-with-arduino. Acesso em 20 out. 2024.

Leroux, P. N. *RTOS versus GPOS*: What is best for embedded development? Canada: QNX Software Systems Ltd, 2005.

Ogata, K. *Engenharia de Controle Moderno*. 5ª Edição Brasil: São Paulo, 2011.

Schneider, J.; Smalley, I. *O que é um microcontrolador?* EUA: IBM, 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/microcontroller. Acesso em 30 jun. 2025.

TPointTech *Difference between Real-Time operating system and general-purpose operating system.* India: Noida, 2025. Disponível em: https://www.tpointtech.com/real-time-operating-system-vs-general-purpose-operating-system. Acesso em 20 out. 2024.

Walls, C. *Embedded RTOS Design*: Insights and Implementation Grã-Bretanha: Newnes, 2021.

Vaibhav. What Makes RTOS an Ideal Choice for the Next Generation Embedded Applications? India: Embitel, 2019. Disponível em: https://www.embitel.com/blog/embedded-blog/what-is-rtos-and-why-is-it-critical-for-embedded-systems/. Acesso em 29 jun. 2025.

Zhao, H. RTOS vs GPOS: A Complete Guide China: WellPCB, 2021. Disponível em: https://www.wellpcb.com/blog/comparisons/rtos-vs-gpos/#Wrapping_Up. Acesso em 20 out. 2024.