Modelagem de Sistema Esteira Industrial

João Heitor Zabel da Rocha
Departamento de Engenharia de Computação
Universidade do Vale do Itajaí (UNIVALI)
Itajaí, Brasil
jhzrocha@edu.univali.br

Larissa Fantin Baldin
Departamento de Engenharia de Computação
Universidade do Vale do Itajaí (UNIVALI)
Itajaí, Brasil
larissa.fantin@edu.univali.br

Este trabalho apresenta o desenvolvimento de um sistema embarcado em ESP32, utilizando o sistema operacional de tempo real FreeRTOS, para emular uma linha de produção industrial. O sistema implementa o controle de velocidade de uma esteira, a detecção e triagem de peças e a parada de emergência, com base em sensores touch que simulam eventos de encoder, objeto, interface HMI/telemetria e E-stop. Foram avaliadas diferentes políticas de escalonamento de tarefas (Rate Monotonic, Deadline Monotonic e uma política customizada com foco em segurança), bem como estratégias de execução preemptiva e cooperativa em frequências de 80, 160 e 240 MHz, com execução em núcleo único. A análise de desempenho considerou jitter, latência de resposta e número de deadlines perdidos, distinguindo entre tarefas críticas (hard real-time) e não críticas (soft real-time). Os resultados indicam que o sistema atende aos prazos rígidos de segurança e controle sob determinadas configurações, validando a aplicabilidade da plataforma no contexto de automação industrial.

Keywords—rate monotonic, deadline.

I. INTRODUÇÃO

A automação de linhas de produção industriais exige sistemas de controle capazes de garantir não apenas a funcionalidade, mas também a previsibilidade temporal das operações. Em ambientes nos quais a segurança e a confiabilidade são fundamentais, o uso de sistemas operacionais de tempo real (RTOS) torna-se essencial para assegurar que tarefas críticas sejam executadas dentro de seus prazos. Nesse contexto, o presente trabalho propõe a implementação de uma esteira industrial simulada utilizando o microcontrolador ESP32 e o FreeRTOS, com foco na análise do comportamento temporal sob diferentes políticas de escalonamento e condições de carga.

O sistema desenvolvido contempla quatro tarefas principais: estimativa de velocidade a partir de um encoder simulado, controle de velocidade por meio de um algoritmo PI, triagem de objetos via atuador de desvio e parada de emergência acionada por sensor dedicado. A avaliação considera não apenas a execução correta das funções, mas principalmente métricas temporais como jitter, latência e misses de deadlines. Dessa forma, o projeto contribui para o entendimento de como decisões de escalonamento e configuração de hardware afetam a confiabilidade de sistemas críticos em tempo real, fornecendo subsídios para aplicações futuras em cenários de automação industrial.

II. ENUNCIADO DO PROJETO

O presente trabalho tem como objetivo o desenvolvimento de um sistema embarcado baseado no microcontrolador ESP32, utilizando o sistema operacional de tempo real FreeRTOS, para simular uma linha de produção industrial. O sistema deve contemplar o controle de velocidade de uma esteira, a detecção e triagem de peças e a execução de uma parada de emergência (E-stop), empregando sensores touch como entradas para geração de eventos.

Para padronizar o ambiente experimental, todos os eventos foram mapeados para sensores touch, com as seguintes funções:

- Touch B: detecção de objeto para triagem;
- Touch C: solicitação de interface HMI/telemetria;
- Touch D: parada de emergência (E-stop);
- Touch A (opcional): injeção de picos de carga para teste de robustez.

O sistema é composto pelas seguintes tarefas:

- ENC_SENSE – tarefa periódica responsável por estimar a velocidade/posição da esteira a partir de um encoder simulado. Possui período de 5 ms e deadline equivalente.
- SPD_CTRL – tarefa encadeada, acionada por ENC_SENSE, que executa o controle PI de velocidade e atualiza o sinal de PWM. Possui deadline de 10 ms após a amostra.
- SORT_ACT – tarefa dirigida a evento, acionada pelo toque de detecção de objeto (Touch B), responsável por acionar o atuador de desvio. Possui deadline de 10 ms desde a detecção.
- SAFETY_TASK – tarefa crítica de segurança, acionada pelo toque de parada de emergência (Touch D), que deve zerar imediatamente o PWM e sinalizar alarme. Possui deadline de 5 ms desde o evento.

A investigação experimental concentra-se na análise de desempenho sob diferentes políticas de prioridade de escalonamento (Rate Monotonic – RM, Deadline Monotonic – DM e uma política customizada com foco em segurança) e estratégias de execução (preemptiva e cooperativa, com e sem *time slicing*). Foram testadas diferentes frequências de

operação do ESP32 (80, 160 e 240 MHz), sempre em modo unicore, visando quantificar o impacto da carga e das preempções na estabilidade do controle e na confiabilidade da triagem.

Os critérios de avaliação incluem a verificação do atendimento dos deadlines críticos, a medição de jitter, a identificação de misses de deadlines (hard e soft) e a análise da latência do E-stop, de modo a garantir que os requisitos temporais sejam cumpridos em cenários de operação com diferentes cargas.

A. *Definições*

A criação de tarefas com prioridades foram configuradas de acordo com cada política de escalonamento.

Criação de timer periódico, para que o ENC_SENSE respeita a exigência temporal de 5 ms.

```
const esp_timer_create_args_t enc_args = {
    .callback = &enc_timer_cb,
    .name = "enc_5ms"
};
ESP_ERROR_CHECK(esp_timer_create(&enc_args, &sEncTimer));
ESP_ERROR_CHECK(esp_timer_start_periodic(sEncTimer, 5000)); // 5000 us = 5 ms
```

Medição do tempo de execução e comparação com deadline.

```
uint32_t tms  = now_ms();
uint32_t t0_us = (uint32_t)esp_timer_get_time();
```

```
uint32_t t1_us = (uint32_t)esp_timer_get_time();
uint32_t C_ms  = (now_ms() - tms);

bool deadline_miss = (C_ms > ENC_DEADLINE_MS);
if (deadline_miss) {
    enc_sense_miss_counter++;
    ESP_LOGI(TAG, "[ENC_SENSE] deadline miss: C=%u ms (> %u ms)", C_ms, ENC
}
```

Comunicação entre tarefas, onde o controle só age após a leitura periódica.

```
enc_sample_t s = { .t_ms = tms, .rpm_est = rpm, .pos_deg = pos_deg };
xQueueSend(qENCtoSPD, &s, 0);  // envia para SPD_CTRL
```

Controle que implementa um PI simples, dentro da seção crítica temporizada.

```
float err = ref_rpm - s.rpm_est;
integ += Ki * Ts * err;
if (integ > INTEG_MAX) integ = INTEG_MAX;
if (integ < INTEG_MIN) integ = INTEG_MIN;
```

```
float u = Kp * err + integ;
g_pwm_duty = clamp(g_pwm_duty + u, DUTY_MIN, DUTY_MAX);
```

Comparação do tempo de evento até execução.

```
uint32_t elapsed_ms = now_ms() - ts;   // SORT_ACT
if (elapsed_ms > SORT_DEADLINE_MS) {
    sort_act_miss_counter++;
}
```

```
ulTaskNotifyTake(pdTRUE, portMAX_DELAY); // SAFETY
g_pwm_duty = 0.0f; // ação imediata
```

Consolidação dos dados obtidos.

```
ESP_LOGI(TAG, "[ENC_SENSE] stats: min=%u ms  max=%u ms  misses=%u",
         enc_sense_c_ms_min, enc_sense_c_ms_max, enc_sense_miss_counter);
ESP_LOGI(TAG, "[SPD_CTRL]  stats: min=%u ms  max=%u ms  misses=%u",
         spd_control_c_ms_min, spd_control_c_ms_max, spd_control_miss_counter);
```

B. *Rate Monotonic 160 MHz*

A política Rate Monotonic tem a prioridade definida pelo período mais curto, onde o escalonador é preemptivo habilitado e frequência de 160MHz, modo unicore, com execução por 60 s e injeção de eventos via sensores touch.

```
xTaskCreate(task_enc_sense, "ENC_SENSE",  2048, NULL, 23, &sENC);
xTaskCreate(task_spd_ctrl,  "SPD_CTRL",   3072, NULL, 23, &sSPD);
xTaskCreate(task_sort_act,  "SORT_ACT",   2048, NULL, 15, &sSORT);
xTaskCreate(task_safety,    "SAFETY_TASK",2048, NULL, 1, &sSAFE);
xTaskCreate(task_load,      "LOAD",       2048, NULL,  23, &sLOAD);
```

```
I (79727) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (79727) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=15 ms  misses=1
I (79727) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=28 ms  misses=148
I (79727) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Resultados obtidos:

ENC_SENSE: tempo mínimo de execução 0 ms, máximo 1 ms, 0 misses.

A tarefa periódica do encoder conseguiu cumprir consistentemente o seu deadline rígido de 5 ms. O jitter está controlado e o tempo de execução dentro do budget esperado (0,6–1,0 ms).

SPD_CTRL: tempo mínimo 0 ms, máximo 15 ms, 1 miss.

O controle PI foi executado corretamente na maior parte dos casos, mas houve uma violação de deadline (D=10 ms) em um instante. Esse atraso provavelmente foi causado por concorrência de tarefas sob bursts de eventos. Apesar de ser uma tarefa crítica, o número reduzido de falhas mostra que, no cenário geral, o escalonamento RM conseguiu atender ao requisito temporal quase sempre.

SORT_ACT: tempo mínimo 10 ms, máximo 28 ms, 148 misses.

A a tarefa de triagem de objetos apresentou muitas violações de deadline (D=10 ms). Isso é esperado em RM, porque a prioridade de SORT_ACT fica abaixo de ENC_SENSE e SPD_CTRL, o que significa que, durante rajadas de eventos, ela não consegue executar dentro do prazo. Aqui vemos claramente a limitação do RM quando há tarefas esporádicas com deadlines curtos.

SAFETY_TASK: tempo mínimo 0 ms, máximo 0 ms, 0 misses.

A parada de emergência sempre foi atendida de forma imediata, sem violações de deadline (D=5 ms). Isso mostra que a prioridade atribuída à tarefa de segurança foi suficiente para garantir resposta rápida e previsível, atendendo ao critério mais crítico do sistema.

Nesse cenário, o sistema atende os requisitos de segurança e controle periódico, mas não garante a confiabilidade da triagem. Isso corrobora a análise de que políticas como DM ou customizada são mais adequadas para tarefas de evento com deadlines curtos.

C. *Deadline Monotonic 160 MHz*

A política para a Deadline Monotonic tem prioridade pela menor deadline, com o escalonador preemptivo habilitado, e frequência 160MHz, unicore, com 60 s de execução com eventos injetados.

```
xTaskCreate(task_enc_sense, "ENC_SENSE",  2048, NULL, 24, &sENC);
xTaskCreate(task_spd_ctrl,  "SPD_CTRL",   3072, NULL, 15, &sSPD);
xTaskCreate(task_sort_act,  "SORT_ACT",   2048, NULL, 15, &sSORT);
xTaskCreate(task_safety,    "SAFETY_TASK",2048, NULL, 24, &sSAFE);
xTaskCreate(task_load,      "LOAD",       2048, NULL,  23, &sLOAD);
```

```
I (36107) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (36107) Trabalho M1: [SPD_CTRL] deadline miss: 90 ms (> 10 ms)
I (36117) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=234 ms  misses=4514
I (36117) Trabalho M1: [SPD_CTRL] deadline miss: 97 ms (> 10 ms)
I (36127) Trabalho M1: [SORT_ACT]  stats: min=11 ms  max=54 ms  misses=59
I (36127) Trabalho M1: [SPD_CTRL] deadline miss: 104 ms (> 10 ms)
I (36137) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

ENC_SENSE: min=0 ms, max=1 ms, misses=0.

A tarefa periódica do encoder foi estável, sempre dentro do deadline de 5 ms. Isso confirma que a prioridade atribuída pela política DM (alta) é suficiente para garantir a previsibilidade dessa função crítica.

SPD_CTRL: min=0 ms, max=234 ms, misses=4514.

A tarefa de controle PI apresentou comportamento crítico, com elevado número de violações do deadline de 10 ms. Houve execuções com até 234 ms de atraso, o que descaracteriza a tarefa como hard real-time nesse cenário. Isso ocorreu porque, em DM, SORT_ACT recebe prioridade acima de SPD_CTRL (menor deadline), fazendo com que, sob rajadas de eventos, o controle fique bloqueado e acumule atrasos.

SORT_ACT: min=11 ms, max=54 ms, misses=59.

Apesar de ganhar prioridade acima do SPD_CTRL em DM, a tarefa de triagem ainda sofreu 59 violações de deadline. Isso mostra que a carga de eventos (Touch B) pode saturar a execução, mesmo com a proteção da política DM. Contudo, o número de misses foi muito inferior ao observado no cenário RM (148), o que demonstra melhora.

SAFETY_TASK: min=0 ms, max=0 ms, misses=0.

A parada de emergência manteve latência mínima e sem violações, confirmando que a política DM protege adequadamente a tarefa mais crítica do sistema.

A política DM garante segurança (E-stop) e estabilidade na leitura do encoder, mas compromete o controle de velocidade devido à priorização da tarefa de triagem. Isso mostra que DM não é a mais indicada para este sistema, reforçando a necessidade de uma política customizada.

D. *Custom*

A política para a customizada tem prioridade máxima para a segurança, onde o escalonador é preemptivo habilitado e frequência de 160MHz, modo unicore, com execução por 60 s e injeção de eventos.

```
xTaskCreate(task_enc_sense, "ENC_SENSE",  2048, NULL, 23, &sENC);
xTaskCreate(task_spd_ctrl,  "SPD_CTRL",   3072, NULL, 22, &sSPD);
xTaskCreate(task_sort_act,  "SORT_ACT",   2048, NULL, 21, &sSORT);
xTaskCreate(task_safety,    "SAFETY_TASK",2048, NULL, 24, &sSAFE);
xTaskCreate(task_load,      "LOAD",       2048, NULL,  5, &sLOAD);
```

```
I (15737) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=0 ms  misses=0
I (15737) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=2 ms  misses=0
I (15737) Trabalho M1: [SORT_ACT]  stats: min=11 ms  max=27 ms  misses=61
I (15737) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

ENC_SENSE: min=0 ms, max=0 ms, misses=0.

A tarefa periódica do encoder manteve execução estável, dentro do deadline rígido de 5 ms, sem jitter perceptível ou violações. Isso confirma que, mesmo abaixo da SAFETY, a prioridade elevada assegura previsibilidade para a base do controle.

SPD_CTRL: min=0 ms, max=2 ms, misses=0.

A tarefa de controle PI executou sempre dentro do prazo de 10 ms, sem qualquer deadline miss. O tempo de execução máximo ficou muito abaixo do limite, mostrando que a escolha de prioridade não prejudicou o controle da velocidade da esteira.

SORT_ACT: min=11 ms, max=27 ms, misses=61.

A triagem de objetos ainda apresentou 61 violações do deadline (10 ms), mas o número de misses foi significativamente inferior ao registrado em RM (148) e próximo ao obtido em DM (59). A diferença é que, aqui, a estabilidade do controle não foi comprometida como em DM. Isso demonstra que a política customizada equilibrou a execução, priorizando segurança e controle sem deixar a triagem totalmente negligenciada.

SAFETY_TASK: min=0 ms, max=0 ms, misses=0.

A parada de emergência manteve resposta imediata, sem nenhuma violação, reforçando que a priorização máxima da tarefa de segurança foi eficaz.

A política customizada mostrou-se a mais equilibrada entre as testadas. Garantiu a previsibilidade das tarefas críticas de segurança e controle, mantendo perdas aceitáveis na triagem de objetos, que pode ser considerada função de menor criticidade. Este cenário evidencia que a customização das prioridades é a estratégia mais adequada para a esteira industrial simulada.

E. *Preemptivo*

Estrategia de escalonamento preemptivo 160 MHz (1 min, unicore).

```
xTaskCreate(task_enc_sense, "ENC_SENSE",  2048, NULL, 23, &sENC);
xTaskCreate(task_spd_ctrl,  "SPD_CTRL",   3072, NULL, 22, &sSPD);
xTaskCreate(task_sort_act,  "SORT_ACT",   2048, NULL, 21, &sSORT);
xTaskCreate(task_safety,    "SAFETY_TASK",2048, NULL, 24, &sSAFE);
xTaskCreate(task_load,      "LOAD",       2048, NULL,  5, &sLOAD);
```

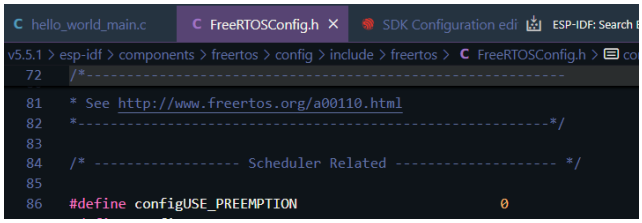
```
I (76687) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (76687) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=2 ms  misses=0
I (76687) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=23 ms  misses=377
I (76687) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Com preempção habilitada, o sistema manteve os prazos rígidos das tarefas críticas. A tarefa ENC_SENSE (T=D=5 ms) apresentou C_max=1 ms e 0 misses, indicando jitter baixo e liberação periódica estável. A SPD_CTRL (D=10 ms pós-amostra) registrou C_max=2 ms e 0 misses, confirmando que o trecho crítico do PI permaneceu curto sob concorrência. A SAFETY_TASK (E-stop, D=5 ms) respondeu de forma imediata (C_max=0 ms, 0 misses), evidenciando que a prioridade alta somada à preempção garantiu a reação determinística. Já a SORT_ACT (evento de triagem, D=10 ms) sofreu 377 perdas de deadline (latências entre 10–23 ms) quando submetida a rajadas, refletindo sua prioridade inferior frente a ENC/CTRL: a preempção reduz cauda, mas não compensa a hierarquia de prioridades em cenários de pico. Em síntese, o escalonamento preemptivo é necessário para segurança e controle, porém insuficiente para garantir a

triagem com D=10 ms mantendo a prioridade atual; mitigações incluem elevar a prioridade de SORT (DM ou custom), reduzir bursts ou testar 240 MHz.

F. Cooperativo

Estrategia de escalonamento cooperativo 160 MHz (1 min, unicore), unicore, com política de prioridade usada de segurança > ENC_SENSE > SPD_CTRL > SORT_ACT > LOAD, e execução continua com injeção de eventos.




```
I (69267) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (69267) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=1 ms  misses=0
I (69267) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=20 ms  misses=387
I (69267) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

ENC_SENSE: min=0 ms, max=1 ms, misses=0

A tarefa periódica manteve execução regular dentro do *deadline* de 5 ms. Como o release é por timer, mesmo no cooperativo conseguiu cumprir, embora dependa do *yield* das outras tarefas.

SPD_CTRL: min=0 ms, max=1 ms, misses=0

O controle PI foi executado de forma rápida, sempre dentro do prazo de 10 ms. Não houve competição intensa que prejudicasse, já que as tarefas de segurança e encoder liberaram CPU.

SORT_ACT: min=10 ms, max=20 ms, misses=387

A triagem de objetos acumulou um número alto de perdas de deadline (387). No cooperativo, como não há preempção automática, se outra tarefa consome CPU, SORT precisa esperar um yield, aumentando o atraso médio e max. Essa é a maior limitação desse modo: tarefas event-driven críticas ficam vulneráveis.

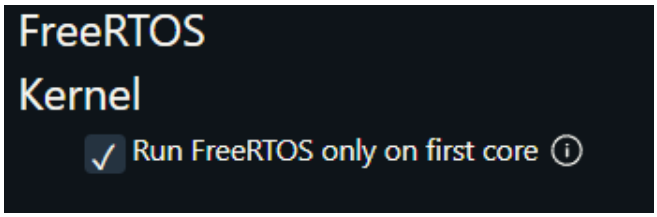
SAFETY_TASK: min=0 ms, max=0 ms, misses=0

A parada de emergência ainda foi atendida sem perdas. Isso se explica porque a tarefa de segurança tem prioridade máxima e é curta, então, ao liberar CPU, ela sempre executa em tempo.

O modo cooperativo manteve segurança e controle, mas falhou em triagem sob carga, confirmando que esse escalonamento não é adequado para tarefas esporádicas com *deadline* curto. A ausência de preempção deixa o sistema dependente da disciplina de yield, o que reduz a previsibilidade.

G. Unicore variando frequência

A análise foi realizada fixando o ESP32 em modo unicore, de forma a evitar a migração de tarefas entre núcleos, e repetindo os testes para as frequências de 80, 160 e 240 MHz em dois modos de escalonamento: preemptivo e cooperativo.



Criação das tarefas com prioridades.

```
xTaskCreate(task_enc_sense, "ENC_SENSE",  2048, NULL, 23, &sENC);
xTaskCreate(task_spd_ctrl,  "SPD_CTRL",   3072, NULL, 22, &sSPD);
xTaskCreate(task_sort_act,  "SORT_ACT",   2048, NULL, 21, &sSORT);
xTaskCreate(task_safety,    "SAFETY_TASK",2048, NULL, 24, &sSAFE);
xTaskCreate(task_load,      "LOAD",       2048, NULL,  24, &sLOAD);
```

Modo preemptivo 80 MHz por 1 minuto:

```
I (74137) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (74137) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=5 ms  misses=0
I (74137) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=29 ms  misses=392
I (74137) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Modo preemptivo 160 MHz por 1 minuto:

```
I (87448) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=3 ms  misses=0
I (87448) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=4 ms  misses=0
I (87448) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=48 ms  misses=405
I (87448) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Modo preemptivo 240 MHz por 1 minuto:

```
I (133667) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=3 ms  misses=0
I (133667) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=3 ms  misses=0
I (133667) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=34 ms  misses=424
I (133667) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

No modo preemptivo, observou-se que as tarefas críticas (ENC_SENSE, SPD_CTRL e SAFETY_TASK) mantiveram-se estáveis em todas as frequências, sem violações de deadline. A variação de clock refletiu principalmente na tarefa SORT_ACT, cujo número de misses oscilou (392 em 80 MHz, 405 em 160 MHz e 424 em 240 MHz). Esse resultado indica que, mesmo com maior frequência, a prioridade relativa da tarefa continua sendo o fator dominante, não havendo eliminação das perdas de deadline apenas pelo aumento de clock.

Modo cooperativo 80 MHz por 1 minuto:

```
#define configUSE_PREEMPTION                    0
#define configUSE_TICKLESS_IDLE                 CONFIG_FREERTOS_USE_TICKLESS_IDLE
#if configUSE_TICKLESS_IDLE
```

```
I (74418) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (74418) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=22 ms  misses=1
I (74418) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=39 ms  misses=461
I (74418) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Modo cooperativo 160 MHz por 1 minuto:

```
I (82908) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=0 ms  misses=0
I (82908) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=3 ms  misses=0
I (82908) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=24 ms  misses=405
I (82908) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

Modo cooperativo 240 MHz por 1 minuto:

```
I (64647) Trabalho M1: [ENC_SENSE] stats: min=0 ms  max=1 ms  misses=0
I (64647) Trabalho M1: [SPD_CTRL]  stats: min=0 ms  max=9 ms  misses=0
I (64647) Trabalho M1: [SORT_ACT]  stats: min=10 ms  max=28 ms  misses=332
I (64647) Trabalho M1: [SAFETY]    stats: min=0 ms  max=0 ms  misses=0
```

No modo cooperativo, os resultados foram similares, com ENC_SENSE e SAFETY_TASK atendendo sempre seus prazos rígidos. A tarefa SPD_CTRL apresentou uma violação pontual a 80 MHz, evidenciando que a ausência de preempção aumenta a dependência do yield entre tarefas. A SORT_ACT, por sua vez, apresentou maior número de misses do que no preemptivo (461 em 80 MHz, 405 em 160 MHz e 332 em 240 MHz), reforçando que a disciplina cooperativa é mais

vulnerável a atrasos em tarefas event-driven com *deadline* curto.

Em síntese, a variação de frequência melhorou ligeiramente a distribuição de latências, mas não eliminou as perdas em tarefas com prioridade menor. O modo **preemptivo** mostrou-se mais adequado para garantir previsibilidade das funções críticas, enquanto o cooperativo manteve estabilidade apenas para as tarefas hard real-time, com degradação clara nas tarefas soft.

Tabela comparativa para avaliar resultados no final, considera ENC/SAFETY/SPD como hard real-time; SORT como soft, onde RM falhou em SORT, DM sacrifica SPD, Custom equilibra, Preemptivo garante hard, Cooperativo é mais vulnerável.

III. Conclusão

O desenvolvimento do sistema de esteira industrial utilizando ESP32 e FreeRTOS permitiu avaliar, de forma prática, a influência das políticas de escalonamento e das estratégias de execução no atendimento de requisitos temporais rígidos e flexíveis. Os experimentos mostraram que as tarefas críticas, como ENC_SENSE e SAFETY_TASK, mantiveram previsibilidade em todos os cenários, garantindo a segurança do sistema. A tarefa SPD_CTRL também apresentou bom desempenho na maioria dos testes, com exceção do caso Deadline Monotonic, em que foi penalizada pela prioridade inferior. Já a tarefa SORT_ACT, de caráter mais próximo a soft real-time, foi a mais suscetível a perdas de deadline, evidenciando a necessidade de políticas customizadas ou ajustes de prioridade para reduzir atrasos em cenários de carga intensa.

De modo geral, a análise confirmou que a política customizada priorizando segurança associada ao escalonamento preemptivo representa a configuração mais equilibrada, garantindo a execução das funções críticas e mantendo as perdas em tarefas não críticas em níveis aceitáveis. Os resultados obtidos reforçam a importância da escolha adequada de políticas de escalonamento em sistemas de tempo real, principalmente em aplicações industriais onde a confiabilidade e a segurança são requisitos indispensáveis.

Tabela comparativa

Tema	Política	Preempção	Slicing	Freq (MHz)	Hard misses	Soft misses	Latência Touch (ms)	Conclusão
Esteira	RM	ON	ON	160	ENC=0 / SAFETY=0 / SPD=1	SORT=148	≤1	Hard OK; SORT com muitas perdas
Esteira	DM	ON	ON	160	ENC=0 / SAFETY=0 / SPD=4514	SORT=59	≤1	Hard comprometido (SPD), SORT melhora
Esteira	Custom	ON	ON	160	ENC=0 / SAFETY=0 / SPD=0	SORT=61	≤1	Hard OK; SORT reduz misses
Esteira	Preemptivo	ON	ON	80	ENC=0 / SAFETY=0 / SPD=0	SORT=392	≤1	Hard OK; SORT falha sob carga
Esteira	Preemptivo	ON	ON	160	ENC=0 / SAFETY=0 / SPD=0	SORT=405	≤1	Hard OK; SORT falha sob carga
Esteira	Preemptivo	ON	ON	240	ENC=0 / SAFETY=0 / SPD=0	SORT=424	≤1	Hard OK; SORT falha sob carga
Esteira	Cooperativo	OFF	–	80	ENC=0 / SAFETY=0 / SPD=1	SORT=461	≤1	Hard instável (SPD miss); SORT piora
Esteira	Cooperativo	OFF	–	160	ENC=0 / SAFETY=0 / SPD=0	SORT=405	≤1	Hard OK; SORT falha sob carga
Esteira	Cooperativo	OFF	–	240	ENC=0 / SAFETY=0 / SPD=0	SORT=332	≤1	Hard OK; SORT ainda falha