Controle e supervisão de um sistema de caldeira simulado

Breno Campos G.

***Resumo*—Neste artigo falarei sobre os resultados e a forma como foi feito um programa em tempo real, usando monitores, *threads* e *buffer* duplo para controle e supervisão de um simulador de caldeira.**

***Index Terms*—sistemas em tempo real, caldeira simulada, threads.**

I. Introdução

PARA o controle do simulador de caldeira, foram usados alguns conceitos de tempo real, como *threads* e monitores, que falaremos brevemente a seguir.

Como em linguagens de programação orientadas a objetos, temos modificadores de acesso como *private* e *public*, para definir quais módulos e classes podem acessar aqueles recursos. No entanto, na linguagem C não temos essas diretivas suportadas nativamente pela linguagem, com isso, somos obrigados a implementar via código. A ideia do monitor é tornar determinados recursos privativos as tarefas [1], como as variáveis de *mutex* que falaremos a seguir, além de restringir acesso a certas variáveis, todo o manipulamento delas é feita através de métodos, e caso esses métodos compartilhem recursos entre tarefas, haverá um sistema de exclusão mútua. **Exclusão mútua** ou *mutex*, é uma forma de evitar múltiplas tarefas acessando o mesmo recurso compartilhado, fazendo assim com que apenas uma tarefa modifique e leia os valores daquele recurso. Outro conceito usado foi o de *buffer* duplo, em que também era protegido por monitores, o motivo para usarmos ele era para que as tarefas de controle não tivessem um *delay* grande ao gravar os dados no disco, assim as tarefas apenas escreviam no *buffer* e havia outra tarefa responsável apenas por escrever esses dados no arquivo.

II. Metodologia

Para o controle do sistema, foi implementado uma tarefa para controle de temperatura do sistema e outra para o controle de altura do nível da água. A tarefa de controle de temperatura com período de 50 milissegundos e alterando os valores dos atuadores **Ni, Na, Q**. Já a tarefa de controle de altura da água, com período de 70 milissegundos, e alterando os valores dos atuadores **Nf, Na e Ni**, o Ni por ser alterado nas duas tarefas, tem uma prioridade maior na tarefa de controle de altura, sendo alterado com maiores valores para caso a tarefa de controle de temperatura tente abaixar. Há uma tarefa somente para capturar os valores dos sensores e armazenar no arquivo **sensor.c** através dos métodos do monitor, outra tarefa que irá imprimir os valores dos sensores na tela recuperando esses dados de métodos presentes no **sensor.c**. Outra tarefa para disparo de alarme caso a temperatura da caldeira esteja acima de 30 graus.

Assim que o sistema inicia, ele pede os valores de referência para altura e temperatura da caldeira, após isso o sistema inicia e começa a armazenar os valores dos sensores em um arquivo, e os valores dos tempos de resposta de cada tarefa de controle. Todas as tarefas foram implementadas usando ***threads***.

O primeiro teste foi feito usando **Ti = 10, No = 0, Ta = 10**, a temperatura estabilizada em 25 graus e altura da água em 2 metros. Na próxima seção veremos os resultados deste teste.

O segundo teste foi feito usando **Ti = 20 e Ta = 25** e para os primeiros 30 segundos **Na = 10**, após isso foi modificado para **No = 70**, e novamente após 30 segundos foi modificado para **No = 100** até o final da execução do programa. Na próxima seção veremos os resultados deste teste.

III. Resultados e discussões

No teste 1, como não tínhamos interferência do No tirando água da caldeira o controle é bem mais fácil e preciso como podemos ver pelos gráficos abaixo.

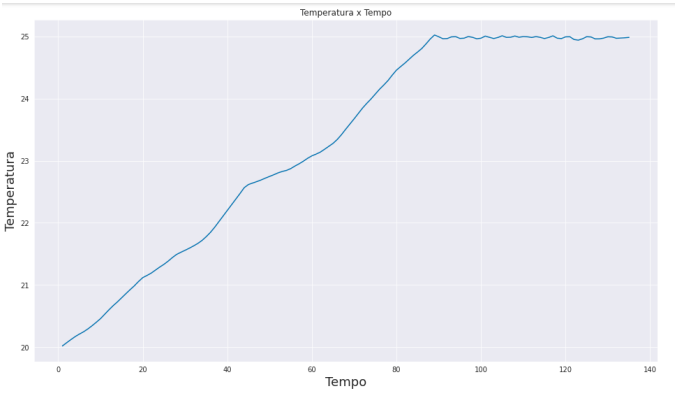

Figura 1. Gráfico com as medições de temperatura

Como podemos ver o controle da temperatura se estabilizou bem nos 25 graus desejados.

A. 1, Universidade Federal do Ceará, Sobral, Brasil e-mail: brenocg@alu.ufc.br

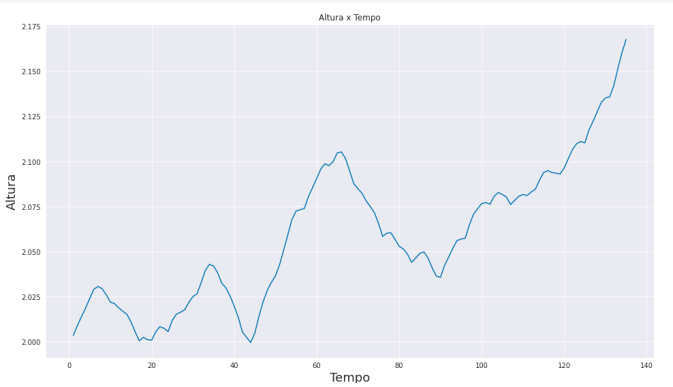

Figura 2. Gráfico com as medições de altura

Aqui podemos ver que a altura teve pouca variação, apenas de 2.0 a 2.175. Com isso podemos considerar que o sistema controlou bem o nível de água na caldeira.

Para o teste 2, tínhamos 3 estágios de mudança no **No**, com isso há uma grande perda de água na caldeira, portanto a dificuldade de controle aumenta. Para elevarmos o nível de água precisamos abrir a torneira de atuador indicada pelo **Ni**, contudo esta água entra a 20 graus, mas queremos manter o sistema a 29 graus precisamos alterar a resistência **Q** dentro do sistema para manter a temperatura subindo.

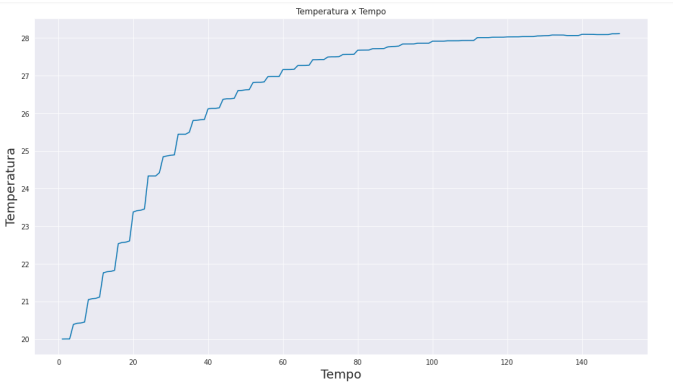

Figura 3. Gráfico com as medições de temperatura do teste 2

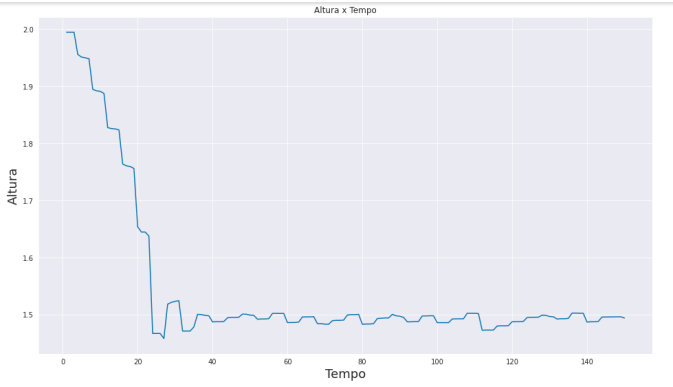

Figura 4. Gráfico com as medições de altura do teste 2

Podemos ver que o sistema controlou bem a altura, com uma variação maior que o primeiro teste, mas mesmo assim foi um bom controle. No entanto, na temperatura não conseguimos chegar na temperatura ideal de 29 graus, ficou por volta de 28,3 graus, pois depois que o **No** é *"setado"* em 100, o aumento de temperatura mesmo com o **Q** no máximo é quase impossível, o ideal seria chegar a temperatura desejada de 29 graus antes da pertubação **No** chegar em seu ápice.

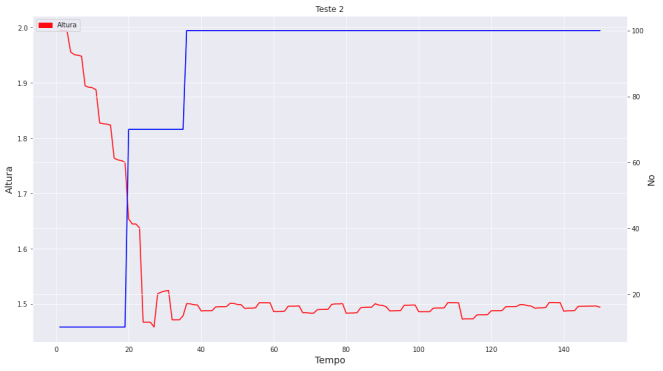

Figura 5. Gráfico com as medições de altura pelo No do teste 2

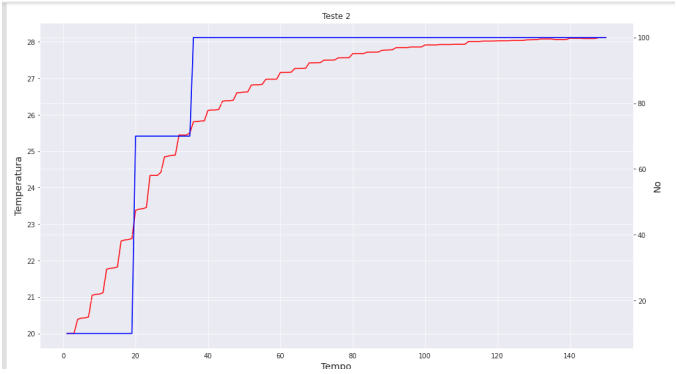

Figura 6. Gráfico com as medições de temperatura pelo No do teste 2

Nos gráficos acima, vemos a temperatura e altura, ambos em relação ao No.

Agora vamos analisar os tempos de resposta das tarefas, em relação aos seus ***deadlines***.

O tempo de resposta, em microssegundos, mínimo, médio e máximo da tarefa de temperatura, foram, respectivamente, **min: 33.000, med: 220.270, max: 3885.000**.

O tempo de resposta, em microssegundos, mínimo, médio e máximo da tarefa de altura, foram, respectivamente, **min: 26.000, med: 204.946, max: 5016.000**.

Com isto, olhando até para os maiores valores obtidos, vemos claramente que os ***deadlines***, de **70 ms** e **50 ms**, foram atendidos de forma bastante satisfatória. Isso se da por alguns motivos, dentre eles podemos destacar como, o *hardware* usado era bem potente para uma tarefa dessas, que normalmente seria executado por um microcontrolador com capacidade de processamento bem menor que uma CPU de uso pessoal, e também foi usado técnicas de programação próprias para sistemas em tempo real, como ***mutex*** e **monitores**

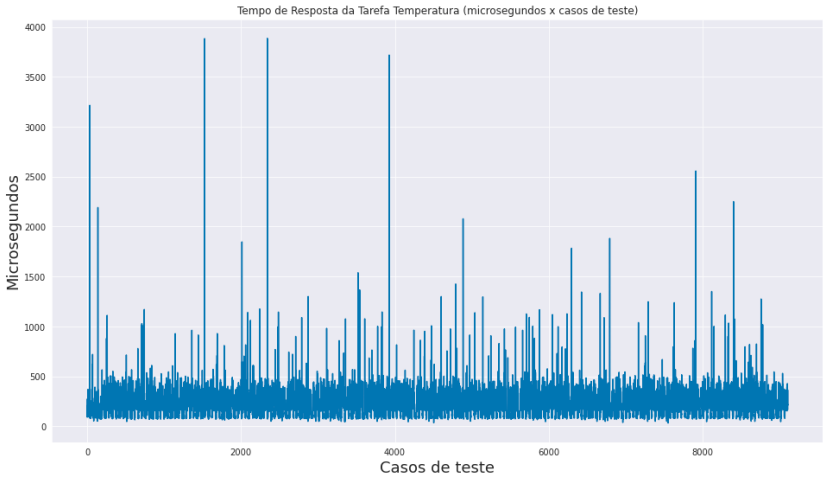

Figura 7. Gráfico com as medições dos tempos de resposta da tarefa de temperatura

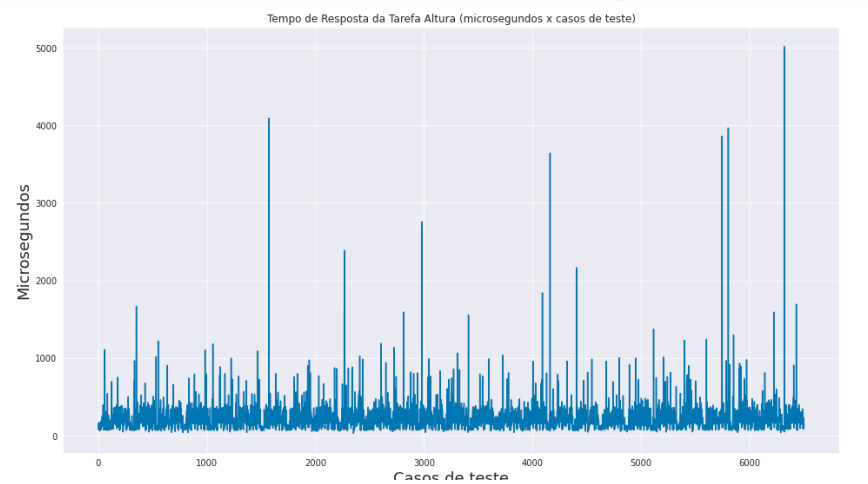

Figura 8. Gráfico com as medições dos tempos de resposta da tarefa de altura

Acima tempos os ***plots*** dos tempos de resposta de cada tarefa periódica de controle.

Como já mencionado anteriormente, os *deadlines* foram perfeitamente atendidos, sem nenhuma perda. Então podemos dizer que o sistema é **(20,20)-firme**, pois para qualquer janela de 20 ativações ele cumpre todos os *deadlines*. Para o **Fator Skip S**, não se aplica.

REFERÊNCIAS

[1] Oliveira, R. S. Fundamentos dos Sistemas de Tempo Real. Original registrado na Biblioteca Nacional. Primeira edição, revisão 3, outubro de 2018. ISBN-13: 9781728694047.