AVALIAÇÃO DE DESEMPENHO DE UM SISTEMA OPERACIONAL EM TEMPO REAL NA PLATAFORMA ARDUINO UNO

FELIPE CHABATURA NETO[1], MARCO AURÉLIO SPOHN[2]

1 Introdução/Justificativa

Atualmente, existe uma grande quantidade de sistemas que possuem requisitos em tempo real (WANG, 2017). Por exemplo, o controle das turbinas e dos motores de um avião depende de uma série de informações que são coletadas constantemente durante o voo. De maneira análoga, o sistema de acionamento do *airbag* de um carro precisa ser extremamente pontual para que possa ser benéfico ao motorista caso ocorra um acidente. Para suprir essas necessidades, existem os sistemas operacionais em tempo real (*Real Time Operating Systems,* RTOS) que oferecem suporte à execução de aplicações com restrições de tempo de resposta (WANG, 2017).

Quando se projeta uma aplicação em tempo real, torna-se necessário tomar decisões considerando todos os aspectos do *hardware* e do *software* empregados. Para isso, deve-se conhecer requisitos da aplicação tais como capacidade de processamento e armazenamento, a fim de não escolher dispositivos demasiadamente potentes ou que não sejam potentes o suficiente para que a aplicação cumpra os requisitos necessários, evitando assim o desperdício de tempo e recursos. Para tomar tal decisão, torna-se necessário que previamente tenha sido feita uma análise, com o intuito de descobrir as capacidades e os limites da combinação de *software* e *hardware* utilizados (WANG, 2017).

Este trabalho propõe a análise de componentes do sistema operacional em tempo real FreeRTOS, um *kernel* em tempo real gratuito e de código livre, na plataforma de *hardware* Arduino Uno, uma plataforma minimalista e de baixo custo. O *port* do FreeRTOS aqui analisado não pertence à distribuição oficial, considerando-se que essa versão foi elaborada à parte por Phillip Stevens e, posteriormente, disponibilizada em um repositório no *github* (STEVENS, 2018). A análise corrente se dá através do estudo dos componentes do FreeRTOS e suas implementações, bem como auxiliada pela elaboração de testes que exploram os limites

1 Acadêmico de Ciência da Computação, Universidade Federal da Fronteira Sul, *campus Chapecó,* **Bolsista** contato: felipechabat@gmail.com
2 Professor Associado do curso de Ciência da Computação, Universidade Federal da Fronteira Sul, **Orientador**.

ISSN 2526-205x

destes componentes no ambiente extremamente limitado que o Arduino Uno proporciona.

2 Objetivos

2.1 Objetivo Geral

O objetivo deste projeto consiste em analisar o desempenho e escalabilidade do sistema operacional em tempo real FreeRTOS na plataforma de *hardware* minimalista Arduino Uno, com o intuito de evidenciar suas capacidades e limitações.

2.2 Objetivos Específicos

- Medir e analisar a escalabilidade e desempenho dos recursos do FreeRTOS na plataforma de *hardware* Arduino Uno.
- Identificar recursos da versão completa do FreeRTOS que foram excluídos para que o *port* para o Arduino Uno fosse possível.
- Elaborar uma documentação que permita melhor entendimento da implementação do sistema operacional FreeRTOS.

3 Material e Métodos/Metodologia

Inicialmente, os recursos e características do FreeRTOS foram estudados para se obter melhor compreensão e detalhamento sobre seu funcionamento. Esse estudo foi baseado no guia de utilização do sistema operacional escrito por seu fundador, Richard Barry, denominado *"Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide"* (BARRY, 2016).

Após esse estudo, alguns testes iniciais foram executados para determinar valores base que são fundamentais para a compreensão da limitação do cenário do trabalho. Estes valores se referem a quantidade mínima de memória *flash* e memória RAM consumidas pela imagem de um código do FreeRTOS com funcionalidade mínima (compreendendo o código correspondente a apenas uma tarefa em *loop* e o escalonador).

Posteriormente, deu-se início a análise do código fonte do *port* do FreeRTOS para a plataforma Arduino UNO, com o objetivo de obter conhecimento técnico mais aprofundado sobre seu funcionamento. Durante essa análise, foi possível observar elementos da versão original do FreeRTOS que tiveram de ser suprimidos para que o *port* fosse possível. Além disso, durante essa análise, uma documentação foi elaborada, detalhando o funcionamento dos principais serviços e funções do FreeRTOS, assim como suas estruturas e requisitos de

ISSN 2526-205x

memória.

Após o estudo do código fonte do sistema operacional, testes foram realizados determinando limites para cada um dos recursos de acordo com parâmetros estipulados como, por exemplo, número máximo de tarefas que podem executar, simultaneamente, com um determinado tamanho de pilha.

Funcionalidades que não puderam ser testadas no período de vigência deste projeto, por conta de limitações de tempo, estão previstas para serem realizadas em um trabalho de conclusão de curso em Ciência da Computação na UFFS.

4 Resultados e Discussão

A partir da análise do referencial teórico, em especial sobre o FreeRTOS, observa-se que o sistema operacional possui vários recursos disponíveis para que o desenvolvedor implemente uma série de aplicações com uma variada gama de requisitos. Por consequência de seus fundamentos e de como foi projetado, o correto funcionamento das aplicações executadas no FreeRTOS dependem diretamente do planejamento por conta do desenvolvedor, uma vez que o sistema operacional oferece apenas os mecanismos e políticas (*e.g.*, tipo de escalonador e atribuição de prioridades às tarefas) previstas para o suporte às aplicações em tempo real.

Através dos primeiros testes, pode-se determinar que para executar um código do FreeRTOS no Arduino UNO são necessários, no mínimo, 7252 *bytes* de memória *flash* de um total de 32K *bytes,* e 165 *bytes* de memória RAM de um total de 2048 bytes disponíveis. Apesar desse limite não ser prático, uma vez que ele não corresponde a um código que executa uma determinada função ou processamento, ele é substancial para se compreender quão limitado é o escopo ao se utilizar a combinação de *software* e *hardware* em questão. Além disso, pode-se determinar que o TCB (*Task Control Block,* ou bloco de controle da tarefa) de cada tarefa ocupa 41 *bytes* de memória ao se utilizar as configurações padrões do RTOS e que o tamanho dos ponteiros é de 2 *bytes,* o suficiente para indexar todos os endereços de memória do Arduino UNO.

Ao testar os recursos de tarefas do sistema, constatou-se que, no máximo, pode-se executar sete tarefas simultaneamente. Além disso, foram obtidos os valores máximos das pilhas das tarefas, de acordo com a quantidade de tarefas simultâneas em execução.

O gerenciador de memória que é usado pelo *port* para o Arduino UNO é uma versão simplificada do gerenciador de alocação de memória dinâmica da linguagem C. Caso hajam

ISSN 2526-205x

múltiplas alocações e liberações de memória, além de gerar fragmentação na *heap,* o que pode afetar severamente o desempenho de novas alocações dinâmicas, cria-se um *overhead* de uso de memória. Isto acontece porque a estrutura utilizada para manter controle dos blocos de memória livres junto à *heap* é implementada de maneira implícita na própria *heap,* empregando 2 *bytes* para cada bloco de memória livre para indicar o tamanho do bloco e outros 2 *bytes* para manter a referência ao próximo bloco de memória livre, formando uma lista encadeada.

Resultados referentes aos demais componentes do sistema operacional poderão ser obtidos ao dar continuidade a essa pesquisa.

5 Conclusão

A partir dos resultados obtidos durante o projeto, foi possível observar o quão limitado é o desenvolvimento de aplicações em tempo real no ambiente de placas microcontroladoras e, em especial, no Arduino Uno. A observação destas limitações reforça a relevância da análise, uma vez que é necessário ter uma consciência precisa do quão limitado o ambiente é antes de se projetar e desenvolver uma aplicação.

É imprescindível a continuação da pesquisa, para que se possa através da análise dos códigos fontes dos recursos do sistema e da elaboração de testes para os mesmos chegar na totalidade dos resultados e obter uma análise conclusiva acerca do desempenho do FreeRTOS na plataforma de *hardware* Arduino Uno. A complementação desta pesquisa está prevista para um Trabalho de Conclusão de Curso em Ciência da Computação na UFFS.

Referências

BARRY, R. Mastering the FreeRTOS Real Time Kernel-a Hands On Tutorial Guide. Real Time Engineers Ltd, [S.l.], 2016.

WANG, K. C. Embedded and Real-Time Operating Systems, Springer International Publishing, 2017,

STEVENS, P. Arduino FreeRTOS Library. Acesso em: 18/06/2018, Disponível em: <https://github.com/feilipu/Arduino_FreeRTOS_Library>.

Palavras-chave: Sistemas operacionais em tempo real, avaliação de desempenho, plataformas de prototipagem de *hardware* livre, Arduino, FreeRTOS.

Financiamento PIBITI/CNPq e PIBITI/UFFS - 2017/2018