Desempenho de memória em RTOS comparado com SO de propósito gerais

Fernanda Pereira Guidotti / Verônica Vannini

Fernanda Pereira Guidotti / Verônica Vannini

Desempenho de memória em RTOS comparado com SO de propósito gerais

Monografia apresentada para a disciplina de Sistemas Operacionais. Instituto de Ciências Matemáticas e de Computação – ICMC-USP.

Área de Concentração: Ciências de Computação e Matemática Computacional

Professores: Prof. Dr. Julio César Estrella / Profa. Dra. Sarita Mazzini Bruschi

USP – São Carlos
Julho de 2018

RESUMO

GUIDOTTI, F. P.; VANNINI, V. **Desempenho de memória em RTOS comparado com SO de propósito gerais**. 2018. 54 f. Monografia (Monografia de S.O – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Sistemas embarcados críticos têm uma urgência em retorno de informações, neste contexto onde é necessário ter a resposta o mais rápido possível, validar se um sistema operacional de tempo real (RTOS) é pertinente em aplicações de sistemas embarcados é essencial. Essa proposta tem por objetivo comparar o uso de memória de um Sistema Operacional (SO) de propósito geral e um RTOS, com o intuito de identificar qual sistema operacional seria ideal para aplicações em sistemas embarcados. Os experimentos utilizaram uma Raspberry Pi, onde foi embarcado o RTOS Xenomai e o SO Raspian, foi realizado uma comparação de desempenho de memória entre eles. O RTOS teve melhor desempenho do que o SO testado, contudo é necessário estudos futuros para validar a real eficiência do RTOS. Com este trabalho, pretende-se contribuir com o desenvolvimento da pesquisa na área de sistemas embarcados e RTOS.

Palavras-chave: Sistemas Embarcados; Sistema operacional; RTOS; Gerenciamento de memória; Raspberry Pi.

ABSTRACT

GUIDOTTI, F. P.; VANNINI, V. **Desempenho de memória em RTOS comparado com SO de propósito gerais**. 2018. 54 f. Monografia (Monografia de S.O – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Critical embedded systems have an urgency to return information, in this context where it is necessary to have the answer as quickly as possible, to validate whether a real-time operating system (RTOS) is pertinent in embedded systems applications. The purpose of this proposal is to compare the memory usage of a General Purpose OS and an RTOS to identify which operating system would be ideal for embedded systems applications. The experiments used a Raspberry Pi, where the RTOS Xenomai and OS Raspian were embedded, a memory performance comparison between them was performed. RTOS performed better than the OS tested, however, further studies are needed to validate the actual efficiency of RTOS. This work intends to contribute to the development of research in the area of embedded systems and RTOS.

Key-words: Embedded systems; Operational system; RTOS; Memory management; Raspberry Pi.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DVS ..... *Dynamic Vol-tage Scaling*
FIFO ..... *First-In First-out*
RDV ..... Regulagem Dinâmica de Voltagem
RTOS .... sistema operacional de tempo real
SO ..... Sistema Operacional
VANT .... Veículo aéreo não tripulado

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

1.1 Contextualização

Os sistemas dedicados a missão em Veículo aéreo não tripulado (VANT) são considerados sistemas embarcados críticos pois colocam em risco equipamentos e recursos de alto custo e/ou vidas humanas. Pela urgência dessas aplicações é questionada a utilização de sistemas operacionais em tempo real ao invés de sistemas operacionais de propósitos gerais, como por exemplo, Linux, onde normalmente são realizados.

1.2 Motivação e Justificativa

Validar se um RTOS é pertinente para a aplicações em sistemas embarcados, relacionados a projetos de pesquisa com uso de VANTs. As crescentes pesquisas relacionadas na área, demonstram a necessidade de estudar as diferenças de desempenho entre estes SOs. Neste contexto, com o objetivo de obter a melhor performance em sistemas críticos, esse trabalho procura avaliar a hipótese que um RTOS tem melhor desempenho no gerenciamento de memória em relação a SO de propósito geral para aplicações embarcadas críticas.

1.3 Objetivo

Estudar e avaliar o gerenciamento de memória no contexto de um RTOS comparado a um SO de propósito geral, com a finalidade de validar a utilização destes em sistemas embarcados críticos.

Objetivos Específicos

Nesse contexto, o presente projeto estabeleceu como meta atingir os seguintes propósitos específicos:

1. Embarcar um SO de propósito geral;
2. Embarcar um RTOS;
3. Comparar o desempenho de uma aplicação *Memory Bound* nos SOs escolhidos;
4. Avaliar o desempenho do gerenciamento de memória nos SOs por meio de monitoramento do uso da memória.

1.4 Organização

A presente proposta está organizada da seguinte forma: o Capítulo 2 apresenta o referencial teórico abordando os principais conceitos necessários para este projeto, as principais ferramentas e aplicações que serão empregadas durante o desenvolvimento do projeto e os trabalhos relacionados. O Capítulo 3 apresenta a proposta de implementação e a metodologia de desenvolvimento. Por fim no Capítulo 4 é apresentado os resultados.

CAPÍTULO

2

REFERENCIAL TEÓRICO

2.1 Conceitos

Sistemas embarcados

Todo sistema computacional integrado a dispositivos com tarefas específicas são considerados sistemas embarcados. Estes tem como características um tempo longo de funcionamento ininterrupto, ambientes hostis, limitação de recursos como memória, processamento e consumo de energia, além de possuírem comunicação em baixo nível com outros dispositivos de hardware. Estes sistemas são considerados críticos quando podem colocar em risco vidas humanas ou instalações de alto valor, de acordo com Trindade *et al.* (2009), como é o caso de carros autônomos ou veículos aéreos não tripulados entre outros.

Sistemas operacionais

A definição de Sistemas operacionais possui duas principais visões de acordo com os principais autores da literatura (TANENBAUM, 1999; GAGNE, 2013; STALLINGS, 2004). São elas:

- ***Top-down:*** É descrita pela visão do usuário do sistema. Através da abstração do hardware e seus dispositivos, tornando-se um intermediário entre os programas (aplicativos) e o hardware (componentes físicos);
- ***Bottom-up:*** Nessa visão o sistema operacional atua como um gerenciador de recursos e aplicações, controlando programas (processos), memória, disco rígido e periféricos e como eles podem ser utilizados.

Sendo assim, um sistema operacional pode ser descrito como uma aplicação de grande

complexidade responsável pelo funcionamento total da máquina, permitindo a interação entre software e hardware.

RTOS - Real Time Operation System

Para que um SO seja considerado um RTOS é necessário que ele reaja a estímulos do ambiente em prazos específicos, definidos por requisitos temporais do comportamento do sistema, ou seja, o RTOS deve processar o resultado correto dentro do requisito temporal atribuído. Sendo assim um sistema em tempo real deve além de garantir tanto a integridade dos resultados quanto o tempo em que esses dados são apresentados (FARINES; FRAGA; OLIVEIRA, 2000).

Alguns dos RTOS mais conhecidos são:

- TI-RTOS
- X-Real Time kernel
- FreeRTOS
- QNX
- Zephyr
- μCOS
- Xenomai

Gerenciamento de memória

Como já descrito, uma das funções do SO é gerenciar os recursos disponíveis no sistema, entre eles a memória (TANENBAUM, 1999). É necessário que o SO organize os espaços livres e ocupados e tenha o controle da relação entre os processos e o espaço de memória por eles utilizados. Para isso, existem vários algoritmos de gerenciamento de memória, entre eles, um dos mais conhecidos é o *First-In First-out* (FIFO).

O gerenciamento de memória afeta diretamente no desempenho do sistema pois todos os processos, dos mais básicos ao mais supérfluos, concorrem por um espaço na memória tanto para sua execução como para seus dados. Sendo assim, é necessário avaliar o desempenho desta parte do sistema de um RTOS para validar a confiabilidade do mesmo.

2.2 Trabalhos Correlatos

Nesta seção são apresentados os trabalhos relacionados a pesquisa, sendo abordados estudos sobre SO, placas e aplicações. A Tabela 1 demonstra os trabalhos que são relacionados a esta pesquisa, fazendo um comparativo entre eles.

Tabela 1 – Trabalhos Relacionados.

Autor	RTOS	S.O	Placa	Aplicação	Desempenho	Memória
(NOVELLI *et al.*, 2005)	Sim			Regulagem dinâmica de voltagem	Sim	Não
(KOH; CHOI, 2013)	Sim	Xenomai/RTAI		Avaliação de desemprenho de RTOS	Sim	Não
(AL-THOBAITI *et al.*, 2014)	Não		Arduino Uno	Sistema de automação residencial sem fio em tempo real	Não	Não
(RAI; KUMAR, 2015)	Sim	Xenomai/RTAI		Estudo comparativo de RTOS e suas aplicações	Não	Sim
(NGUYEN *et al.*, 2015)	Não	Raspbian OS	Raspberry Pi Model B+	Sistema de Monitoramento	Sim	Não
(ONKAR; KARULE, 2016)	Não	Raspbian OS	Raspberry Pi 2 Model B	Manutenção para aplicação industrial	Não	Não
(SHAH; SHAH, 2016)	Sim			Algoritmos de Gerenciamento de memória para RTOS	Sim	Sim
(DOCEKAL; SLANINA, 2017)	Sim	FreeRTOS	Raspberry Pi 3	Sistema de controle para dados de aquisição e distribuição na robótica swarm	Não	Não
Proposta de Pesquisa	Sim	Raspbian OS/Xenomai	Raspberry Pi Model B+	Cálculo de matrizes para previsão de tempo no Brasil	Sim	Sim

No trabalho de Novelli *et al.* (2005) é apresentado um estudo para a economia de energia em sistemas de tempo real, os autores focam no processador, através da técnica conhecida como Regulagem Dinâmica de Voltagem (RDV) ou do ingês *Dynamic Vol-tage Scaling* (DVS) em conjunto com um algoritmo para determinação de folgas de processamento. Utilizam escalonadores do tipo Taxa Monotônica do ingês *Rate Monotonic*. O trabalho é apresentado por duas fases, na primeira fase os autores definem um conjunto de tarefas, em que o objetivo é escolher a menor frequência possível de operação e garantir as restrições de tempo. Na segunda fase aproveitam o tempo adicional e a folga remanescente para reduzir a frequência de operação, fizeram simulações e obtiveram resultados positivos na redução do consumo de energia. Novelli *et al.* (2005) foca no processador, enquanto esta proposta de trabalho tem por objetivo fazer uma comparação entre o gerenciamento de memórias em RTOS e SO de propósito geral, embora seja obtidos dados do processamento também.

Em Koh e Choi (2013) os autores abordam sobre o uso de RTOS em sistemas embarcados, onde há a necessidade de respostas exatas em um determinado período de tempo. Mesmo assim, muitos sistemas embarcados acabam utilizando sistemas de propósito geral pelo alto custo de desenvolvimento, baixa flexibilidade de desenvolvimento, indisponibilidade de documentação detalhada e a falta de suporte técnico. No artigo é feita a análise temporal dos sistemas RTOS RTAI e Xenomai em chamadas de tempo real, onde é descrito testes no modo kernel e no modo usuário de quatro estruturas, sendo elas: processos periódicos, semáforos, FIFO em tempo real e Mailbox/Message Queue. Ao comparar os dois sistemas operacionais, apenas no teste do FIFO o sistema RTAI foi superior ao Xenomai e em todos os casos os testes em modo kernel foram mais eficiente do que os em modo usuário. Na Figura 1 é ilustrado o diagrama das tarefas das avaliações e na Figura 2 pode-se visualizar os resultados de acordo com período de testes variando entre 10, 30 e 50 ms, tanto para o modo kernel quanto para o modo usuário.

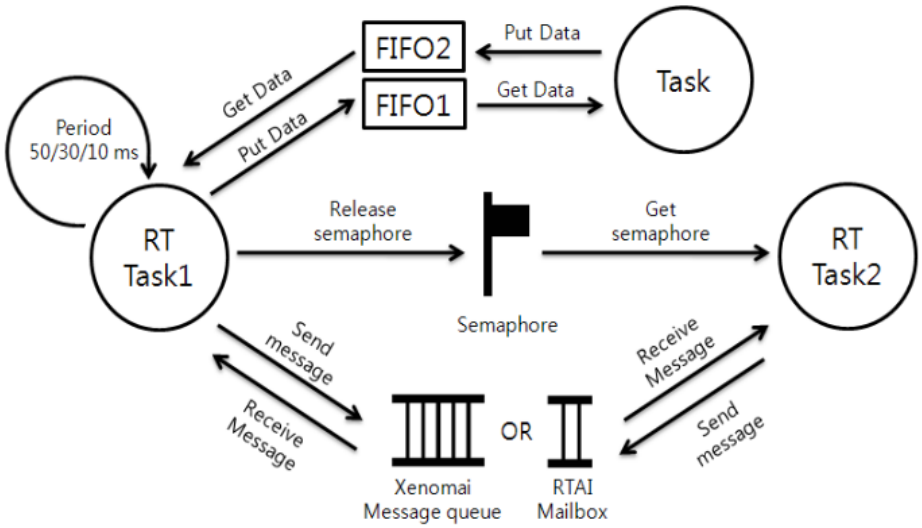

Figura 1 – Diagrama das tarefas de avaliações (KOH; CHOI, 2013).

Este trabalho faz uma comparação entre o RTOS e SO de propósito geral, enquanto o trabalho de Koh e Choi (2013) compara dois RTOS, os trabalhos se assemelham ao utilizar métricas de avaliações parecidas.

RTOS	SPACE	Period [ms]	Periodic Task [ns]	Semaphore [ns]	FIFO [ns]	Mailbox & Message queue [ns]
RTAI	Kernel	10	3837	1722	443	5771
		30	6146	2002	481	6482
		50	8170	2253	533	7020
	User	10	6034	7207	981	10550
		30	9068	8037	1051	11525
		50	10696	8145	1168	11717
Xenomai	Kernel	10	4840	1161	461	3445
		30	6541	1401	522	4014
		50	6635	1558	592	4120
	User	10	6381	2420	1766	6362
		30	7159	2520	1876	6696
		50	9883	2605	1913	6724

Figura 2 – Resultados experimentais de acordo com vários períodos (KOH; CHOI, 2013).

Al-thobaiti *et al.* (2014) propõem uma sistema de automação residencial em tempo real utilizando o Arduino Uno. O sistema proposto tem dois modos operacionais, o modo manual e o modo automatizado. No primeiro modo o usuário pode monitorar e controlar seus eletrodomésticos usando um *smartphone* utilizando de conexões Wi-Fi, enquanto no segundo modo os controladores são capazes de monitorar e controlar diferentes aparelhos na casa automaticamente. Os autores implementaram um hardware com interface Matlab-GUI. A Figura 3 ilustra o fluxograma do processo de automatização, tanto no modo manual quanto no modo automático.

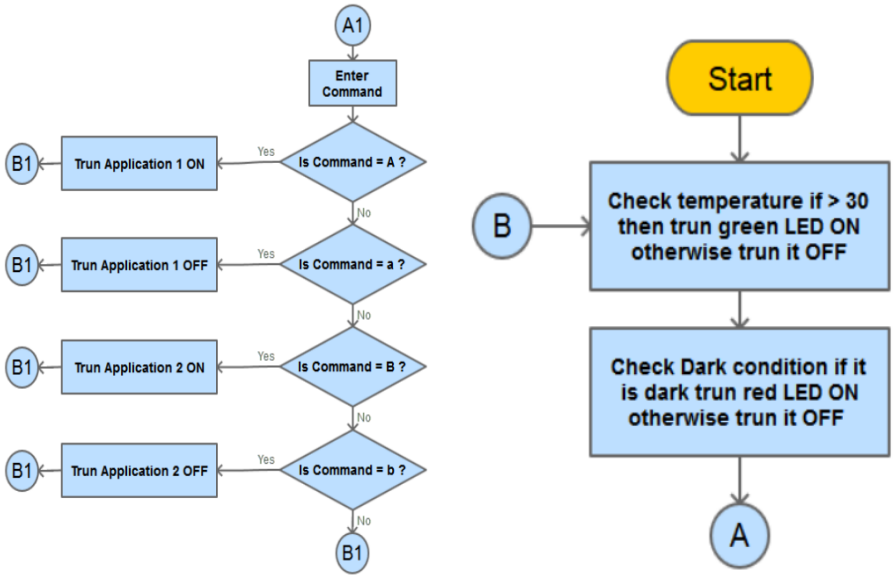

Figura 3 – Fluxograma de algoritmo manual e automático para controle de eletrodomésticos (AL-THOBAITI *et al.*, 2014).

A Figura 4 ilustra a arquitetura do sistema de automação residencial proposta pelos autores Al-thobaiti *et al.* (2014).

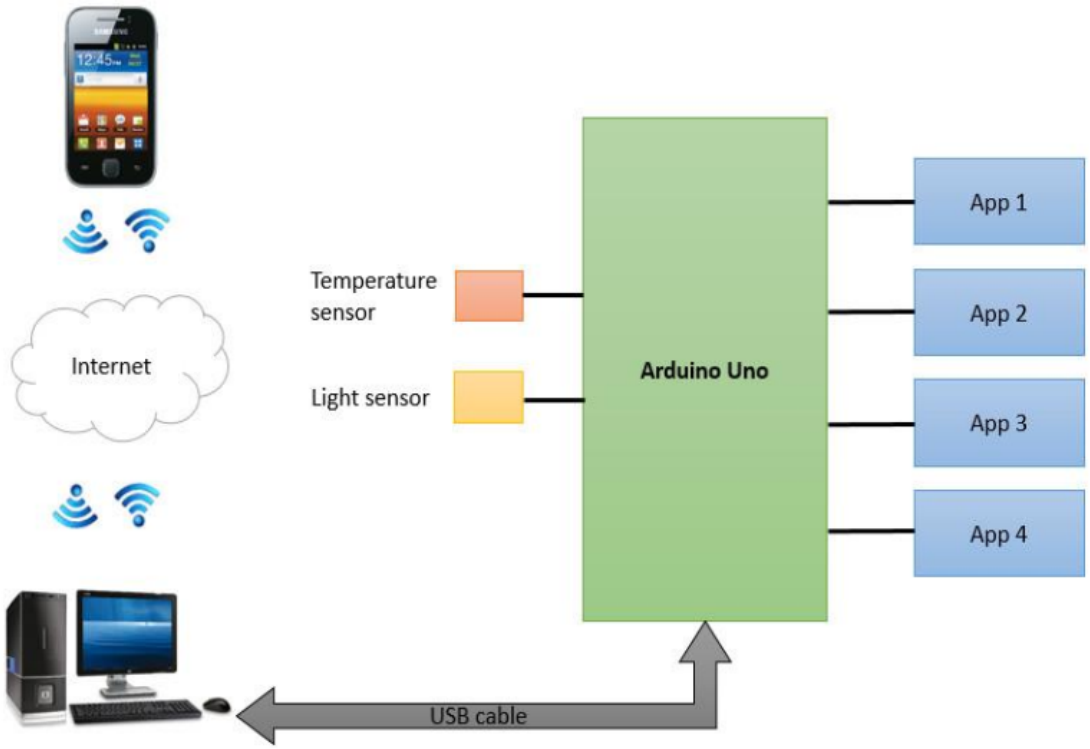

Figura 4 – Arquitetura do sistema de automação residencial(AL-THOBAITI *et al.*, 2014).

Enquanto os autores Al-thobaiti *et al.* (2014) utilizam Arduino para monitorar um sistema em tempo real, o trabalho proposto utiliza uma Raspberry Pi com um RTOS.

Em (RAI; KUMAR, 2015), os autores fazem algumas comparações entre estruturas dos sistemas RTAI e Xenomai. O artigo apresenta as estruturas de *Multitasking and scheduling*, sincronização, gerenciamento de interrupção, comunicação entre processos e gerenciamento de memória. O artigo apresenta que o sistema Xenomai possui as configurações de Memória Virtual e alocação dinâmica. A presente proposta estuda o Xenomai para analisar o tempo de memória gasto na execução de uma aplicação *memory bound*.

Nguyen *et al.* (2015) abordam um estudo sobre sistema de monitoramento de vigilância de baixo custo baseado em Raspberry Pi, utilizando um algoritmo em Python de detecção de movimento para diminuir o uso de armazenamento e economizar custos de investimento, além de permitir câmera de transmissão ao vivo juntamente com a detecção de movimento em que pode ser visualizada a partir de qualquer navegador, sistemas móveis em tempo real. A Figura 5 ilustra a arquitetura do sistema de monitoramento. A câmara de vídeo é conectada à placa Raspberry Pi, o algoritmo entregará automaticamente o fluxo de dados de vídeo para o servidor em nuvem. Os usuários poderão assistir ao vídeo em qualquer dispositivo que tenha um navegador da web.

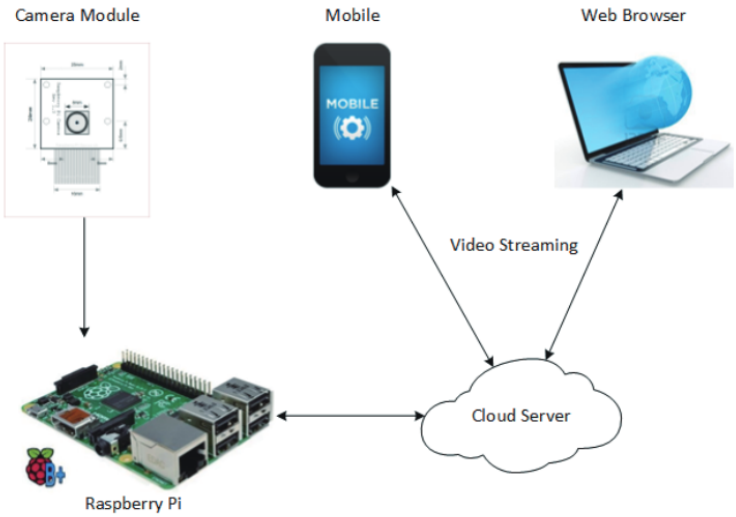

Figura 5 – Arquitetura do sistema de monitoramento e vigilância (NGUYEN *et al.*, 2015).

A detecção do movimento funciona com base nos quadros, é feito uma comparação em que exista uma mudança do estado do quadro. A Figura 6 ilustra o fluxograma do algoritmo de detecção de movimento, se há detecção de movimento é salvo o video.

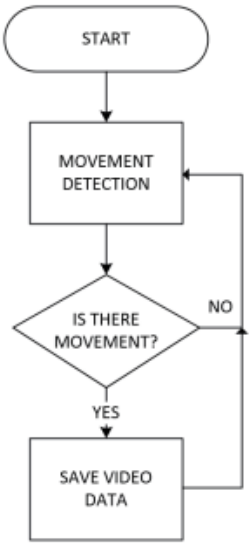

Figura 6 – Fluxograma do algoritmo de detecção de movimento (NGUYEN *et al.*, 2015).

O projeto de Nguyen *et al.* (2015) faz uma comparação do espaço de armazenamento usando o algoritmo de detecção de movimento e sem usar o algoritmo. O espaço de armazenamento ao usar a detecção de movimento é de 236 GB/mês, e sem o uso do algoritmo é de 740 GB / mês. O algoritmo de detecção de movimento auxilia para minimizar a capacidade de armazenamento de dados gravados.

O trabalho proposto nessa pesquisa, usa uma placa Raspberry Pi, porém não é previsto o uso de algoritmos para melhorar a capacidade de armazenamento, e sim fazer uma comparação de uma determinada aplicação *memory bound* para viabilizar o uso de sistemas em tempo real, a fim de que possa contribuir para um melhor desempenho.

Onkar e Karule (2016) aborda um sistema para manutenção industrial, os autores descrevem um método para controlar e fazer manutenção de máquinas, com o intuito de aumentar a vida útil da máquina, assim melhorar o processo de produção na indústria. Utilizam Raspberry Pi 2 Model B e um algoritmo em python.

A Figura 7 ilustra a arquitetura do sistema que os autores propuseram, os principais componentes são o sensoriamento de temperatura, proteção contra surtos de tensão, interruptor de partida da máquina, interruptor de início de trabalho na entrada e indicador LED, visor LCD *Touch Screen*, módulo Wi-Fi e campainha. Com o sensoriamento da temperatura e voltagem é possível proteger melhor o sistema, pois se a temperatura ou a voltagem aumentar, pode-se tomar a decisão de desarmar o sistema. A chave de partida da máquina e a chave de início da tarefa ajudam a tirar a duração do tempo para a mesma. O LED é usado como indicador. A exibição na tela de toque é usada para auxiliar o usuário do sistema. O Wi-Fi é usado para a conectividade com a Internet. O relé aciona a máquina. Na Figura 8 pode observar o protótipo do hardware do sistema.

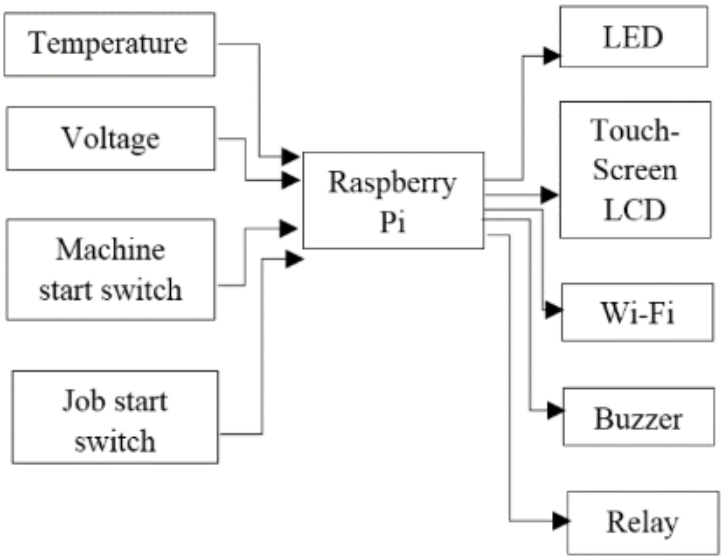

Figura 7 – Arquitetura do sistema para aplicações industriais (ONKAR; KARULE, 2016).

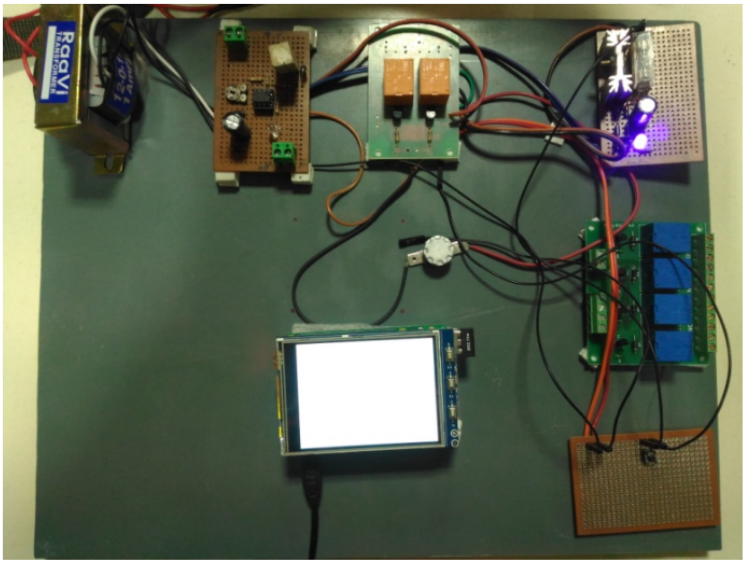

Figura 8 – Protótipo do hardware do sistema (ONKAR; KARULE, 2016).

A Figura 9 mostra o algoritmo de manutenção usado pelo sistema que trabalha com restrições temporais. O algoritmo é aplicado em tempo real. As etapas do algoritmo são:

1. Verificar o valor anterior sem contagem de carga;
2. Verificar se há manutenção;
3. Caso exista manutenção, fazer a manutenção;
4. Salvar o valor;
5. Enviar os dados online para o site.

O registro de manutenção do trabalho é feito diariamente e enviado no site para conectá-lo ao proprietário ou à equipe de manutenção. O sistema também fornece proteção contra sobretensão e proteção de temperatura para máquinas pequenas. A Figura 9 ilustra o fluxograma do funcionamento da manutenção. A Figura 10 exemplifica um cronograma de manutenção. A proposta deste trabalho utiliza uma placa Raspberry Pi, porém estuda o uso da memória em sistemas operacionais, e, não tem sensores e atuadores.

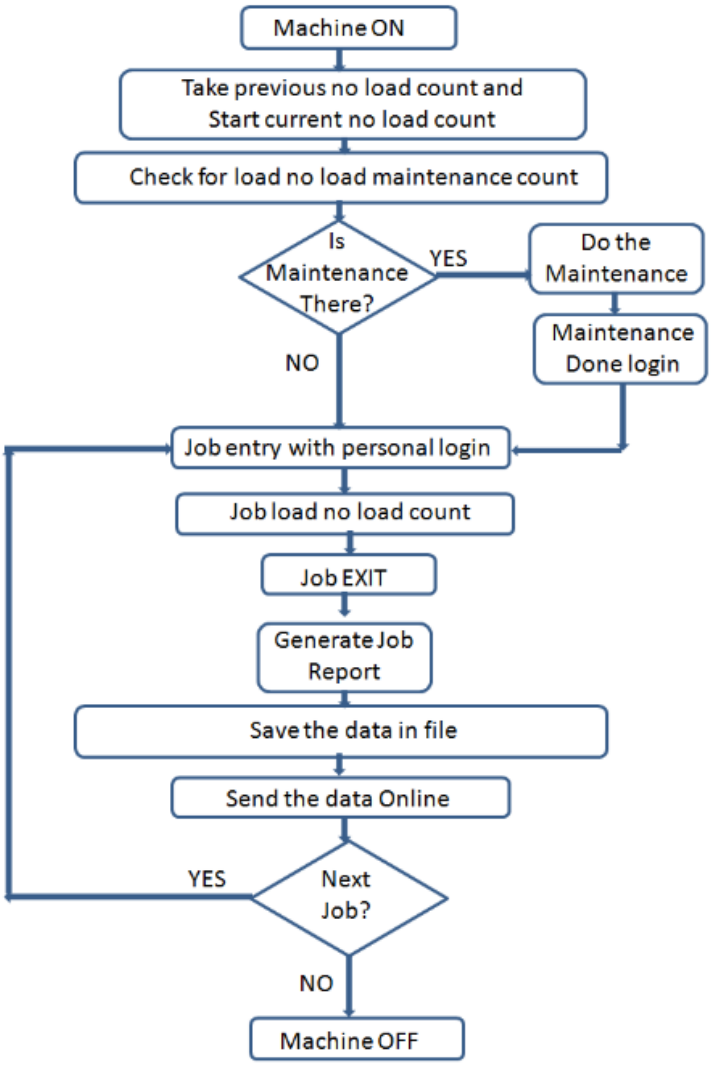

Figura 9 – Fluxograma do funcionamento de manutenção (ONKAR; KARULE, 2016).

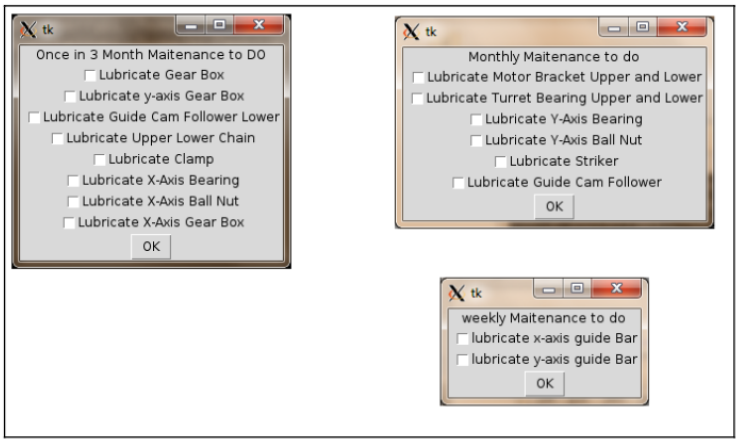

Figura 10 – Exemplo de cronograma de manutenção (ONKAR; KARULE, 2016).

Existem vários algoritmos de gerenciamento de memória que podem ser utilizados tanto em SO de propósito geral como em RTOS. Em Shah e Shah (2016), os autores descrevem

alguns dos algoritmos convencionais e não convencionais mais utilizados em RTOS de alocação dinâmica. Entre os convencionais os autores descreve sobre a forma de alocação dos algoritmos: Sequenciais (*First-Fit*, *Next-Fit*, *Best-Fit* e *Worts-Fit*), Indexados, e com utilização de *Bitmap*. Entre os algoritmos não convencionais: *Two level segregated algorithm (TLSF)*, *Improved two level segregated algorithm* e *Smart memory allocator algorithm*. Ao comparar os algoritmos em termos de fragmentação, uso de memória e tempo de resposta os algoritmos *TLSF-I* e *Smart memory allocator algorithm* têm melhor desempenho do que os outros. Os autores comparam o tempo de resposta do algoritmo enquanto a presente proposta compara o tempo de execução de uma aplicação tanto no RTOS quanto no SO de propósito geral.

Docekal e Slanina (2017) utiliza um sistema operacional em tempo real, o FreeRTOs, junto a uma placa Raspberry Pi na plataforma de robótica de enxame. A principal tarefa do sistema de controle realizado é garantir a comunicação com os diferentes tipos de sensores e preparar informações obtidas para o uso posterior. A vantagem do uso do sistema operacional FreeRTOS é a possibilidade de trabalhar com threads e pseudo-paralelismo no microcontrolador. Isso fornece a possibilidade de separar toda a atividade em tarefas individuais com prioridades atribuídas. Na Figura 11 observa-se o protótipo da plataforma robótica.

Figura 11 – Protótipo da plataforma robótica(DOCEKAL; SLANINA, 2017).

Os autores Docekal e Slanina (2017) utilizaram comunicação *Bluetooth*, assim conectou o microcontrolador com o programa pelo celular ou computador. Foi projetado um aplicativo para *smartphones* para a visualização dos dados e controle do robô. A Figura 12 exemplifica o controle da aplicação *mobile*, exibe o nível da bateria e as distâncias dos sensores ultrassônicos, é possível controlar a velocidade e a direção do robô e ainda ter a decisão em que o robô para antes do obstáculo sem colidir com ele. O trabalho proposto utiliza o Xenomai com a placa Raspberry Pi, para determinar se é valido utilizar um sistema em tempo real para aplicações embarcadas.

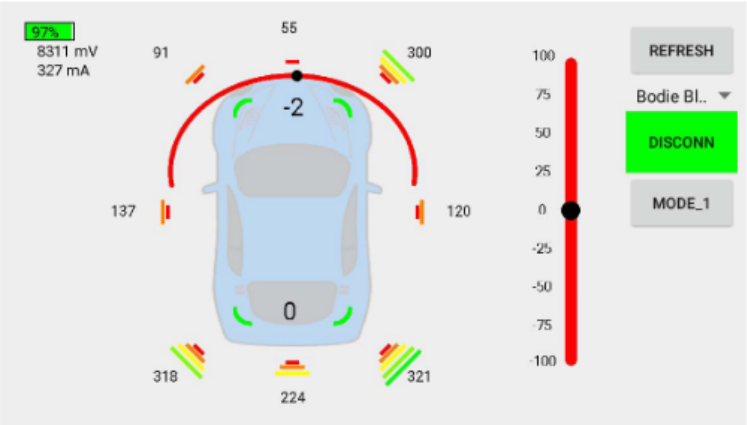

Figura 12 – Controle do protótipo da aplicação para Smartphone (DOCEKAL; SLANINA, 2017).

CAPÍTULO

3

MATERIAL E MÉTODOS

Este capítulo fornece uma visão geral sobre o gerenciamento de memória em sistemas operacionais. Na seção 3.1 apresenta uma visão sobre a caracterização da pesquisa , na seção 3.2 é discutido a metodologia utilizada neste trabalho e na seção 3.3 é apresentada a descrição da proposta.

3.1 Caracterização da Pesquisa

Segundo Wazlawick (2017), a pesquisa pode ser classificada de acordo com sua natureza, objetivos ou procedimentos técnicos. Com relação a sua natureza a pesquisa é classificada como trabalho original, com relação aos fins ou objetivos a pesquisa é classificada como exploratória. Referente aos procedimentos técnicos, o trabalho realizará uma pesquisa experimental.

3.2 Metodologia

A metodologia para desenvolvimento e avaliação desta proposta de trabalho será composta pelas fases especificadas abaixo:

1. **Estudo das tecnologias**: Nesta etapa foram estudadas as tecnologias utilizadas no projeto, incluindo a aplicação para testes.

2. **Seleção de parâmetros e métricas:** Nesta etapa foi definidos, os parâmetros e métricas que foram selecionadas para possíveis comparações com trabalhos semelhantes, com o objetivo de avaliar o desempenho da memória.

3. **Preparação e configuração do SO:** Nesta etapa foi embarcado o SO de propósito geral e o RTOS na placa Raspberry Pi Model B, para uma futura comparação entre elas.

4. **Testes:** Nesta etapa foi realizado testes para a validação da hipótese.

5. **Planejamento de experimentos:** Nesta etapa foi realizados o planejamento e execução dos experimentos com a finalidade de avaliar com base nas métricas e parâmetros definidos.

6. **Análise dos dados:** Nesta etapa foi analisados os dados produzidos na fase anterior utilizando a ferramenta BioEstat[1], para análise estatística.

A Figura 13 apresenta um fluxograma simplificado das etapas da metodologia a serem seguidas no desenvolvimento deste trabalho.

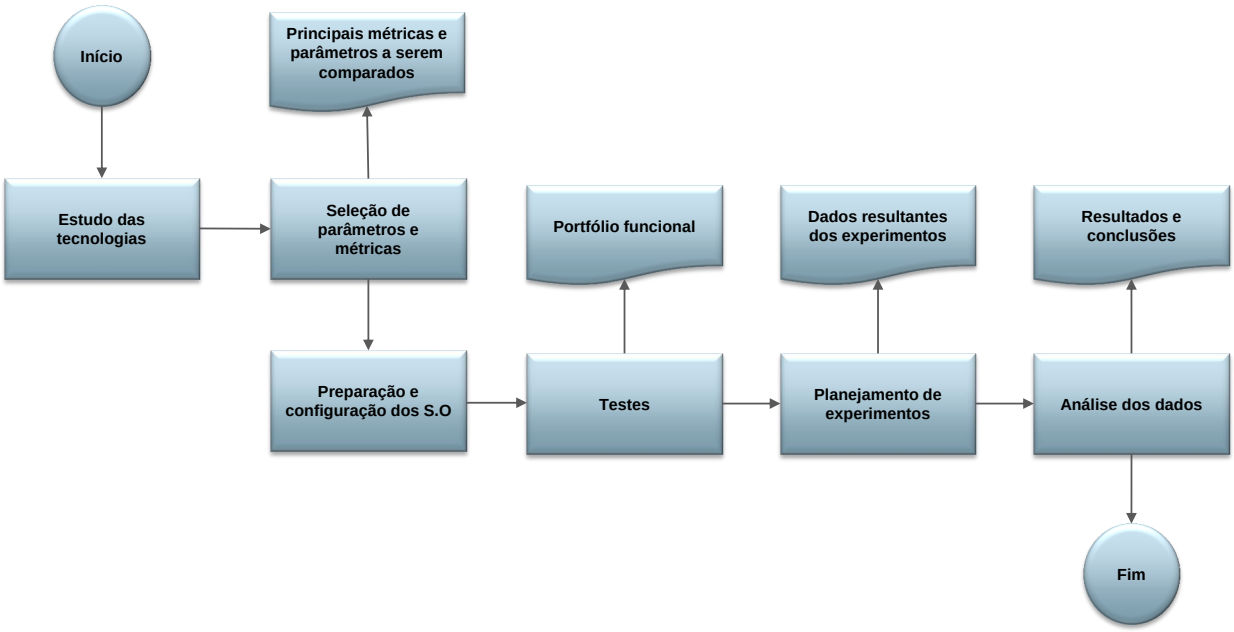

Figura 13 – Fluxograma simplificado da metodologia

3.3 Descrição da proposta

A arquitetura proposta neste projeto é ilustrada na Figura 14. Através de uma interface foi acessado a Raspberry Pi 1, onde foi embarcado os SO de propósito geral e o RTOS, posteriormente foi executada parte de uma aplicação *memory bound*, que foi desenvolvida e disponibilizada pelos autores Pereira e Marques (2016). Esta aplicação em tempo real realiza cálculos de matrizes para determinar a previsão do tempo no Brasil, o BRAMS[2], sigla em inglês para Desenvolvimento Brasileiro sobre o Sistema Brasileiro Regional de Modelagem Atmosférica, que é um sistema de modelagem numérica projetado para previsão e pesquisa

[1] https://www.mamiraua.org.br/
[2] http://brams.cptec.inpe.br/

atmosférica em escala regional, com foco em química atmosférica, qualidade do ar e ciclos biogeoquímicos. Foi monitorado a memória enquanto a aplicação estava em execução para testar e analisar o uso de memória no RTOS e SO de propósito geral. A aplicação foi cedida apenas para experimentos neste projeto.

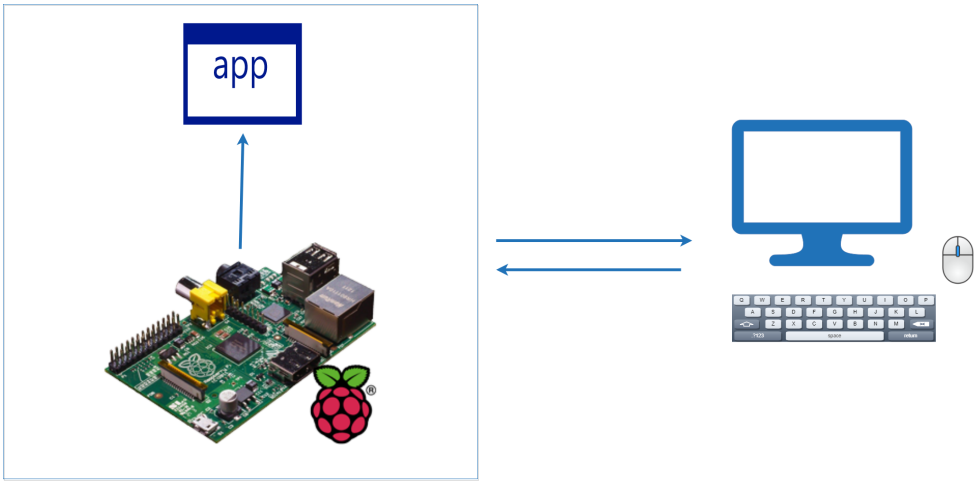

Figura 14 – Arquitetura da implementação da aplicação

Descrição dos Experimentos

Todos os experimentos foram realizado nas placa Raspberry Pi 1 modelo B que tem como especificação:

- **Chip:** Broadcom BCM2708
- **Tipo Core:** ARM1176JZF-S
- **Nº Core:** 1
- **Clock CPU:** 700 MHz
- **GPU:** VideoCore IV
- **RAM:** 256 MB

SO de propósito geral

Foi utilizado o Raspian, SO padrão dos sistemas Raspberry Pi basedo em sistemas Linux Debian. Este SO foi escolhido tanto pela facilidade de encontra-lo como pela grande utilização dele na comunidade desenvolvedora. Sendo assim, o processo de embarcar o sistema foi simples e rápido seguindo os passos do site oficial da Raspberry Pi.[3]

[3] https://www.raspberrypi.org/

Para o experimento principal com a aplicação *memory bound* foi apenas necessário carregar a aplicação na placa, já que o compilador necessário (gcc) é nativo do sistema. O monitoramento do uso de memória e de CPU foi realizado com o comando **top**, concomitante a execução do programa, através da linha:

```
top -d 1 -b > top_logfileXX.txt
```

Onde XX indica o número do teste realizado.

Foi escolhido um sistema mais simples de monitoramento para padronizar a coleta de dados entre os sistemas pela falta de ferramentas no RTOS.

RTOS

Como sistemas RTOS são mais específicos, e cada sistema suporta apenas um conjunto de arquitetura de hardware, foi necessário testar quatro sistemas na Raspberry Pi antes de realizar os testes com a aplicação *memory bound*. Os sistemas testados foram:

- FreeRTOS[4];
- BitThunder[5];
- ChibiOS[6];
- Xenomai[7].

O FreeRTOS foi o primeiro a ser considerado pela sua ampla utilização em sistemas embarcados. Porém os desenvolvedores oficiais não suportam a placa escolhida para este projeto. Algumas versões modificadas para a arquitetura em questão foram encontradas no GitHub, mas por falta de documentação nenhuma delas funcionaram na placa.

[4] https://www.freertos.org/
[5] https://github.com/jameswalmsley/bitthunder/
[6] http://www.stevebate.net/chibios-rpi/GettingStarted.html
[7] https://gitlab.denx.de/Xenomai/

Com o sistema BitThunder tivemos o problema de incompatibilidade entre o código e o compilador do kernel. Impossibilitando a utilização deste sistema.

O ChibiOS foi compilado e embarcado com sucesso para a placa. Porem além do sistema apenas possuir o modo kernel, é necessário obter um módulo HDMI/Serial para poder usar o shell disponibilizado pelo sistema com o auxilio de um monitor. Com a falta deste módulo, o sistema foi descartado.

O sistema utilizado foi o Xenomai. Que é caracterizado por ser um RTOS de propósito geral. Facilitando o processo de embarcar o sistema na Raspberry Pi pois já existe uma imagem especifica para a placa disponível[8]. Outra motivação para a escolha desse sistema foi a maior documentação online do sistema como também a disponibilidade de artigos avaliando o RTOS.

Assim como no SO de propósito geral, para o experimento principal com a aplicação *memory bound* foi necessário carregar a aplicação na placa, já que o compilador necessário (gcc) também é nativo do RTOS em questão.

Como já dito anteriormente, o monitoramento do uso de memória e CPU foi realizado com o comando top concomitante a execução do programa.

Como o RTOS é mais simples em questões gráficas e apenas um shell de comando é apresentado na tela, foi realizado uma conexão ssh com a placa para rodar a aplicação *memory bound* concomitante com a aplicação de monitoramento. Também foi necessário salvar os arquivos de log do monitoramento direto no computador, pois a memória da Raspberry Pi com o RTOS fica limitada e não era possível salvar o arquivo por conta do seu tamanho, isso é realizado pela linha de comando:

```
top -d 1 -b | ssh USER_PC@IP_PC "cat > top_logfileXX.txt"
```

Onde *XX* indica o número do teste realizado, *USER_PC* é o usuário do computador destino e *IP_PC* é o seu IP.

[8] http://www.cs.ru.nl/lab/xenomai/raspberrypi.html

CAPÍTULO

4

CONCLUSÃO

4.1 Resultados

Para a coleta de dados, foi realizado cinquenta execuções da aplicação em cada sistema operacional.

Na Figura 15 é apresentado as médias dos dados de uso de CPU e de memória no RTOS, assim como na Figura 16 é apresentada do SO de proposito geral.

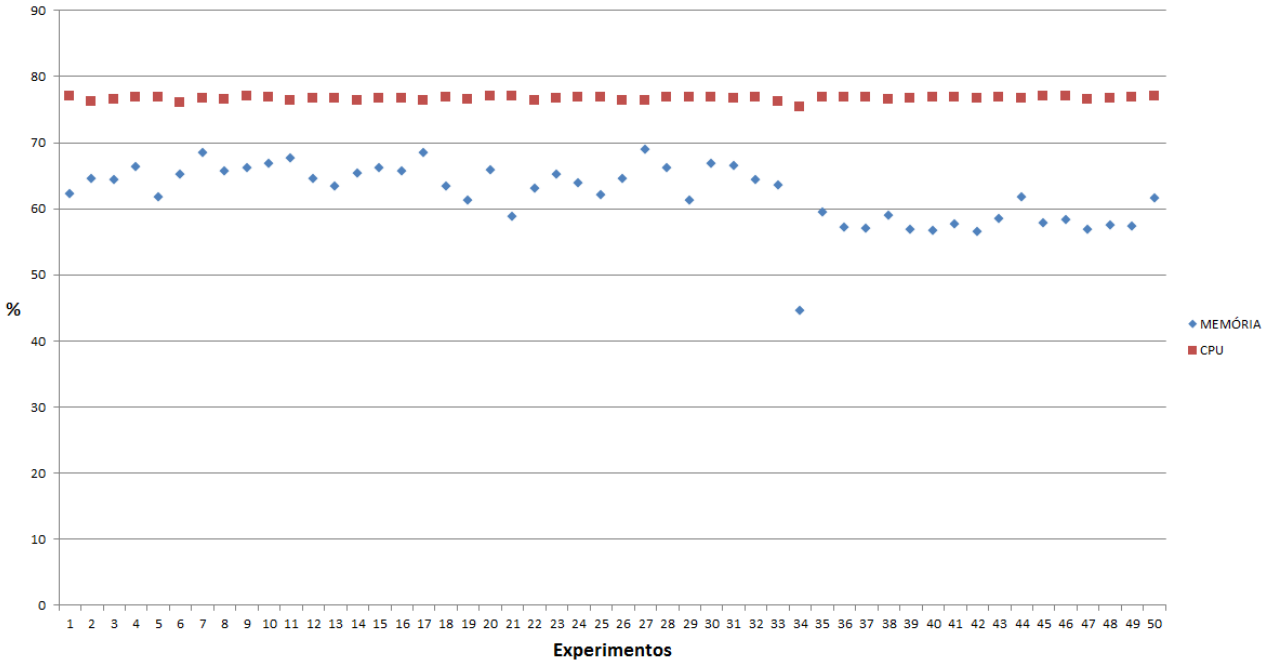

Figura 15 – Uso de CPU e Memória no RTOS

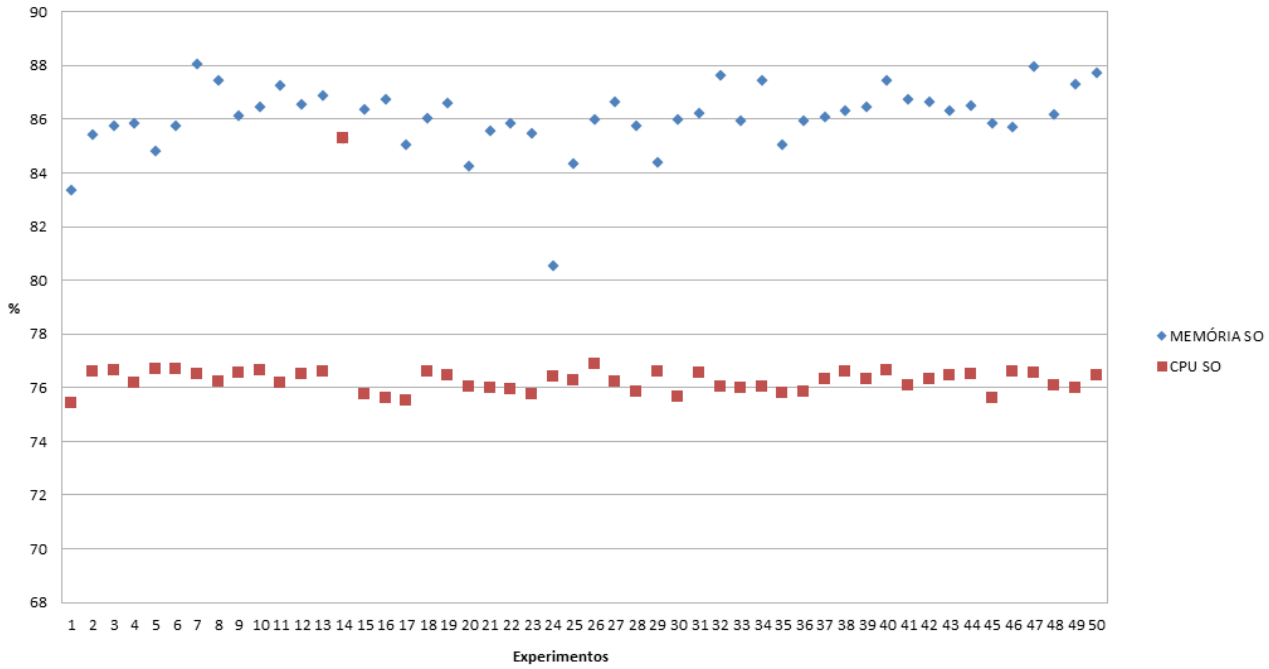

Figura 16 – Uso de CPU e Memória no SO

Para um melhor efeito de comparação considerando que os experimentos não possuem correlação um com o outro e que a ordem de execução não faz diferença, os dados na Figura 17 e na Figura 18, foram ordenados de forma crescente, com a finalidade de comparar o melhor e pior caso de cada sistema.

Na Figura 17, é possível analisar que mesmo com uma taxa de erro de 5% a porcentagem de utilização de memória no RTOS é inferior ao do SO de propósito geral. Indicando um melhor desempenho de utilização da memória. Já na Figura 18, podemos observar que, excluindo uma única discrepância nos testes do SO, os dados são muito semelhantes. Assim podemos indicar que os dados da Figura 15 não são inconsistentes com uma aplicação *memory bound*, mas que a utilização da memória no RTOS tem menor custo que no SO de propósito geral.

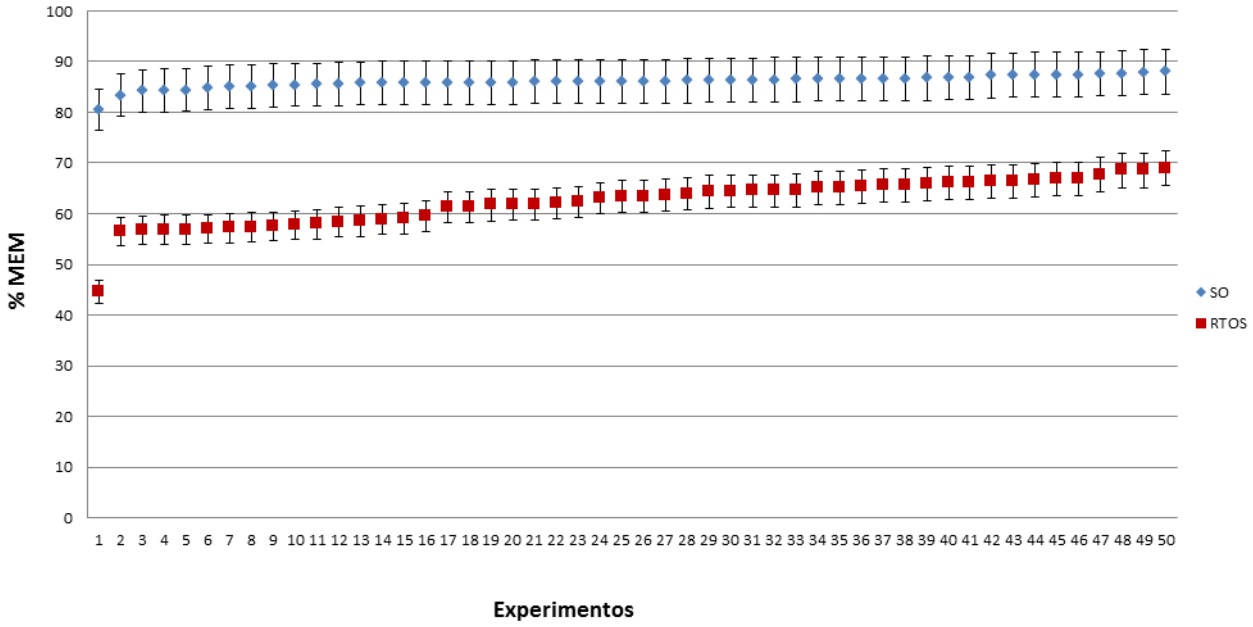

Figura 17 – Comparação de uso de memória

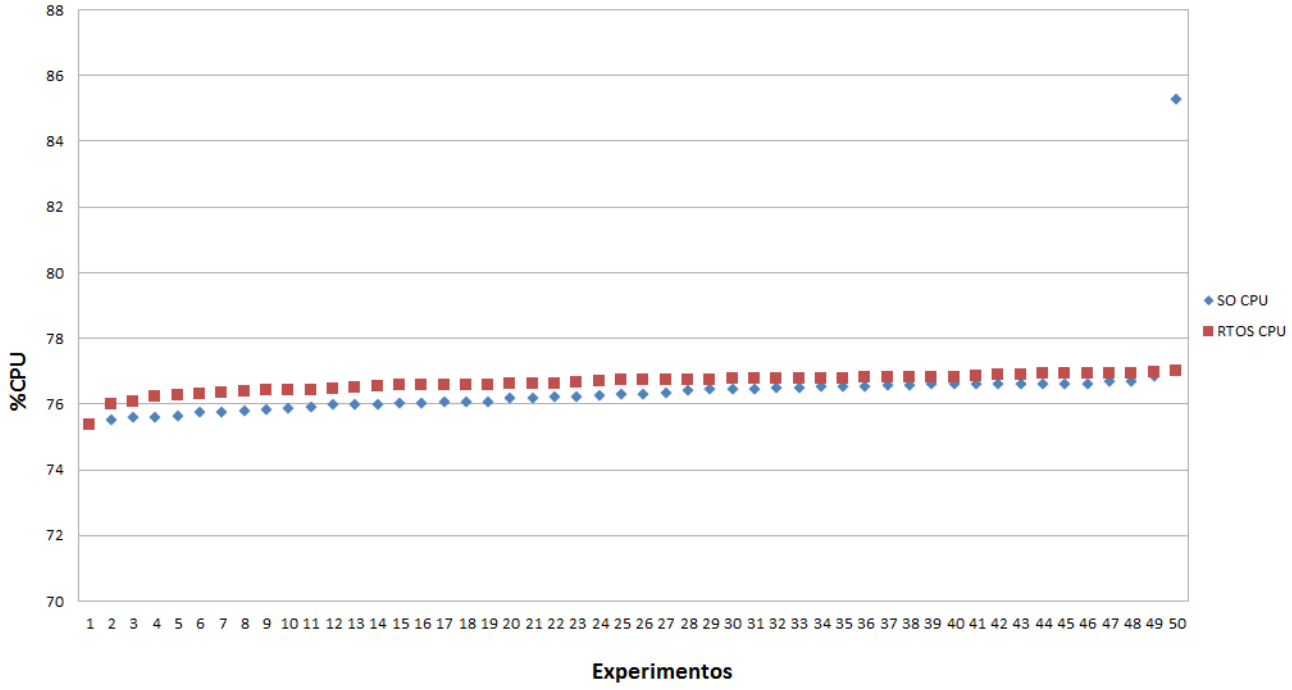

Figura 18 – Comparação do uso de CPU

A aplicação utilizada retorna em tela os tempos gastos da execução do problema e o tempo que a aplicação gasta para salvar os resultados em um arquivo. Nas Figura 19, 20 e 21 é apresentado a comparação dos tempos de execução, de escrita no arquivo e tempo total, entre

os sistemas avaliados. Na Figura 19 é possível observar que o tempo de execução no RTOS é inferior ao tempo do SO de propósito geral. Já o tempo de escrita no arquivo, apresentado na Figura 20, é maior no RTOS. Como o tempo de escrita é maior que o tempo de execução (por volta de 30 vezes no RTOS e de 20 vezes no SO), e a escrita no RTOS é mais lenta, obtemos assim a Figura 21 que apresenta o RTOS com mais gastos de tempo apesar de sua execução ser mais eficiente.

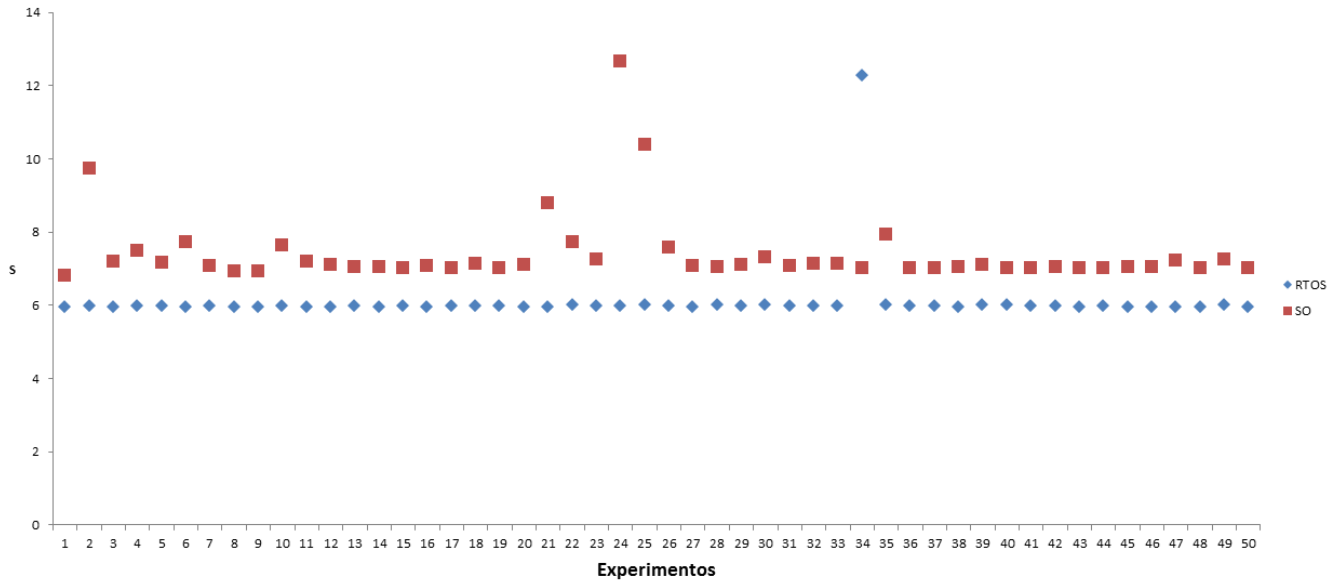

Figura 19 – Tempo de Execução da Aplicação

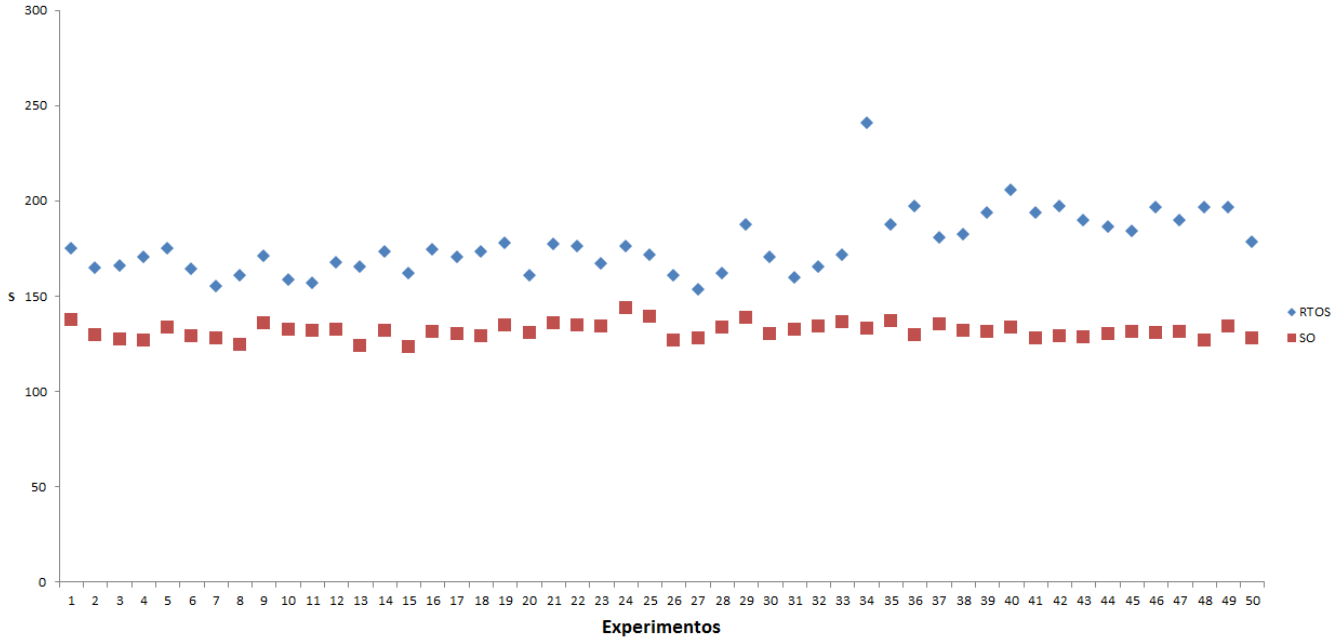

Figura 20 – Tempo de Escrita do Arquivo

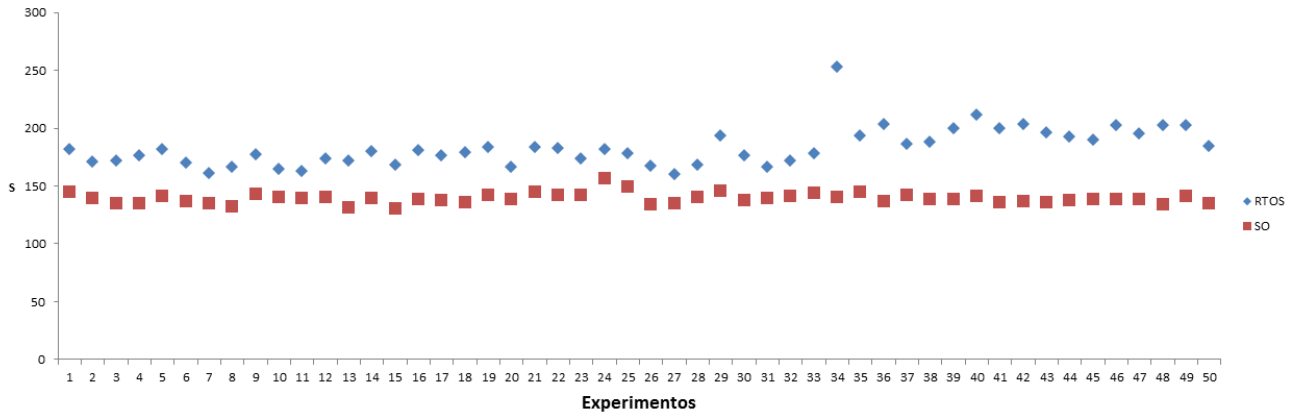

Figura 21 – Tempo Total de Execução da Aplicação

Uma representação mais completa dos dados de uso de memória dos experimentos é apresentada na Figura 23 e na Figura 24 dos cinquenta experimentos em RTOS e cinquenta em SO de propósito geral. A representação destas é feita por meio de *boxplot*. Esta representação se torna mais completa pela sua representatividade de dados e suas distribuições. A Figura 22 informa os 5 dados que esta estrutura apresenta. Entre eles se encontram:

- **Limite Inferior:** Valor Mínimo;
- **Primeiro Quartil (Q_1):** Corresponde a 25% das menores medidas;
- **Segundo Quartil (Q_2) ou Mediana:** Corresponde a 50% das maiores medidas e 50% das menores medidas (mediana);
- **Terceiro Quartil (Q_3):**Corresponde a 25% das maiores medidas;
- **Limite Superior:** Valor Máximo.

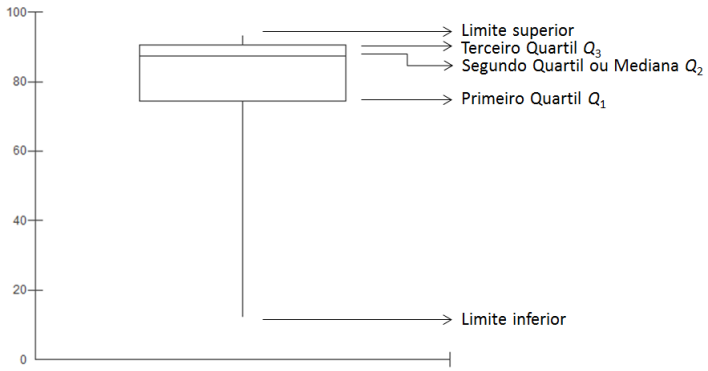

Figura 22 – Representação do *boxplot*

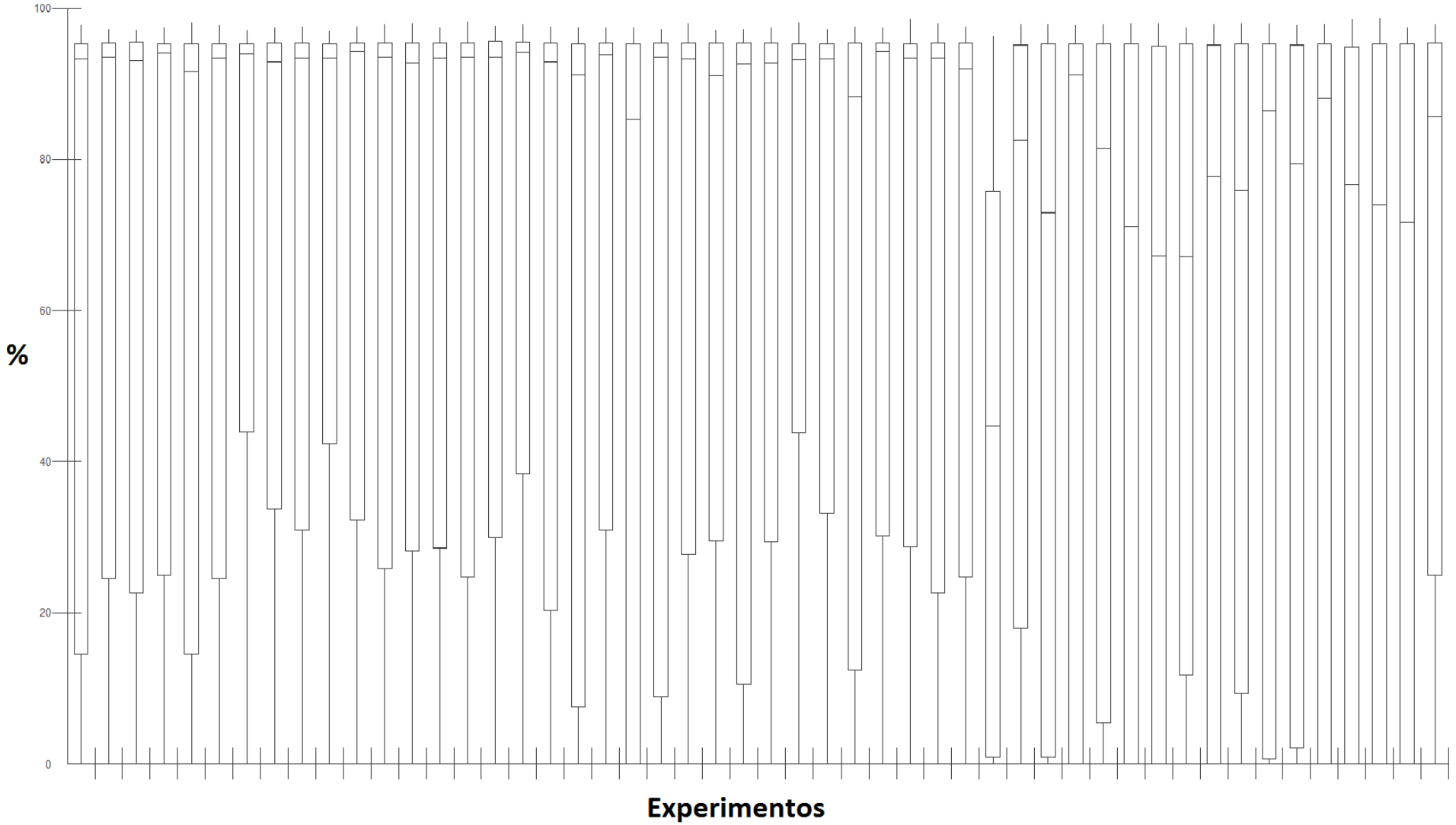

Figura 23 – Uso de memória no RTOS

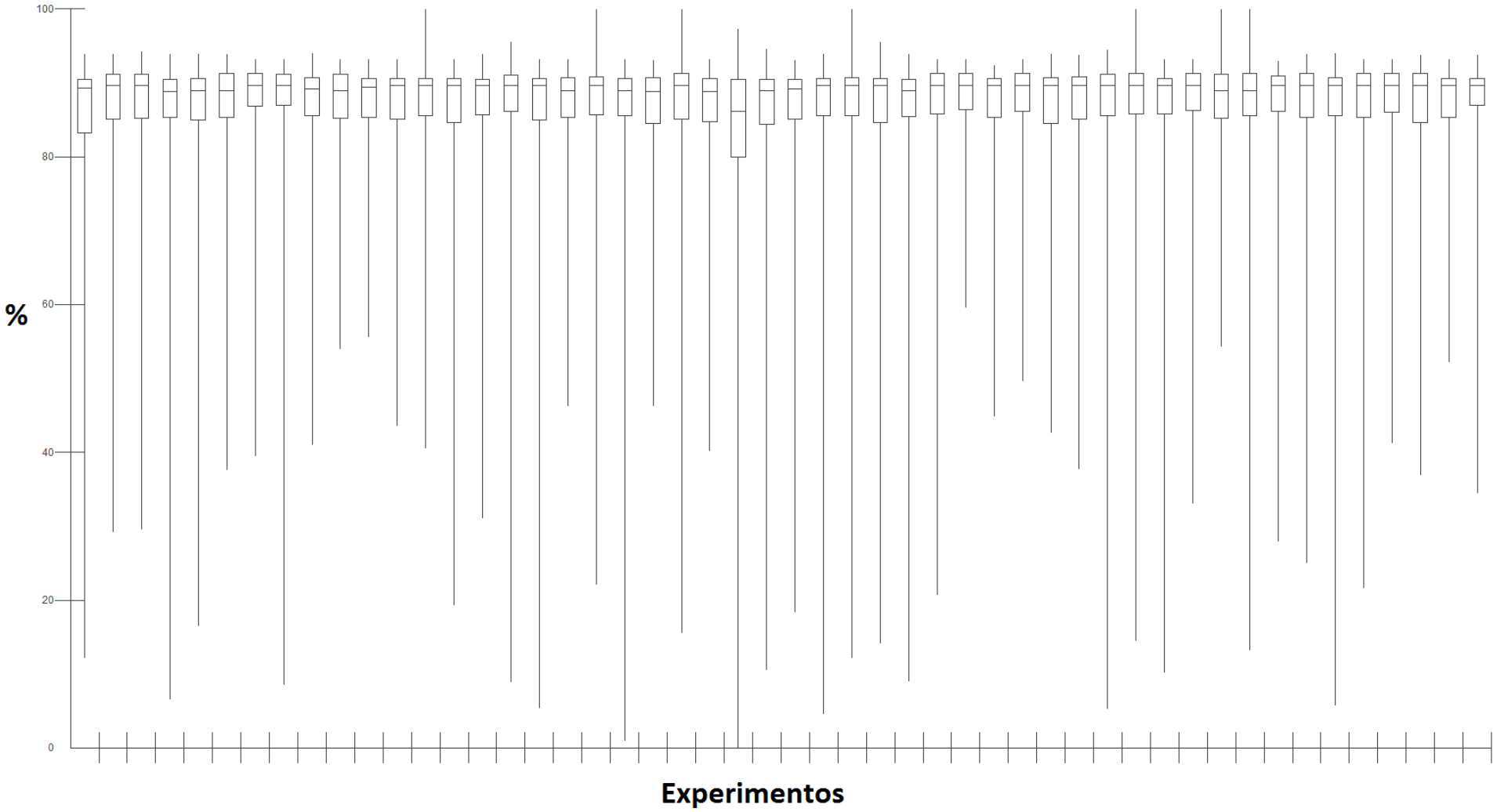

Figura 24 – Uso de memória no SO

Tabela 2 – Teste de Normalidade - Shapiro Wilk

		RTOS	**SO**
Média	Máx	84.428	87.69
	Mín	40.224	75.266
Desvio Padrão	Máx	41.2506	24.1923
	Mín	23.3754	6.0376
W	Máx	0.9073	0.7852
	Mín	0.5267	0.3835
P	Máx	0.0098	0.0084
	Mín	0.0057	0.0041

Os principais testes estatísticos têm como suposição a normalidade dos dados, o teste de Shapiro e Wilk (1965) é um teste de partida de normalidade. A normalidade deve ser verificada antes da realização das análises principais. Portanto foi realizado o teste de Shapiro-Wilk para verificar se os dados tinha distribuição normal. O valor de p é uma probabilidade que mede a evidência contra a hipótese nula. Um valor de p menor fornece uma evidência mais forte contra a hipótese nula. Utiliza-se o valor de p para determinar se os dados não seguem uma distribuição normal. Geralmente, um nível de significância de 0,05 funciona bem, que indica um risco de 5% de concluir que os dados não seguem a distribuição normal. Portanto, observando o p na Tabela 2 pode-se concluir que os dados seguem uma distribuição normal.

Como a distribuição é normal foi realizado o teste t, que é um teste de hipótese, que usa conceitos estatísticos, e pode determinar se uma hipótese é nula ou não.

Na Tabela 3 é um comparativo do Teste t com duas amostras independentes, sendo a primeira de SO e a segunda de RTOS, de tamanho 50, 100 e 120. Pode-se observar que quanto maior amostragem, menor o p. Apesar de no tamanho da amostra 120 sugerir uma distinção entre os dados, não fica claro quando visualizado no tamanho da amostra de tamanho 50, neste contexto seria necessário realizar um número maior de testes para assim ter uma amostragem maior e poder validar a hipótese.

Tabela 3 – Teste t

	SO	RTOS	SO	RTOS	SO	RTOS
Tamanho	50	50	100	100	120	120
Média	79.436	76.548	83	74.156	80.4692	68.2917
Variância	299.757	1287.661	189.6301	1298.2574	313.45	1393.7297
	Heterocedasticidade	—	Heterocedasticidade	—	Heterocedasticidade	—
Variância	31.7484	—	14.8789	—	14.2265	—
t	0.5126	—	2.2928	—	3.2286	—
Graus de liberdade	70.64	—	127.32	—	169.95	—
p (unilateral)	**0.3049**	**—**	**0.0118**	**—**	**0.0008**	—
p (bilateral)	0.6099	—	0.0235	—	0.0016	—
Poder (0.05)	0.127	—	0.7414	—	0.9434	—
Poder (0.01)	0.0129	—	0.4852	—	0.8156	—
Diferença entre as médias	2.888	—	8.844	—	12.1775	—
IC 95% (Dif. entre médias)	-8.4360 a 14.2120		1.1530 a 16.5350		4.7009 a 19.6541	
IC 99% (Dif. entre médias)	-12.2121 a 17.9881	—	-1.3793 a 19.0673	—	2.2861 a 22.0689	—

Os Benchmarks são aplicações capazes de comparar a performance e desempenho relativo de sistemas computacionais utilizando testes padrões e ensaios de ações. Para nível de curiosidade, foi realizado testes com o Banchmarking Sysbench atraves da linha de comando:

```
sysbench --num-threads=1 --test=memory run
```

Os resultados do Raspian e do Xenomai se encontram na Tabela 4. Onde pode-se observar que o RTOS possui um melhor desempenho quando aplicado carga nos sistemas.

Tabela 4 – Resultado Sysbench para memória

	Raspbian	**Xenomai**
Sem carga	0.0061s	0.0052s
Com carga	0.0185s	0.0126s

4.2 Considerações Finais

Este presente trabalho avaliou e comparou o uso de memória em um Sistema Operacional de propósito geral embarcados e um Sistema Operacional em Tempo Real embarcado. A proposta inicial foi completada ao embarcar o SO Raspian e o RTOS Xenomai em uma Raspberry Pi e nestes executado uma aplicação *memory bound* com o intuito de avaliar seus desempenhos.

Com os dados recolhidos é possível concluir que o RTOS escolhido tem um melhor desempenho no gerenciamento de memória em comparação com o SO testado. Contudo é necessário mais estudos para comprovar a superioridade de eficiência geral, pois características como gerenciamento de processos, gerenciamento de mensagens, gerenciamento de arquivos e gerenciamento de Entrada/Saída não foram avaliados.

Outro ponto a ser levantado, é que este estudo foi específico para o ambiente de desenvolvimento escolhido, sendo assim não é possível generalizar que RTOS possuem melhor desempenho de memória que SO de propósito geral.

É necessário também destacar que a curva de aprendizado necessária para utilizar um RTOS é maior assim como a dificuldade de encontrar documentação e suporte, elevando o custo de desenvolvimento em aplicações em tempo real. Neste projeto, a falta de documentação de RTOS, no geral, levou o atraso do cronograma ao ser necessário testar vários sistemas que não funcionavam apropriadamente.

Todos os dados na integra dos experimentos se encontram em no repositório GitHub[1].

[1] github.com/veevannini/desempenhoMemoria

REFERÊNCIAS

AL-THOBAITI, B. M.; ABOSOLAIMAN, I. I.; ALZAHRANI, M. H.; ALMALKI, S. H.; SOLIMAN, M. S. Design and implementation of a reliable wireless real-time home automation system based on arduino uno single-board microcontroller. **International journal of control, Automation and systems**, v. 3, n. 3, p. 11–15, 2014. Citado 4 vezes nas páginas 7, 19, 21 e 22.

DOCEKAL, T.; SLANINA, Z. Control system based on freertos for data acquisition and distribution on swarm robotics platform. In: IEEE. **Carpathian Control Conference (ICCC), 2017 18th International**. [S.l.], 2017. p. 434–439. Citado 4 vezes nas páginas 7, 19, 27 e 28.

FARINES, J.-M.; FRAGA, J. d. S.; OLIVEIRA, R. S. d. **Sistemas de Tempo Real**. [S.l.]: Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, 2000. v. 1. Citado na página 18.

GAGNE, S. G. **Fundamentos de Sistemas Operacionais**. [S.l.]: Ltc, 2013. v. 11. Citado na página 17.

KOH, J. H.; CHOI, B. W. Real-time performance of real-time mechanisms for rtai and xenomai in various running conditions. **International Journal of Control and Automation**, v. 6, n. 1, p. 235–246, 2013. Citado 4 vezes nas páginas 7, 19, 20 e 21.

NGUYEN, H.-Q.; LOAN, T. T. K.; MAO, B. D.; HUH, E.-N. Low cost real-time system monitoring using raspberry pi. In: IEEE. **Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on**. [S.l.], 2015. p. 857–859. Citado 4 vezes nas páginas 7, 19, 22 e 23.

NOVELLI, B. A.; LEITE, J. C.; URRIZA, J. M.; OROZCO, J. D. Regulagem dinâmica de voltagem em sistemas de tempo real. **XXXII Seminário Integrado de Software e Hardware (SBC 2005 SEMISH)**, 2005. Citado na página 19.

ONKAR, T. A.; KARULE, P. Web based maintenance for industrial application using raspberry-pi. In: IEEE. **Green Engineering and Technologies (IC-GET), 2016 Online International Conference on**. [S.l.], 2016. p. 1–4. Citado 5 vezes nas páginas 7, 19, 24, 25 e 26.

PEREIRA, E. d. S.; MARQUES, E. Um framework para coprojeto de hardware e software do modelo climático brams. **Monografia de qualificação**, 2016. Citado na página 30.

RAI, G.; KUMAR, S. A comparative study: Rtos and its application. **International Journal of Computer Trends and Technology (IJCTT)**, 2015. Citado 2 vezes nas páginas 19 e 22.

SHAH, V. H.; SHAH, A. An analysis and review on memory management algorithms for real time operating system. **International Journal of Computer Science and Information Security (IJCSIS)**, 2016. Citado 2 vezes nas páginas 19 e 26.

SHAPIRO, S. S.; WILK, M. B. **An analysis of variance test for normality (complete samples)**. [S.l.]: JSTOR, 1965. v. 52. 591–611 p. Citado na página 42.

STALLINGS, W. **Comunicaciones y redes de computadores**. [S.l.]: Pearson Educación,, 2004. Citado na página 17.

TANENBAUM, A. S. **Sistemas Operacionais Modernos-Livros Técnicos e Científicos Ed**. [S.l.: s.n.], 1999. Citado 2 vezes nas páginas 17 e 18.

TRINDADE, O. J.; BRAGA, R. T. V.; NERISY, L. d. O.; BRANCO, K. R. L. J. C. Uma metodologia para desenvolvimento de sistemas embarcados críticos com vistas a certificação. **Anais do IX Simpósio Brasileiro de Automação Inteligente**, 2009. Citado na página 17.

WAZLAWICK, R. **Metodologia de pesquisa para ciência da computação**. [S.l.]: Elsevier Brasil, 2017. v. 2. Citado na página 29.

APÊNDICE

A

TESTE DE NORMALIDADE DE SHAPIRO-WILK

Tabela 5 – Teste de Normalidade de Shapiro-Wilk no RTOS

Amostra	Tamanho da amostra	Média	Desvio padrão	W	p
1	50	76.548	35.884	0.5619	0.006
2	50	80.556	31.972	0.5267	0.0057
3	50	81.172	30.317	0.5426	0.0058
4	50	72.668	35.0284	0.6793	0.0073
5	50	66.464	38.8122	0.7125	0.0077
6	50	80.668	30.9703	0.5322	0.0057
7	50	84.428	24.3912	0.5444	0.0059
8	50	79.66	29.7798	0.6034	0.0065
9	50	79.496	30.5062	0.591	0.0064
10	50	78.946	30.0987	0.6148	0.0066
11	50	78.096	31.6058	0.6121	0.0066
12	50	79.53	30.218	0.5982	0.0064
13	50	82.408	25.327	0.6162	0.0066
14	50	78.16	31.2423	0.6235	0.0067
15	50	78.576	32.3804	0.5802	0.0062
16	50	79.18	32.1844	0.5716	0.0061
17	50	74.42	34.6931	0.6523	0.007
18	50	76.044	35.0487	0.5976	0.0064
19	50	76.474	31.0898	0.6685	0.0072
20	50	78.47	33.714	0.5566	0.006

continua na próxima página

Tabela 5 – continuação

Amostra	Tamanho da amostra	Média	Desvio padrão	W	p
21	50	75.682	33.1645	0.643	0.0069
22	50	75.264	35.6334	0.6018	0.0065
23	50	80.744	29.8763	0.5684	0.0061
24	50	73.942	34.4729	0.6607	0.0071
25	50	74.418	36.0944	0.6079	0.0065
26	50	72.422	35.8847	0.6636	0.0071
27	50	75.738	34.3532	0.624	0.0067
28	50	73.334	35.7003	0.6383	0.0069
29	50	78.42	31.3322	0.6154	0.0066
30	50	76.862	33.8803	0.6004	0.0065
31	50	78.402	31.9671	0.6033	0.0065
32	50	71.964	35.8119	0.6762	0.0073
33	50	73.334	37.4667	0.6105	0.0066
34	50	40.224	23.3754	0.9073	0.0098
35	50	70.914	35.0342	0.712	0.0077
36	50	65.272	41.2506	0.6673	0.0072
37	50	75.108	35.6971	0.6084	0.0065
38	50	81.99	29.4749	0.5291	0.0057
39	50	72.646	37.0577	0.6389	0.0069
40	50	64.044	40.6406	0.7071	0.0076
41	50	71.084	35.1158	0.7078	0.0076
42	50	67.408	40.1178	0.665	0.0072
43	50	65.748	36.8112	0.7577	0.0081
44	50	63.376	41.0812	0.6986	0.0075
45	50	75.358	34.1834	0.6313	0.0068
46	50	83.102	24.948	0.604	0.0065
47	50	76.83	31.5846	0.6451	0.0069
48	50	77.26	32.987	0.612	0.0066
49	50	74.304	34.1537	0.6623	0.0071
50	50	68.31	38.877	0.6833	0.0073

Tabela 6 – Teste de Normalidade de Shapiro-Wilk no SO

Amostra	Tamanho da amostra	Média	Desvio padrão	W	p
1	50	79.436	17.3135	0.734	0.0079
2	50	83.066	13.1554	0.6996	0.0075
3	50	84.006	12.5001	0.6901	0.0074
4	50	85.726	12.4294	0.4181	0.0045
5	50	85.312	10.569	0.6117	0.0066
6	50	83.988	13.5521	0.6297	0.0068
7	50	87.1	9.3748	0.5499	0.0059
8	50	86.406	12.9601	0.452	0.0049
9	50	86.836	9.3766	0.6035	0.0065
10	50	86.824	7.342	0.6833	0.0073
11	50	87.116	6.0376	0.7464	0.008
12	50	86.364	8.2027	0.6603	0.0071
13	50	86.884	9.7977	0.611	0.0066
14	50	86.58	11.3249	0.4914	0.0053
15	50	86.364	9.8854	0.6043	0.0065
16	50	86.032	12.8881	0.4548	0.0049
17	50	84.538	14.1267	0.5345	0.0057
18	50	84.956	10.4391	0.6832	0.0073
19	50	85.902	12.4768	0.5364	0.0058
20	50	85.622	10.611	0.6057	0.0065
21	50	83.61	11.3009	0.7389	0.0079
22	50	85.314	12.3479	0.6061	0.0065
23	50	84.574	10.2414	0.732	0.0079
24	50	75.266	24.1923	0.7177	0.0077
25	50	84.052	14.5457	0.5661	0.0061
26	50	83.716	13.3273	0.6124	0.0066
27	50	86.41	12.6296	0.3835	0.0041
28	50	86.254	13.3605	0.4961	0.0053
29	50	84.088	15.2323	0.5609	0.006
30	50	84.712	13.6041	0.529	0.0057
31	50	84.538	13.9164	0.5733	0.0062
32	50	87.69	6.1814	0.7055	0.0076
33	50	84.742	11.4454	0.6507	0.007
34	50	87.402	6.7385	0.7037	0.0076

continua na próxima página

Tabela 6 – continuação

Amostra	Tamanho da amostra	Média	Desvio padrão	W	p
35	50	83.126	11.6546	0.7244	0.0078
36	50	84.812	13.2934	0.563	0.0061
37	50	84.27	14.195	0.5502	0.0059
38	50	84.95	14.96	0.5359	0.0058
39	50	86.546	11.9811	0.4318	0.0046
40	50	86.74	9.9414	0.5407	0.0058
41	50	86.37	8.7014	0.7852	0.0084
42	50	85.57	13.4857	0.5245	0.0056
43	50	85.908	11.8674	0.4928	0.0053
44	50	84.458	14.4726	0.5566	0.006
45	50	84.526	16.035	0.4951	0.0053
46	50	84.024	14.2574	0.5875	0.0063
47	50	87.298	7.7317	0.5441	0.0059
48	50	86.038	9.8906	0.6458	0.0069
49	50	87.11	7.3763	0.5991	0.0064
50	50	86.89	9.1925	0.535	0.0058

APÊNDICE

B

MÉDIA DE USO DE CPU E MEMÓRIA DOS EXPERIMENTOS

Tabela 7 – Média de uso de CPU e Memória dos experimentos

Experimento	Memória [%]		CPU [%]	
	RTOS	SO	RTOS	SO
1	80.5377	44.52353	75.38142	75.34118
2	83.32389	56.43706	75.49912	75.98418
3	84.22735	56.71471	75.59658	76.0697
4	84.32222	56.80647	75.59914	76.22407
5	84.36	56.83647	75.62261	76.25337
6	84.81111	56.95	75.73119	76.28701
7	85.00789	57.13765	75.75641	76.33533
8	85.03917	57.29294	75.78833	76.36272
9	85.26522	57.44	75.82857	76.38735
10	85.41875	57.72765	75.84696	76.38912
11	85.44701	57.88529	75.90783	76.40176
12	85.56723	58.27471	75.9563	76.44294
13	85.6823	58.47765	75.97203	76.46266
14	85.73929	58.83728	75.98559	76.53831
15	85.75487	59.00176	75.99739	76.54471
16	85.75487	59.45353	76.02696	76.55941
17	85.83391	61.25353	76.03559	76.56125
18	85.83661	61.32059	76.05714	76.56536
19	85.84052	61.66294	76.05943	76.56941
20	85.92174	61.70119	76.16339	76.58258

continua na próxima página

Tabela 7 – continuação

Experimento	Memória [%]		CPU [%]	
	RTOS	SO	RTOS	SO
21	85.94576	61.70412	76.17281	76.58813
22	85.97545	62.0128	76.21842	76.61497
23	85.98696	62.24192	76.22432	76.64765
24	86.03894	63.03136	76.26496	76.67703
25	86.06134	63.30723	76.29076	76.69747
26	86.10336	63.33987	76.29912	76.70882
27	86.17358	63.58606	76.3094	76.72294
28	86.1913	63.85417	76.4041	76.72679
29	86.2819	64.28608	76.43421	76.72761
30	86.30263	64.40633	76.45089	76.7369
31	86.33486	64.46169	76.46	76.73709
32	86.4265	64.46173	76.46752	76.74157
33	86.45304	64.6025	76.48142	76.75226
34	86.46957	65.13938	76.50435	76.75412
35	86.52966	65.14114	76.52522	76.75412
36	86.60348	65.38976	76.53826	76.78059
37	86.62895	65.6013	76.54561	76.78165
38	86.63947	65.62275	76.57434	76.78765
39	86.7	65.91699	76.58929	76.79294
40	86.73077	66.08839	76.5931	76.80613
41	86.87798	66.12	76.59469	76.82622
42	87.24957	66.23252	76.5975	76.86765
43	87.30169	66.2638	76.59908	76.87647
44	87.40609	66.52745	76.60354	76.90471
45	87.4161	66.84901	76.60877	76.90647
46	87.44054	66.87791	76.61356	76.92367
47	87.60593	67.62933	76.68983	76.92575
48	87.68661	68.47905	76.69558	76.92699
49	87.93043	68.50982	76.84182	76.94052
50	88.0177	68.97687	85.26522	76.98647

APÊNDICE

C

TEMPO DE EXECUÇÃO DA APLICAÇÃO

Tabela 8 – Tempo de Execução da Aplicação

Experimento	RTOS			SO		
	Execução	**Escrita**	**Total**	**Execução**	**Escrita**	**Total**
1	5.945501	175.1847	181.1302	6.793229	137.4248	144.218
2	5.985809	164.6716	170.6574	9.728221	129.4079	139.1361
3	5.949368	165.6091	171.5584	7.184869	127.4015	134.5864
4	5.986807	170.4744	176.4612	7.472655	126.7637	134.2363
5	5.992645	175.2	181.1926	7.167422	133.6679	140.8353
6	5.946318	164.1824	170.1287	7.71908	129.1588	136.8779
7	5.992049	154.8179	160.81	7.085737	127.5884	134.6741
8	5.950114	160.6171	166.5672	6.933327	124.68	131.6133
9	5.950902	171.0247	176.9756	6.926859	135.5648	142.4916
10	5.990531	158.5153	164.5059	7.633775	132.34	139.9738
11	5.94908	156.718	162.6671	7.19299	131.6062	138.7992
12	5.950877	167.365	173.3159	7.115538	132.5992	139.7147
13	5.98601	165.294	171.28	7.037933	123.7538	130.7917
14	5.957908	173.4715	179.4294	7.029763	131.6791	138.7089
15	5.986874	161.9072	167.8941	7.012902	123.34	130.3529
16	5.94085	174.5343	180.4752	7.075934	131.4699	138.5459
17	5.989636	170.1915	176.1811	7.008669	129.9219	136.9305
18	5.987911	173.2093	179.1973	7.146043	128.7905	135.9366
19	5.990003	177.5075	183.4975	7.025579	134.5123	141.5378
20	5.942913	160.5176	166.4605	7.089765	130.7588	137.8485
21	5.947843	176.9819	182.9298	8.781922	135.78	144.5619

continua na próxima página

Tabela 8 – continuação

Experimento	RTOS			SO		
	Execução	**Escrita**	**Total**	**Execução**	**Escrita**	**Total**
22	6.013196	176.3299	182.3431	7.710754	134.5855	142.2963
23	5.967022	167.1928	173.1598	7.257897	134.3445	141.6024
24	5.991856	175.9113	181.9032	12.6462	143.4739	156.1201
25	5.994195	171.7278	177.722	10.37637	138.931	149.3074
26	5.992416	160.7824	166.7749	7.562225	126.4756	134.0378
27	5.94719	153.4896	159.4368	7.071924	127.8315	134.9034
28	6.017685	162.0058	168.0235	7.036603	133.2926	140.3292
29	5.99122	187.4386	193.4298	7.104677	138.3805	145.4852
30	5.996643	170.4951	176.4917	7.310569	130.2574	137.568
31	5.987799	159.8209	165.8087	7.078724	132.3936	139.4723
32	5.992719	165.5716	171.5643	7.119126	134.0049	141.124
33	5.96582	171.7127	177.6785	7.122455	136.4186	143.541
34	12.26524	240.6625	252.9277	7.019093	132.7442	139.7633
35	5.997409	187.3347	193.3321	7.923942	136.8023	144.7262
36	5.975733	197.13	203.1057	7.018004	129.5664	136.5844
37	5.993338	180.3692	186.3626	7.005933	135.2211	142.2271
38	5.956707	182.1215	188.0783	7.031081	131.6531	138.6841
39	5.994618	193.8766	199.8712	7.111258	131.3535	138.4647
40	6.010086	205.6746	211.6847	7.00561	133.6356	140.6412
41	5.991512	193.9288	199.9204	7.009421	128.1105	135.1199
42	5.973526	197.1496	203.1231	7.032216	129.2082	136.2405
43	5.954065	189.8137	195.7678	7.026855	128.3188	135.3457
44	5.98964	186.3757	192.3653	7.027646	129.8977	136.9254
45	5.949811	183.7645	189.7143	7.042589	131.0604	138.103
46	5.948023	196.5822	202.5302	7.038402	130.836	137.8744
47	5.953475	189.5126	195.4661	7.226053	131.1582	138.3843
48	5.947932	196.6913	202.6392	7.023723	126.6266	133.6503
49	6.015437	196.3526	202.3681	7.255456	133.9478	141.2033
50	5.949132	178.1637	184.1128	7.02033	127.7167	134.737