INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE P
DEPARTAMENTO ACADÊMICO DE SISTEMA, PROCESSOS E CONTROLES EI
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE

Daniel Barlavento Gomes

COLOQUE SEU TÍTULO AQUI

Recife – Pernambuco

2017

Daniel Barlavento Gomes

COLOQUE SEU TÍTULO AQUI

Trabalho de conclusão apresentado ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas da IFPE, como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Mestre Paulo Abadie Guedes

Recife – Pernambuco

2017

Daniel Barlavento Gomes

COLOQUE SEU TÍTULO AQUI

Trabalho de conclusão apresentado ao curso de Tecnologia em Análise e Desenvolvimento de Sistemas da IFPE, como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Recife, 04/12/2017.

BANCA EXAMINADORA

Paulo Abadie Guedes

Prof. Mestre - IFPE

(Professor Orientador)

Examinador Interno 1

Prof. Doutor - IFPE

(Professor do Instituto Federal de Pernambuco)

Examinador Externo 2

Prof. Doutor - IFPE

(Professor do Curso de Sistemas de Informações da Universidade)

Dedico a minha família, por todo o apoio e confiança.

AGRADECIMENTOS

"Prefiro não fazer".

Herman Melville (Bartleby, o escriturário)

RESUMO

Morbi efficitur molestie pellentesque. Fusce tincidunt vitae dolor ac ornare. Mauris nibh mi, condimentum nec ex a, semper posuere augue. Ut sagittis condimentum lacus, et lacinia sem ornare nec. Praesent cursus sagittis lacus ut iaculis. Nunc faucibus, elit ac imperdiet malesuada, velit est faucibus diam, vitae ullamcorper sapien augue a nisi. Morbi consectetur pulvinar felis vel feugiat. Phasellus tempor magna eget purus placerat luctus. Vestibulum bibendum dapibus arcu in semper. Phasellus vel porta mauris. Ut nec mauris vel ante auctor vulputate a sed nisi. Nam ullamcorper purus vel dolor interdum efficitur. Aenean rhoncus mollis porta. Vivamus est urna, finibus vel leo at, porta tempus sem.

Palavras-chave:

ABSTRACT

Curabitur malesuada ante lorem, a auctor urna euismod et. Nam viverra, dolor eu feugiat euismod, justo velit tincidunt purus, faucibus interdum mauris metus in turpis. Maecenas hendrerit, felis quis condimentum convallis, metus turpis porttitor ex, non iaculis nisi ex id ligula. Vivamus sed consectetur felis. Maecenas non ligula eu nulla iaculis dictum. Phasellus accumsan tempus purus et consectetur. Praesent dapibus, arcu ut porta dictum, velit lacus ultricies nisl, vitae congue purus mi id ipsum. Pellentesque ac tempus enim, at egestas nulla. Quisque vitae ultrices odio. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed vitae purus ultricies, maximus magna a, aliquet mauris. Aliquam ornare odio sit amet urna placerat vestibulum. Aenean a cursus mauris, quis vulputate erat. Nullam convallis scelerisque ligula, at finibus lectus laoreet at.

Keywords:

Sumário

1 INTRODUÇÃO

Sistemas de tempo real se tornaram elemento constante na vida das pessoas e estão presentes em locais que vão de aparelhos condicionadores de ar a grandes usinas geradores de energia.

Devido ao aumento da complexidade dos sistemas de tempo real os desenvolvedores veem procurando soluções que permitam o desenvolvimento destes sistemas de forma rápida, estruturada e que possível de ser mantida a longo prazo, o que sempre foi uma grande dificuldade se comparado com os sistemas desenvolvidos em *assembly*. Devido ao aumento da capacidade de processamento dos processadores atuais, como parte da solução, sistemas operacionais de tempo real vem sendo cada vez mais utilizados no desenvolvimento de novos projetos, pois proporcionam uma grande variedade de funcionalidades previamente implementadas e gerenciam praticamente todo o hardware.

Grande parte dos sistemas operacionais de tempo real disponíveis no mercado são proprietários, com um alto custo de licenciamento, ou não possuem suporte a recursos mais avançados exigidos por algumas aplicações. Devido a este problema vários projetos foram desenvolvidos com a finalidade de transforma o Linux em um verdadeiro sistema operacional de tempo real. Dentre as soluções criadas podemos destacar o *patch* Preempt_RT, suportado oficialmente pelos desenvolvedores do *kernel* Linux, e o RTAI uma solução baseada em *microkernel*.

A validação por meio da execução de *benchmarks* da transformação do Linux em um sistema de tempo real pelo Preempt_RT e pelo RTAI, fornece uma base sólida de dados que permitem a um projetista de sistemas de tempo real comparar as soluções baseadas em Linux com outros sistemas e verificar se suas restrições temporais podem ser atendidas.

1.1 Objetivos

Este trabalho tem como objetivos gerais e específicos:

1.1.1 Gerais

Esse trabalho tem como objetivo avaliar a capacidade do *patch* Preempt_RT, oficialmente suportado pelos desenvolvedores do *kernel*, de transformar um PC com um único processador em um computador capaz atender aos requisitos de uma aplicação de tempo real rígida e comparar os resultados com o RTAI, um sistema maduro, testado e consolidado.

Para esta avaliação foram escritas duas aplicações, baseadas nos testes realizados por Anderson(2007) e no *benchmark* Cyclictest, e portadas para o RTAI para que fosse possível realizar a comparação. Nesse processo também foi observada a facilidade de instalação dos respectivos patch e desenvolvimento de aplicações para ambas as soluções.

1.1.2 Específicos

- Verificar a viabilidade de uso do *patch* Preempt_RT e do RTAI na máquina de testes e distribuição escolhida
- Criação de *kernels* de tempo real baseados nas duas soluções estudadas
- Escrita dos testes
- Aplicação dos testes
- Avaliação dos resultados obtidos

1.2 Trabalhos relacionados

Durante a pesquisa bibliográfica foram identificados alguns trabalhos semelhantes e que nos proveram diversos recursos para a elaboração deste trabalho, dentre eles se destacam (MOREIRA, 2007), que nos forneceu o modelo de teste utilizado neste trabalho, assim como medições de performance do RTAI, utilizados como valores de referência para verificação dos testes desenvolvidos.

Também podem ser destacados os trabalhos de (SAOUD, 2011) e (HALLBERG, 2017) que forneceram valores de referência e demonstram a real qualidade dos resultados oferecidos pelo *benchmark* Cyclictest na avaliação e comparação de sistemas de tempo real baseados em Linux.

2 Fundamentação Teórica

2.1 Sistemas de Tempo Real

Quando falamos de sistemas de tempo real (STR) logo pensarmos em sistemas de controle industrial ou em algum tipo de sistemas embarcado ultra rápido, porém os STR vão além destas aplicações, indo, em seus primórdios computacionais, da decodificação de mensagens inimigas aos modernos sistemas computacionais utilizados por instituições financeiras que por questões regulamentares precisam garantir que os tempos de suas transações estejam dentro do limite estipulado sem, no entanto, serem longas ao ponto de provocarem perdas monetárias. Essa vasta gama de aplicações nos mostra o quão heterogêneos são estes sistemas e o quanto diversificada são suas implementações, indo de sistemas implementados utilizando alguma linguagem de montagem em microcontroladores de 8 bits a supercomputadores que executam complexos sistemas operacionais sobre os quais aplicações ainda mais complexas são também executas.

Segundo (LAPLANTE, 2012),um sistema pode ser dito de tempo real quando sua correção lógica está relacionada tanto a correção das saídas produzidas pelo sistema quanto pela sua pontualidade, ou seja são sistemas que além de prezarem pela qualidade e correção dos seu algoritmos computacionais, também devem prezar pela pontualidade com que suas ações são tomadas, do contrario o sistema falhará.

Ao contrário do que diz o senso comum, um sistemas de tempo real não tem como principais objetivos a diminuição dos tempos de resposta a estímulos ou simplesmente ser extremamente veloz, um sistemas de tempo real tem como principal objetivo atender aos prazos estabelecidos no domínio do problema de forma determinística e por consequência previsível. (FARINES, 2000) afirma que, um sistema de tempo real é previsível nos domínios lógico e temporal quando, independente das circunstâncias do hardware, carga ou falhas, pode-se saber com antecipação o seu comportamento antes da execução. A rigor, para que o comportamento do sistemas seja estabelecido de forma determinística é necessário conhecer todas as variáveis que compõem o ambiente em que o sistema está inserido como: hipóteses de falha, carga computacional, arquitetura do hardware,

sistema operacional, linguagem de programação, etc, e que em cada fase do seu desenvolvimento metodologias e ferramentas sejam aplicadas na verificação do seu comportamento e previsibilidade.

2.1.1 Classificação

Existem diversas formas de se classificar os sistemas de tempo real, uma das formas mais comuns de faze-lo, (FARINES, 2000) e (LAPLANTE, 2012), é pela observação do rigor com que tratam seus requisitos temporais e no tipo de problemas que falhas relacionadas a esses requisitos podem provocar. Sendo assim, os sistemas, podem ser classificados como sendo do tipo *soft* (brandos) ou *hard* (rígidos).

Um sistema de tempo real é brando quando o não cumprimento de suas restrições temporais provoca apenas alguma degradação no seu desempenho porém sem provocar falhas graves. Geralmente nesses sistemas os prejuízos provocados pela degradação do desempenho são compensados pelos benefícios de sua operação normal.

Um sistema de tempo real é rígido quando a falta no cumprimento de uma única restrição temporal pode levar a uma falha catastrófica. Nesse caso, uma falha no cumprimento de um prazo provoca prejuízos infinitamente maiores, como: uma catástrofe ambiental, a perda de vidas humanas ou grandes perdas materiais, que as benesses do sistema em operação normal.

(LAPLANTE, 2012) ainda estabelece uma terceira classificação entre os sistemas brandos e rígidos, o sistema de tempo real firme, onde se enquadram sistemas em que a perda de uma quantidade limitada de prazos não compromete o desempenho de forma crítica, porém a extrapolação do limite de prazos perdidos leva a uma falha catastrófica.

2.1.2 Tarefas de Tempo Real

Como todo sistema computacional, STR executam uma ou um conjunto de tarefas com o objetivo de produzir algum trabalho útil. (KOPETZ, 2002) as define simplesmente como a execução de um programa sequencial, iniciando-se, normalmente, com a leitura de alguns dados de entrada e findando com a produção de algum resultado e alterando seu estado interno.

Uma tarefa pode se classificada como de tempo real se obedece aos dois pre-

ceitos básicos definidos anteriormente para sistemas de tempo real, os seja sua correção lógica está ligada a correção de suas saídas e a correção temporal, estando sujeitas ao cumprimento de prazos (*deadline*), e assim como os sistemas de tempo real, uma tarefa é dita crítica ou rígida, quando o não cumprimento de seus prazos pode provocar alguma falha catastrófica, e branda quando não lhes cumprir provocam, no máximo, uma diminuição do desempenho, sem grandes consequências.

As tarefas de tempo real, também são classificadas quanto a regularidade de sua ativação podendo ser periódicas, quando sua ativação ocorre em intervalos regulares predefinidos, ou como aperiódicas quando sua ativação ocorre de modo aleatório, normalmente em resposta a algum evento externo ou interno ao sistema. Quando executam múltiplas tarefas os sinais que determinam o inicio (ativação) e termino de uma tarefa são controlados pelo escalonador do sistema.

Quando um sistema executa múltiplas tarefas é necessária a implementação de um algoritmo de escalonamento. O escalonador é definido por (FARINES, 2000) como o componente do sistema que é responsável pela implementação das políticas de acesso e gerenciamento de uso do processador pelas tarefas. Quando o número de tarefas em execução simultânea é maior que o número de unidades de processamento disponíveis no sistema, além de um escalonador, o sistema deve prover algum mecanismo capaz de guardar o estado atual do processador associado as tarefas para que posteriormente esse estado possa ser carregado novamente e a execução retomada do ponto onde parou, este processo é chamado de chaveamento de contexto. Dentro deste cenário, segundo (TANENBAUM, 2009), uma tarefa pode assumir diversos estados conforme a figura 2.1.

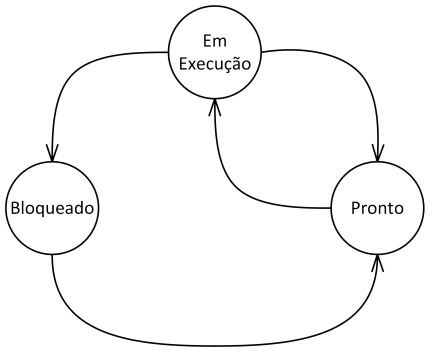

Figura 2.1: Estados de uma tarefa e seus relacionamentos

Na modelagem de tarefas de tempo real são utilizados alguns parâmetros que definem seu comportamento temporal: tempo de computação (Tc), tempo inicial (Ti),

tempo final (Tf), tempo de liberação (Tl), *deadline* (D) e, caso a tarefa em questão seja periódica, o período (P). O tempo de computação corresponde ao tempo total exigido para que a tarefa seja completada. Os tempos inicial e final correspondem aos instantes em que a tarefa inicia e finaliza, respectivamente, sua execução durante uma janela de ativação. O tempo de liberação é o momento em que uma tarefa entra no estado de "Pronto". O *deadline*, como já foi dito, é o prazo máximo que uma tarefa tem para concluir sua execução. E o período corresponde ao intervalo de tempo com que a tarefa repete sua execução. Outro parâmetro que deve ser considerado, principalmente quando trabalhamos com múltiplas tarefas e que é usado por diversos algoritmos de escalonamento, é a prioridade(PR, que representa a urgência relativa de execução de uma tarefa em relação as outras. O conhecimento desses parâmetros é bastante importante para garantir a previsibilidade do sistema, além de possibilitarem a verificação da viabilidade, montar as tabelas e definir as políticas, de escalonamento. Os parâmetros de tempo estão ilustrados na figura 2.2.

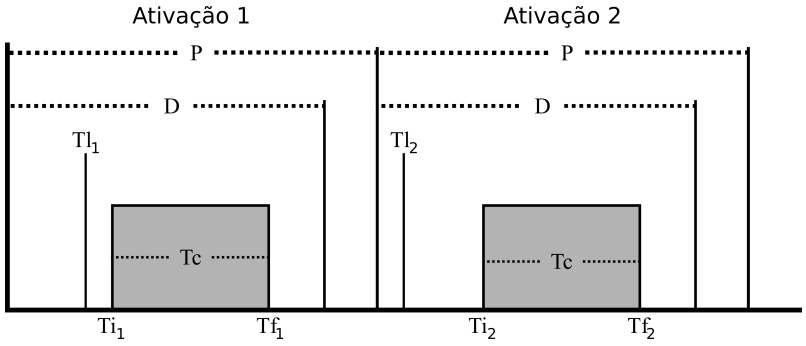

Figura 2.2: Características temporais de uma tarefa de tempo real

2.2 Sistemas Operacionais de Tempo Real

Com o aumento da complexidade das aplicações de tempo real (ATR) e consequentemente do hardware utilizado na execução das aplicações de tempo real, ouve também um aumento na complexidade do trabalho realizado pelos desenvolvedores de aplicações. Para contornar parte desse problema a utilização de sistemas operacionais de tempo real (SOTR) tornou-se cada vez mais comum. outro ponto importante, é o fato de que é bastante comum ATR possuírem algumas funcionalidades que não possuem restrições temporais como interações com banco de dados, acesso a internet, interfaces gráficas, etc, e que sem o uso de um sistema operacional, implementar estes recursos é um processo bastante exaustivo.

Segundo (TANENBAUM, 2009), um sistema operacional (SO) é um dispositivo de software que tem como principal finalidade fornecer um ambiente em que certas funcionalidades do sistema possam ser gerenciadas de forma autônoma e oculta ao desenvolvedor, proporcionando uma camada de abstração com um conjunto próprio de instruções sobre a qual o desenvolvedor possa maximizar seu trabalho utilizando uma interface de programação mais amigável.

Estendendo essa definição aos STR podemos dizer que um SOTR, além de possuir os principais objetivos comuns aos SO, deve criar um ambiente de desenvolvimento previsível e determinístico qualquer que seja a carga do sistema. Um SOTR deve fornecer um ambiente no qual ATR tenham seus requisitos temporais respeitados e executar de modo que suas ações possam ser previstas. Como geralmente são sistemas reativos, SOTR devem atender a requisitos relacionados a responsividade. Um SOTR deve garantir que respostas a estímulos, sejam internos ou externos, sempre serão dadas dentro de um intervalo de tempo que respeite os requisitos do sistema.

Assim como dito para os STR, um SOTR não tem como principais objetivos a redução dos tempos de resposta a estímulos ou ter uma performance superior quando comparado a um Sistemas Operacionais de Proposito Geral (SOPG). SOTR de qualidade podem ter desempenho global semelhante a SOPG, porém o primeiro, normalmente, sacrificará a performance em detrimento da previsibilidade. SOPG podem, em 99,9% dos casos, executar tarefas num tempo menor que SOTR, todavia nos 0,1% dos casos restantes, o tempo de execução de uma tarefa será imprevisível, podendo ser até 1000 vezes mais longo que em um SOTR, isso seria mais que suficiente para reprovar o sistema em uma aplicação crítica. Embora a execução de tarefas em um SOTR possam ser executadas num tempo maior, são executadas com a garantia de estarem sempre dentro dos prazos estabelecidos.

Dentre as principais funções de um SOTR, está o fornecimento mecanismos e ferramentas para que seja possível a execução de ATR de modo previsível e satisfatório aos requisitos impostos pelo domínio do problema. Os mecanismos oferecidos devem possibilitar ao desenvolvedor avaliar se o SOTR em questão é adequado ou não para a execução das aplicações pretendidas, isso inclui tornar acessível conhecer os valores de tempo máximo de execução de suas chamadas de sistema, os valores máximos das latências provocadas pelas operações internas do sistema e os valores de tempo relacionados as ATR. É importante observar que a qualidade dos valores apresentados está diretamente ligada

a granularidade dos temporizadores disponibilizados pelo sistema, quanto maior for a resolução dos temporizadores melhor a qualidade das medições. Estes valores de tempo também podem variar de acordo com a arquitetura do processador em que o sistema é executado, com o código gerado pelo compilador utilizado para produzir o sistema, com os algoritmos utilizados nos processos internos do sistema e com a qualidade das suas respectivas implementações.

Alguns SOTR, como o FREERTOS descrito em (BERRY, 2016), fornecem ao desenvolvedor apenas um conjunto mínimo de funcionalidades: suporte a multitarefa, escalonador com diversas políticas de escalonamento, *mutex*, semáforos e temporizadores. Estes pequenos SOTR, também chamados de núcleos de tempo real ou *real time kernel*, são pensados para serem pequenos, velozes e capazes de implementar políticas de tempo bastante rígidas. Suas características se aplicam muito bem a sistemas embarcados baseados em microcontroladores e que trabalham com restrições temporais bastante rígidas.

Os grandes sistemas operacionais como: Linux, Windows Mac OS X, oferecem outros recursos mais avançados: serviços de entrada e saída de dados, proteção de memória, sistemas de arquivos, políticas de segurança, interface com o usuário etc. Estes recursos tornam possível ao desenvolvedor construir e executar aplicações mais complexas que proporcionam um maior nível de segurança e interação com outros sistemas e usuários. Nativamente estes sistemas não foram projetados para atender aos requisitos dos STR, porém seus desenvolvedores vem desenvolvendo alternativas para este fim como as diferentes versões do Windows Embedded e os *patchs* disponibilizados para Linux: Preempt_RT e RTAI. Vale lembrar que muitos dos sistemas em que se baseiam alguns SOTR, como Linux, já são utilizados em várias aplicações de missão crítica, além de que sistemas de controle baseados em PC são uma realidade na indústria como complemento e até substitutos aos tradicionais CLPs como os (*Siemens PC-based Automation*). Estes sistemas permitem uma maior flexibilidade na programação, expansão e configuração de software e hardware.

2.2.1 Latência

Segundo o dicionário (PRIBERAM, 2017), latência é o tempo decorrido entre o estímulo e a resposta correspondente.

No caso de sistemas computacionais e mais especificamente em SO, um estímulo, pode ser externo, que necessita de uma resposta do sistema, ou interno, como uma *thread* que foi colocada no estado "pronto"e espera para ser executada.

Assim como todos os sistemas reais, SOTR, estão sujeitos a latências que surgem como consequência do seu próprio funcionamento e do hardware sobre os quais executam. O conhecimento dos valores de latência e principalmente sua constância, mesmo que em um cenário de sobrecarga do sistema, são primordiais na garantia da previsibilidade. O conhecimento dos valores de latência também são de vital importância na seleção de um SOTR que seja capaz de atender aos requisitos temporais de uma aplicação. Conforme (HART, 2007), são causas comuns de latência em SO: latência de interrupção, latência de escalonamento, latência por inversão de prioridade, latência por inversão de interrupção.

A latência de interrupção corresponde ao tempo decorrido entre a ocorrência de uma interrupção e o momento em que é atendida. Há também o tempo entre o atendimento da interrupção e a rotina que realmente processará o sinal recebido. O tempo decorrido entre o despertar de um processo de alta prioridade e a sua execução as vezes podem ser consideradas latência de interrupção, uma vez que o seu despertar muitas vezes ocorre devido a algum evento externo.

A latência provocada pelo escalonamento, é o tempo entre o instante em que uma tarefa de alta prioridade acorda (tempo de liberação) e o momento em que ela inicia a execução (tempo inicial).

As latências por inversão de prioridade e inversão de interrupção consistem no tempo que uma *thread* com prioridade alta espera para utilizar algum recurso em uso por uma *thread* de prioridade baixa, seja ela associada a uma outra tarefa ou, no caso da interrupção, a um manipulador de interrupções. Diferente da inversão de prioridade e inversão de interrupção não pode ser antecipada visto que a ocorrência da maior parte das interrupções não pode ser prevista.

2.2.2 Linux Para Tempo Real

As modernas aplicações de tempo real estão muito mais conectadas e iterativas, isso exige a implementação de novas funcionalidades, como interfaces gráficas, comunicação com serviços web e banco de dados. Essas funcionalidades além de um grande reuso

de código, também fazem necessário um melhor suporte dos sistemas operacionais sobre as quais executam. Os SOTR tradicionais e os núcleos de tempo real, embora tenham um ótimo desempenho no cumprimento de metas temporais, geralmente, oferecem pouco ou nenhum suporte aos recursos utilizados nas aplicações mais modernas.

Na busca por melhor suporte as aplicações, vários projetos tomaram a iniciativa de incorporar funcionalidades de tempo real a SOPG. Devido ao seu código fonte aberto, sua comprovada robustez em aplicações críticas e sua portabilidade entre diferentes plataformas de hardware, o Linux, tornou-se um dos sistemas mais utilizados nas conversões para tempo real. Porém o Linux é concebido para uso e computadores pessoais e servidores e por este motivo seu *kernel* é otimizado para obter um melhor desempenho global e alocar recursos de forma justa para todos os processos em execução por meio do seu escalonador Completely Fair Agendador (CFS).

Dentre os projetos mais ativos, no trabalho de transformar o Linux em um SOTR, podemos citar: RTAI, Xenomai e Preempt_RT. Cada um desses projetos implementa uma versão modificada do *kernel*, com arquitetura própria, vantagens e desvantagens. Neste trabalho as soluções estudadas e avaliadas são o RTAI e o *patch* Preempt_RT.

2.2.3 O *Patch* PREEMPT_RT

O *patch* Preempt_RT é a solução de tempo real oficialmente suportada pelo *kernel* Linux. Este projeto é a solução mais bem sucedida no esforço para transformar o Linux em um SOTR sem o auxilio de um *microkernel*.

As modificações impostas pelo *patch* Preempt_RT ao *kernel* incluem, a transformação de alguns manipuladores de interrupção em *threads* de baixa prioridade (manipuladores de interrupção importantes para o funcionamento do sistema como um todo, como o temporizador, são mantidas com prioridade máxima), substituição das *spin_ locks* por *mutexes* para permitir que as regiões críticas do *kernel* passem a ser preemptíveis, herança de prioridade. Algumas alterações não são mais necessárias pois foram incorporadas ao ramo principal do *kernel*, como temporizadores de alta resolução e *mutexes* no espaço de usuário.

As funcionalidades do sistema de forma geral permanece a mesma para gerenciamento, comunicação e sincronia entre *threads*, com exceção de três novos modos

de escalonamento, *First In First Out* (FIFO) orientado a prioridades (em que 99 é a prioridade máxima), *Round Robin* (RR), e *Earliest Deadline First* (EDF).

2.2.4 O RTAI

O RTAI, acrônimo de *Real-time Application Interface*, surgiu como uma variação do antigo *RTLinux* desenvolvida pelo *Dipartimento di Ingeneria Aerospaziale* da Universidade *Politecnico di Milano*.

Sua abordagem para a transformação do Linux em um SOTR utiliza um *microkernel* em paralelo ao *kernel* do Linux que é responsável pela execução das tarefas de tempo real, enquanto o *kernel* fica responsável pela execução das tarefas de baixa prioridade, está solução é comumente chamada de *Dual Kernel*. Neste sistema o gerenciamento do hardware fica a cargo de uma camada de abstração criada abaixo dos *kernels*, chamada *ADEOS* (*Adaptive Domain Environment for Operating Systems*). Esta camada é quem permite dois núcleos compartilhare o mesmo hardware simultaneamente. A arquitetura implementada pelo sistema é ilustrada na figura 2.3.

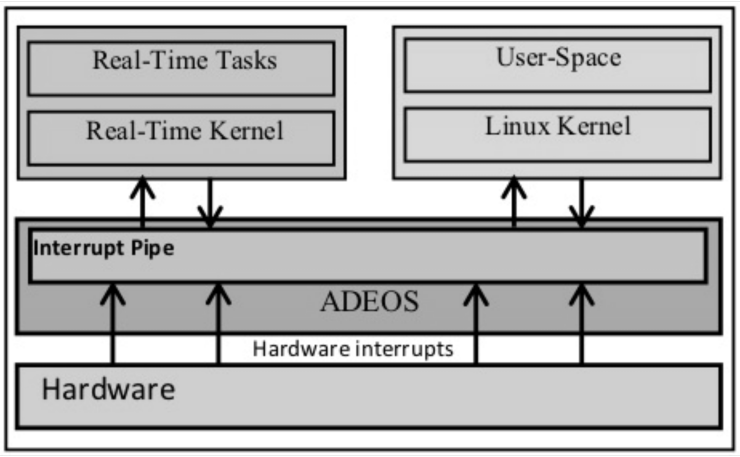

Figura 2.3: Arquitetura *Dual Kernel* adotada pelo RTAI, (SAOUD, 2011)

O *microkernel* implementado pelo RTAI, a semelhança dos NTR, suporta todo o conjunto básico de funcionalidades para a construção de ATR, como rotinas para a criação, destruição, suspensão, sincronia e comunicação entre tarefas, temporizadores com alta resolução, políticas de escalonamento, capacidade de execução de tarefas de tempo real no espaço do usuário e mecanismos para controle de recursos compartilhados. Seu

escalonador suporta políticas *Rate Monotonic* (RM), EDF, FIFO orientado a prioridades (em que 0 é a maior prioridade) e RR.

3 Metodologia

A avaliação de um SOTR é guiada principalmente pela verificação da cidade de suas características atenderem aos requisitos de um determinado projeto, o que pode envolver diversas variáveis que influenciam o desempenho do sistema em várias circunstâncias diferentes. Diferentes características relacionadas a requisitos não funcionais de uma aplicação também podem ter peso maior ou menor na avaliação de um SOTR, características como: suporte e reputação dos desenvolvedores, documentação, custo, integração com sistemas legados, suporte a hardwares específicos etc, corroboram com o número de fatores que podem tornam a comparação entre SOTRs um processo complexo.

As avaliações de desempenho de SOTR mais completas, normalmente são baseadas na observação do sistema como aplicação destinada a fins específicos. Estas avaliações são difíceis de generalizar e portar para outras soluções e que possuam arquitetura de destino diferente da proposta nos testes originais. Pesquisas vem sendo desenvolvidas na tentativa de criar métodos genéricos de avaliação com a escolha de parâmetros quantitativos e qualitativos que sejam comuns a maior parte dos sistemas de tempo real, e que estejam diretamente relacionados aos principais casos em que se aplicam. Estes métodos de avaliação são classificados quanto ao grau de detalhamento com que observam os sistemas testados, segundo (SCHWAN, 1994), em: *Fine-grained benchmarks*, *Application-oriented benchmarks* e *Simulation-based evaluations*.

Os *Fine-grained benchmarks* analisam a execução de STR observando suas características de baixo nível, analisando o desempenho do conjunto hardware e software, este tipo de análise confere excelente precisão aos resultados obtidos porém requer um alto grau de conhecimento sobre o funcionamento do hardware assim como acesso ao hardware por meio de instrumentos de medição específicos. *Simulation-based evaluations*, utilizam a execução de modelos, com um grau de detalhes considerado suficiente para a aplicação estudada, do sistema avaliado. Este método possui a vantagem de poder executar testes em sistemas, tant de hardware como software, que ainda não estejam completamente implementados, no entanto os resultados obtidos possuem um grau de precisão proporcional a qualidade do modelo avaliado e a quantidade de variáveis, que alteram o funcionamento do sistema, implementadas. *Application-oriented benchmarks*,

observam a execução de STR por meio de parâmetros de alto nível como: cumprimento de *deadlines*, tempo de execução das tarefas e latências, relacionados a uma aplicação sintética com requisitos relacionados a execução semelhantes ao de aplicações reais. Este método de análise tem como principal vantagem observar o comportamento do conjunto hardware e software, porém sem que seja possível atribuir com precisão as causas dos valores medidos.

3.1 Modelagem dos Teste

Este trabalho utiliza uma abordagem de testes do tipo *Application-oriented benchmarks*, concebidos para verificar a capacidade do *patch* Preempt_RT e do RTAI em transformarem um sistema Linux de propósito geral em um SOTR a partir da medida dos valores de latência, do tempo de computação das tarefas executadas e da verificação do cumprimento dos deadlines estabelecidos.

O modelo de *benchmark* implementado foi baseado na solução proposta por (MOREIRA, 2007) junto com o algoritmo de medição de latências do programa *Cyclictest* (FOUNDATION, 2017). Em seu trabalho, (MOREIRA, 2007), propõe duas aplicações de teste que une as definições e conceitos operacionais dos *benchmarks MiBench* (BROWN, 2001) e *Hartstone* (KAMENOFF, 1992). Em ambas as aplicações é proposto que cada uma das tarefas que as compõe execute uma função dentre as sugeridas por *benchmarks MiBench* e (MOREIRA, 2007) para simulação de aplicações automotivas e de controle industrial, embora vários produtos de consumo e de uso hospitalar também implementem funções semelhantes. As funções implementadas neste trabalho foram:

- Multiplicação de duas matrizes 5 x 5
- Ordenação de 20 inteiros usando *quicksort*
- Transformação de graus para radianos
- Cálculo da raiz quarada de inteiros utilizando séries de Taylor
- Cálculo polinomial cúbico

Dentre as duas aplicações teste desenvolvidas, a primeira utiliza a definição do *benchmark Hartstone* para a Série-PH, tarefas periódicas e harmônicas, e simula o

comportamento do que poderia ser um programa responsável pelo monitoramento de um conjunto de sensores com taxas de amostragem diferentes e sem a intervenção de interrupções ou do usuário. Na aplicação são implementadas cinco *threads* periódicas com frequência igual a um múltiplo de todas as outras frequências maiores, no caso dos testes implementados as frequências foram as mesmas sugeridas nas definições do *Hartstone*: 1Hz, 2Hz, 4Hz, 8Hz e 16Hz. Cada *thread* executa uma das funções descritas anteriormente, o *deadline* para a execução das tarefas foi definido como sendo igual ao seus respectivos períodos. A prioridade de cada tarefas foi a mesma para todas.

A segunda aplicação executa, além das cinco *threads* da primeira, mais duas *threads* aperiódicas, simulando aplicações que precisam responder a eventos provocados por interrupções, conforme a definição do *benchmark Hartstone* para Série-AH. O intervalo de ativação das *threads* aperiódicas foi gerado de forma aleatória dentro de um intervalo de 20ms a 40ms, seus *deadlines* foram definidos em 20ms, e a função executada por elas foi a conversão de graus para radianos.

Além das funções listadas acima, as tarefas, tanto periódicas quanto aperiódicas, foram responsáveis pela execução das medições das suas próprias latências, tempo de computação e verificação do cumprimento de *deadline*.

3.2 Implementação dos Testes

Os testes foram implementados em linguagem C, linguagem para a qual são fornecidas as bibliotecas tanto do RTAI quanto as chamadas de sistema e bibliotecas fornecidas pelo Linux, utilizadas com o Preempt_RT.

A construção dos programas foi realizada seguindo algumas premissas sobre os sistemas que executariam as aplicações de teste. Foi levado em conta o suporte a temporizadores de alta resolução, funções com tempo de execução previsíveis, isolamento entre as tarefas, execução das aplicações em modo usuário, suporte a mecanismos que evitem paginação, cache e memória em disco. O atendimento de todas estas premissas foi feito por ambas as soluções, seja pela utilização de funções do próprio sistema Linux, seja pelo provimento de uma biblioteca específica.

Por necessidade das aplicações desenvolvidas para RTAI, que precisam utilizar funções de tempo real fornecidas por bibliotecas próprias e não podem executar chamadas

de sistema dentro do código de tempo real, foram desenvolvidos quatro programas de teste: Série-PH e AH para Preempt_RT e Série-PH AH para RTAI.

Todos os programas seguiram a mesma sequência básica de funcionamento:

1. Alocação de memória
2. Travamento das posições de memória atuais e futuras, para que permaneçam na memória *RAM*
3. Configuração das *threads* como tarefas de tempo real, com a definição da prioridade e do algoritmo de escalonamento (FIFO).
4. Execução das *threads* e suas respectivas computações

Também foram implementados nos programas de teste, um mecanismo que permite a todas as *threads* de tempo real iniciarem a execução o mais próximas possível e um mecanismos para a impressão de histogramas contendo a distribuição dos resultados de medição das latências.

3.3 O Ambiente de Testes

Na comparação entre diferentes SO, e mais especificamente de SOTR, é importante que a configuração do hardware utilizado nos testes propostos seja igual ou no mínimo equivalente, isso garante que os resultados obtidos sejam consistentes e que não tenham sido influenciados por funcionalidades específicas de uma determinada configuração de hardware.

O hardware utilizado para testar as duas soluções de tempo real escolhidas foi um netbook Acer, modelo Aspire One D250-1023, processador com arquitetura x86, Intel Aton N270, *clock* de 1,60GHz, memória cache L2 de 512KB, 1GB de memória DDR2-533, disco rígido de 320GB SATA.

Antes da escolha de um hardware para a execução de qualquer uma das soluções estudadas é importante observar algumas particularidades relacionadas a possíveis conflitos entre as configurações do *kernel* e o hardware que foram verificadas neste trabalho e explicadas mais adiante.

Ambas as soluções de tempo real testadas usam como base o sistema operacional Linux e a distribuição escolhida foi Debian 8.8 (Jessie) para processadores de 32 bits. A distribuição Debian foi escolhida, dada a facilidade de se produzir um sistema com funcionalidades reduzidas, sua ampla documentação, sua grande coleção de pacotes contendo programas e bibliotecas pré compilados e por ser a base de inúmeras outras distribuições que se aplicam de servidores a sistemas embarcados.

Foi considerado de grande importância produzir *kernels* com configurações idênticas, com obvia exceção às opções específicas exigidas por cada uma das soluções, para que o máximo de recursos não alterassem o desempenho dos sistemas de forma a favorecer alguma das soluções testadas. As configurações utilizadas tiveram como ponto de partida a configuração *vanilla* de cada *kernel*. A versão utilizada do *patch PREEMPT_RT* foi a 4.4.17-rt25 publicada em 25 de agosto de 2016, aplicado sobre um *kernel*, *vanilla*, versão 4.4.17. A versão testada do RTAI foi a 5.0.1 publicada em 15 de maio de 2017, o *patch HAL* foi aplicado em um *kernel*, *vanilla*, versão 4.4.43. Vale mencionar que não existem versões do *kernel* que sejam suportadas por ambas as soluções simultaneamente.

3.4 Produzindo um *kernel* de tempo real

Para produzir um sistema Linux de tempo real utilizando uma das soluções estudadas neste trabalho não é necessário nenhuma truque especial. Seguem-se todos os passos de construção de um *kernel* de proposito geral (configuração, compilação, instalação dos módulos e instalação do *kernel*), porém é necessária uma atenção especial durante o processo de configuração.

A construção de um SOTR baseado em Linux utilizando as duas soluções estudadas se inicia com a aplicação dos respectivos *patchs* ao *kernel*. Estes *patchs* alteram algumas características e adicionam novas funcionalidades ao *kernel*.

As principais mudanças promovidas pelo *patch* Preempt_RT é tornar o *kernel* completamente preemptível, inclusive em sua região crítica, ao substituir as *spin_locks* por *mutexes*, transformar os tratadores de interrupção, com exceção de interrupções vitais para o sistema como a dos temporizadores, em *threads* com prioridade menor que as tarefas de tempo real, evitando assim o problema da latência provocada pela inversão de interrupção, e a implementação de uma política de herança de prioridade, minimizando

o problema da inversão de prioridade.

Já o *patch HAL*, fornecido junto com o RTAI, é menos invasivo e apenas adiciona ao *kernel* suporte a camada de abstração de hardware *ADEOS*. Esta camada de abstração permite que dois *kernels* funcionem simultaneamente compartilhando o mesmo hardware.

3.4.1 Configuração do *Kernel*

Como foi dito anteriormente este processo requer uma atenção especial, pois é nesta etapa que identificaremos as funcionalidades do *kernel* que provocam alterações nos valores de latência, por vezes os deixando imprevisíveis. Para auxiliar neste processo foi utilizado o programa *Cyclictest* (FOUNDATION, 2017).

O *Cyclictest* é um programa fornecido junto a suíte de teste *rt-tests* que por sua vez é fornecida e oficialmente mantida pelo grupo de desenvolvimento do *kernel*. Sua principal função é a medição do conjunto de latências provocadas pelo sistema ou por outros processos a que estão submetidas um conjunto de tarefas. *Cyclictest* proporciona uma poderosa funcionalidade que detecta processos executados pelo sistema que provoquem distúrbios nos valores de latência.

As várias execuções do programa *Cyclictest* mostraram que os recursos do *kernel* voltados ao gerenciamento de energia foram os que mais interferiram nos valores de latência. Recomenda-se que serviços como: *Advanced Configuration and Power Interface* (ACPI), CPU Frequency scaling e CPU Idle, sejam desabilitados. É importante notar que isso é uma grande restrição ao uso das soluções estudadas em sistemas alimentados por bateria.

Algumas outras funcionalidades forma desabilitadas e habilitadas tendo como princípio exigências e características das soluções de tempo real abordadas como: temporizadores com alta resolução, carregamento de módulos etc. Outras foram alteradas afim de produzir um *kernel* sem otimizações que privilegiassem algum tipo de arquitetura de hardware específica, numa tentativa de tornar o sistema o mais genérico possível.

A habilitação e remoção de recursos do *kernel*, podem produzir efeitos indesejados e conflitos com algumas configurações de hardware. Alguns *notebooks* não suportam a inicialização de um *kernel* sem suporte a ACPI ou a incompatibilidade do RTAI com o

suporte a recursos usados por alguns processadores da AMD inviabilizam o uso do sistema nesta plataforma. Infelizmente nenhuma das duas soluções possui alguma documentação sobre o hardware suportado o reforça o fato dos testes serem imprescindível.

3.4.2 Instalação do RTAI, compilação e execução de programas

Após a configuração, compilação e instalação do *kernel* de tempo real, diferente do Preempt_RT, o RTAI precisa ser configurado e instalado. O processo é semelhante a instalação de programas em Linux a partir do código fonte.

Outra exigência do RTAI é que para a compilação e execução de programas é necessário definir as variáveis *CFLAGS* e *LDFLAGS*, estas variáveis são utilizadas no processo de compilação e definem os locais onde se encontram as bibliotecas utilizadas na construção dos programas. Para que os programas possam ser executados os módulos do RTAI precisam ser carregados, como os módulos são possuem um hierarquia de dependência é preciso carrega-los em uma ordem específica. Este processo foi automatizado por meio de um *script* para *shell*.

3.5 Execução dos Testes

Os testes forma executados sem maiores problemas e os sistemas responderam de forma adequada, sem quebras ou travamentos.

Para que os resultados pudessem servir como valores de referência e para que as reais capacidades dos sistemas testados pudessem ser avaliadas, foram executados dois procedimentos para sobrecarregar os recursos da maquina de testes conforme o proposto por (LIM, 2017). Um dos procedimentos, que consiste na execução infinita do comando *ping* o que produz uma utilização de 100% do processador, o outro processo foi uma compilação do *kernel* Linux, que além de provocar uma sobrecarga no processamento, também provoca um intenso uso de memória *RAM* e do disco rígido. Para que os valores medidos fossem consistentes os testes foram executados por um período de duas horas.

Após a execução foram plotados gráficos dos histogramas gerados, contendo os valores de latência medidos para cada uma das *threads* executada, e os valores máximos de computação de cada tarefa executada foram compilados em uma tabela.

4 RESULTADOS

Os testes para *Serie-PH* nos mostram intervalos de latência bastante consistentes com diferença entre os extremos suficientemente restrita, na maior parte das tarefas executadas tanto no Preempt_RT quanto no RTAI (figura 4.1), com exceção da *thread* 4 do teste executado no RTAI na qual fica evidente a existência de uma anomalia, e que ainda não teve sua causa identificada. Embora estejam distribuídos de forma adequada, os valores máximos de latência, em alguns casos, superam 100% do tempo de computação máximo das tarefas (tabela 4.1) o que pode ser um grande problema para tarefas com *deadlines* na casa dos microssegundos, porém para as tarefas executadas, a soma dos valores de Latência e Tempo de Computação foram bem inferior aos deadlines definidos.

Quando adicionamos duas tarefas aperiódicas aos testes (Série-AH) tivemos alguns comportamentos interessantes (figura 4.1). A execução das tarefas pelo *patch* Preempt_RT a primeira vista se mostrou inalterada, mas uma análise detalhada dos valores de latência mostram alguns pontos fora da curva e registros de latência máxima bem superiores a maioria das medições, embora os valores não tenham comprometido a execução da aplicação, a soma dos valores de latência e tempo de computação ainda foram bem inferiores ao *deadline*, esse tipo de comportamento reforça a necessidade de testes de medição de latência com a aplicação pretendida.

Já o RTAI teve um comportamento que parece adiar a execução das tarefas aperiódicas ao longo do tempo, embora os valores tenha estado dentro de um intervalo definido e as curvas serem muito parecidas. Podemos nos questionar se a adição de novas tarefas aperiódicas provocaria o aumento das latências destas tarefas.

Mais uma vez os valores somados da latência com o tempo de computação das tarefas estejam bem abaixo dos valores de seus deadline, o tempo de computação das tarefas executadas pelo Preempt_RT foram bastante elevados, enquanto no RTAI praticamente não foram alterados.

	Preempt_RT		RTAI	
Thread	Latência Máx.	Latência Mín	Latência Máx.	Latência Mín
0	39	13	41	28
1	43	8	38	22
2	30	8	44	27
3	25	7	40	24
4	44	7	74	20

Tabela 4.1: Valores (em µs) máximos e mínimos de latência obtidos nos testes *Serie-PH*- Preempt_RT x RTAI

	Preempt_RT		RTAI	
Thread	Latência Máx.	Latência Mín	Latência Máx.	Latência Mín
0	70	9	46	32
1	72	8	39	25
2	71	7	43	28
3	74	7	39	21
4	67	7	45	23
Ap. 0	80	6	72	52
Ap. 1	71	7	59	37

Tabela 4.2: Valores (em µs) máximos e mínimos de latência obtidos nos testes *Serie-AH*- Preempt_RT x RTAI

	Preempt_RT		RTAI	
Thread	Tc (*Serie-PH*).	Tc (*Serie-AH*)	Tc (*Serie-PH*)	Tc (*Serie-AH*)
0	18	70	19	21
1	38	81	45	45
2	39	83	37	36
3	27	70	28	20
4	39	79	43	40
Ap. 0	-	62	-	42
Ap. 1	-	56	-	44

Tabela 4.3: Valores (em µs) do tempo de computação máximo obtidos nos testes *Serie-PH* e *Serie-AH* - Preempt_RT x RTAI

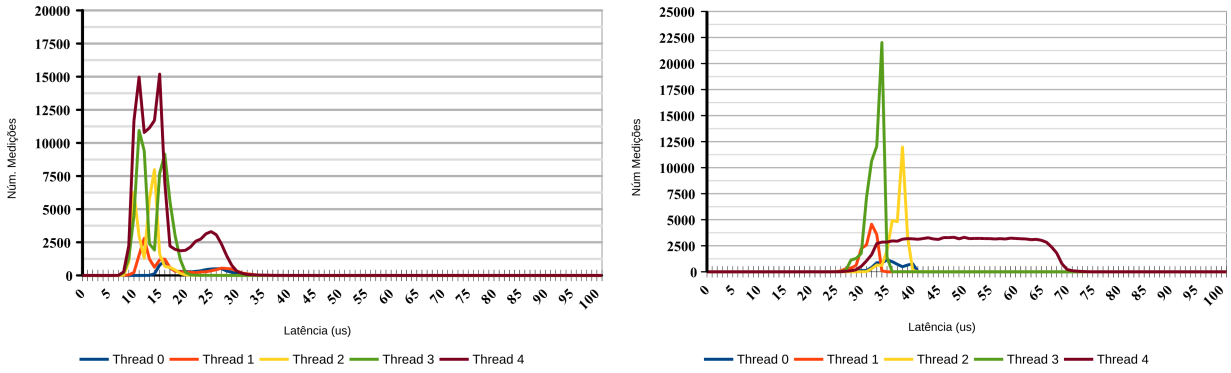

Figura 4.1: Medidas de latência dos testes *Serie-PH* - Preempt_RT x RTAI

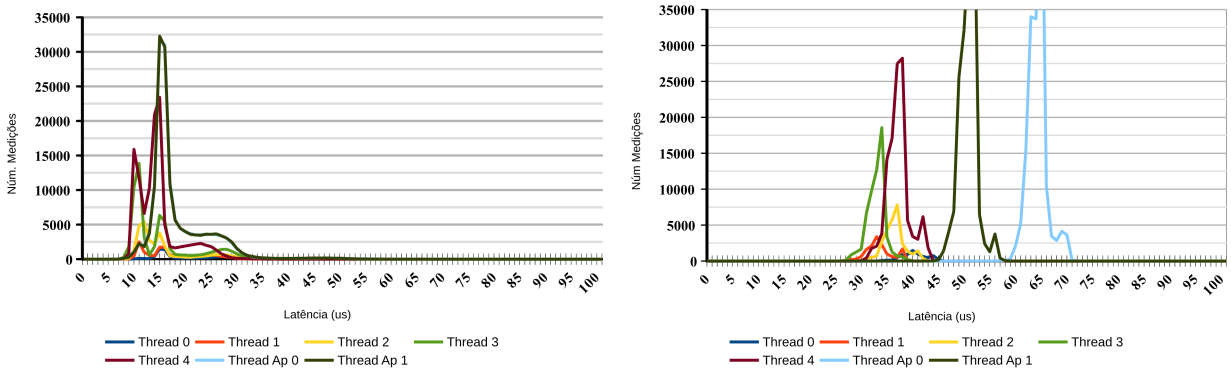

Figura 4.2: Medidas de latência dos testes *Serie-AH* - Preempt_RT x RTAI

5 CONCLUSÕES

Este trabalho apresentou uma análise quantitativa do desempenho de duas soluções, *patch* Preempt_RT e RTAI, que transformam um sistemas Linux de propósito geral em um SOTR. Os resultados desta análise, os programas de teste desenvolvidos e a documentação gerada contribuem com informações valiosas para projetistas de STR e estudantes no momento de comparar outros SOTR com os sistemas estudados assim como base para a criação de ATR utilizando Linux.

A análise dos valores medidos para latências a que as tarefas de tempo real estão sujeitas e dos seus respectivos tempos de computação nos mostram que tanto o Preempt_RT quanto o RTAI podem executar com segurança, tarefas de tempo real com restrições temporais na casa dos milissegundos e, nos casos de tarefas exclusivamente periódicas, centenas de microssegundos.

Além dos resultados obtidos com os testes, algumas conclusões sobre a utilização do *patch* Preempt_RT foram:

- Facilidade de instalação
- Simplicidade na criação de aplicações
- Boa documentação, atualizada e organizada
- Para algumas aplicações o aparecimento de valores espúrios de latência podem comprometer seu uso

Quanto ao RTAI pode-se dizer que:

- Instalar e usa-lo pela primeira vez pode ser um tormento
- A documentação e escassa, dispersa e desatualizada
- Algumas de suas funcionalidades são divulgadas pelos desenvolvedores, mas não estão documentadas
- Sua arquitetura lhe confere eficiência, mas pouca integração com o sistema

- Chamadas de sistema tornam a execução de tarefas imprevisível
- A maior consistência no seus valores de latência permitem produzir sistemas mais previsíveis

5.1 Trabalhos Futuros

Seria bastante desejável comprovar a eficiência do Preempt_RT e RTAI por meio de um prova de conceito em uma aplicação prática como em um sistema de controle. Como um dos algoritmos de escalonamento para tarefas de tempo real, o EDF é suportado tanto pelo Preempt_RT quanto pelo RTAI, testar a eficiência dos sistemas utilizando este algoritmo seria de grande importância. Com a popularização de processadores com múltiplos núcleos torna inevitável o estudo do comportamento de SOTR nessas plataformas. Como também se tronaram algo popular, seria de grande interesse estudar o comportamento de um sistema Linux de tempo real em plataformas utilizadas em dispositivos embarcados baseadas em processadores ARM. Avaliar a necessidade e a possibilidade de executar o Linux com a aplicação do *patch* Preempt_RT e do RTAI simultaneamente com o objetivo de tentar sanar deficiências de ambas as soluções.

Bibliografia

BERRY, Richard. **Mastering the FreeRTOS Real Time Kernel**. [S.l.]: Real Time Engineers Ltd., 2016.

BROWN, Richard B. MiBench: A free, commercially representative embedded benchmark suite. **Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on**, p. 3–14, 2001.

EMDE, Carsten. Long-term monitoring of apparent latency in PREEMPT RT Linux realtime systems. **RTLWS12**, 2010.

FARINES, Jean-Marie. **Sistemas de Tempo Real**. [S.l.]: Departamento de Automação e Sistemas da Universidade Federal de Santa Catariana, 2000.

FOUNDATION, Linux. **Cyclictest**. 2017. Disponível em: <https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest>. Acesso em: 3 ago. 2017.

GITE, Vivek. **How to Compile and Install Linux Kernel v4.5 Source On a Debian / Ubuntu Linux**. 2015. Disponível em: <https://www.cyberciti.biz/faq/debian-ubuntu-building-installing-a-custom-linux-kernel/>. Acesso em: 12 abr. 2017.

HALLBERG, Andréas. **Time Critical Messaging Using a Real-Time Operating System**. 2017. Mestrado – Chalmers University of Technology e University of Gotherburg.

HART, Darren V. Internals of the RT Patch. **Proceedings of the Linux Symposium**, 2007.

KAMENOFF, Nick I. Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems. **Real-Time Systems**, Springer, v. 4, n. 4, p. 353–382, 1992.

KERRISK, Michael. **The Linux programming interface : a Linux and UNIX system programming handbook**. [S.l.]: No Starch Press, 2010.

KOPETZ, Hermann. **Real-Time Systems - Design Principles for Distributed Embedded Applications**. [S.l.]: Kluwer Academic Publishers, 2002.

LAPLANTE, Phillip A. **Real-Time Systems Design and Analysis**. [S.l.]: Wiley, 2012.

LIM, Geunsik. **Worst Case Latency Test Scenarios**. 2017. Disponível em: <https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/worstcaselatency>. Acesso em: 3 ago. 2017.

MANTEGAZZA, Paolo. **RTAI 3.4 User Manual rev 0.3**. [S.l.]: October, 2006.

MHALA, N. N. Comparison of Open Source RTOSs Using Various Performance Parameters. **International Journal of Electronics Communication and Computer Engineering**, v. 4, n. 2, p. 86–91, 2013.

MOREIRA, Andeson Luiz Souza. **Análise de Sistemas Operacionais de Tempo Real**. 2007. Mestrado – Universidade Federal De Pernambuco.

PRIBERAM. **Priberam da Língua Portuguesa**. 2017. Disponível em: <https://www.priberam.pt/dlpo/>. Acesso em: 17 out. 2017.

SAOUD, Slim Ben. Impact of the linux real-time enhancements on the system performances for multi-core intel architectures. **International Journal of Computer Applications**, v. 17, n. 3, 2011.

SCHWAN, Karsten. **A Survey of Real-Time Operating Systems**. [S.l.], 1994.

SOUSA, Cristóvão. RTAI Tutorial. **Combra University**, 2009.

TAN, Su-Lim. Real-time operating systems for small microcontrollers. **IEEE micro**, IEEE, v. 29, n. 5, 2009.

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 3ª edição. [S.l.]: Pearson Prentice Hall, 2009.

A APÊNDICE

Para adicionar outros apêndices ou anexos basta usar adicionar capítulos a este arquivo.

A.1 Apêndice 1

1. Item 1