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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed imperdiet lacus consectetur vestibulum scelerisque. Integer accumsan odio nisi, sed aliquet quam consequat tincidunt. Ut at sollicitudin felis. Duis tempor condimentum velit ac molestie. Donec vitae luctus velit, vitae faucibus mi. Suspendisse potenti. Mauris accumsan mi quis neque aliquet ultricies.

Suspendisse porta ultricies turpis, id porta risus. Sed nec bibendum ligula. Praesent sapien tortor, condimentum ut interdum quis, ornare ac nisi. Phasellus blandit ipsum ac mollis porta. Nunc egestas elementum est, sit amet hendrerit leo finibus placerat. Etiam finibus lorem quis dolor pellentesque, eu hendrerit nisl rhoncus. Aenean a mi consectetur, efficitur justo non, aliquam risus. Sed feugiat nisl vitae venenatis pulvinar. Suspendisse molestie quis tellus eget fringilla. Nunc at posuere tortor. Sed venenatis dui risus, non congue magna vehicula eu.

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.

1.1 Justificativa

1.2 Objetivos

1.2.1 Gerais

1.2.2 Específicos

- Item 1
- Item 2
- Item 3

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, (HART, 2007). 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 deve 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 rigor com que trata os 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 desempenho do sistema mais sem provocar falhas graves. Geralmente nesses sistemas os prejuízos provocados pela degradação do sistema são compensados pelo benefícios do sistema em sua operação normal.

Um sistema de tempo real é rígido quando a falta no cumprimento de uma única restrição temporal pode levar o sistema 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 do sistema de forma crítica, porém a extrapolação do limite de prazos estabelecidos 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**) 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 preceitos 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 do sistema, sem grandes consequências.

As tarefas de tempo real, também são classificadas quanto a periodicidade 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.

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 é colocada como pronta para a execução. 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. 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 estão ilustrados na figura 1 (ver como colocar uma figura em Latex).

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. No caso dos sistemas avaliados nesse trabalho a implementação dos algoritmos de escalonamento e suas respectivas políticas são implementadas pelo sistema operacional e serão explicadas adiante.

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, FIFO (*First In First Out*) orientado a prioridades (de 0 a 99), RR (Round Robin) com fatiamento de tempo igualitário entre as tarefas, e EDF (*Earliest Deadline First*).

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 utilizarem o mesmo hardware. A arquitetura implementada pelo sistema é ilustrada na figura 2.

O RTAI 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 politicas *Rate Monotonic* (RM), EDF, FIFO orientado a prioridades e RR.

3 Metodologia

A avaliação de um SOTR é definida principalmente pela capacidade de suas características atenderem aos requisitos de um determinado projeto, o que pode envolver diversas variáveis que influenciam o desempenho do sistema em grande número circunstâncias 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. A escolha de parâmetros quantitativos que sejam comuns a maioria dos sistemas de tempo real, e que estejam diretamente relacionados a execução dos principais casos em que estes sistemas se aplicam, facilita a comparação entre as diversas soluções existentes e proporcionam uma excelente forma de avaliar SOTR.

3.1 Parâmetros e premissas utilizados na Avaliação

No sentido aplicado nesse texto o termo latência pode ser definido como: "Tempo decorrido entre o estímulo e a resposta correspondente. "Latência", em Dicionário Priberam da Língua Portuguesa [em linha], 2008-2017, https://www.priberam.pt/dlpo/Latência[cons em 21-12-2017]. 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 e determinismo dos SOTR. 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.

"Isso é uma reflexão minha!"Do mesmo modo que trata outros recursos de hardware, um SOTR deve ser capaz de abstrair e gerenciar de forma adequada as latências decorrentes de suas interações com o hardware, de modo que os valores de latência apresentados ao desenvolvedor sejam o mais consistente possível dentro de um intervalo de valores bem definido, tornando o ambiente de desenvolvimento homogêneo e independente de plataforma.(???)

As causas de latência em um SOTR que mais influenciam a execução de ATR estão relacionados a:

3.2 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.

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 exceção das opções específicas exigidas por cada uma das soluções, para que recursos específicos não alterassem o desempenho dos sistemas de forma a favorecer uma das soluções testadas. As configurações utilizadas tiveram como ponto de partida a versão *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.

3.2.1 Instalação dos Sistemas e Testes Preliminares

As soluções estudadas foram submetidos a testes preliminares, utilizados tanto para identificar possíveis falhas nos processos de instalação, como para identificar funcionalidades do *kernel* que pudessem alterar o desempenho e o determinismo do sistema. Nestes testes o principal parâmetro observado foi a latência (QUAL LATÊNCIA!!!) do sistema. Embora a redução de latência (QUAL!!!), como dito anteriormente, não seja um dos principais objetivos de um SOTR, é de vital importância, junto com os outros parâmetros vistos anteriormente, que seus valores sejam conhecidos para que o sistema possa ser classificado como determinístico. Os valores de latência obtidos como resultado dos testes preliminares também serviram como referência para avaliar a qualidade e a uniformidade dos resultados obtidos com os benchmarks desenvolvidos neste trabalho.

Os testes preliminares foram executados por meio de ferramentas recomendadas e fornecidas pelos próprios desenvolvedores dos sistemas avaliados. Foram utilizados os programas: *Latency*, para testes executados no *RTAI* e *Cyclictest*, para testes executados no *kernel* com o patch *PREEMPT-RT* aplicado. O algoritmo de medição do programa *Cyclictest* foi utilizado como base para os testes desenvolvidos neste trabalho.

—-Falar sobre o programa Latency —-

O programa *Cyclictest* é fornecido junto a suíte *rt-tests*, um conjunto de ferramentas para teste de sistemas de tempo real desenvolvidas e mantidas pelos desenvolvedores do *kernel Linux* e hospedada no próprio repositório do *kernel*.

O programa *Cyclictest* mede com alto grau de precisão, os resultados são fornecidos em microssegundos, a latência do sistema para um número definido de tarefas. Mostrou-se de extrema utilidade seu recurso que possibilita o rastreio de funcionalidades do *kernel* que provocam o aumento da latência do sistema, por meio da função *FTRACER*. Este recurso foi utilizado para produzir uma configuração adequada do *kernel linux*. Para que os valores das medições, obtidos com os testes, sejam válidos, é preciso que os testes

sejam executados diversas vezes por um período de tempo suficiente longo e que os recursos do sistema (entradas, saídas, CPU, etc) estejam sobrecarregados, reproduzindo um cenário com a pior situação possível para a execução de uma aplicação de tempo real. Como os programas *Cyclictest* e *Latency* medem a latência do sistema, um cenário adequado de sobrecarga é o uso intensivo do processador, que no pior caso deve estar com valores próximos de 100% de utilização com poucas variações durante o período de execução dos testes. A solução adotada para produzir esse cenário foi a proposta por Geusik Lin. Esta abordagem, além de proporcionar o uso de 100% do processador, possui uma construção simples que utiliza um conjunto de instruções e programas que já se encontram pré instalados na maioria das distribuições *Linux*.

3.2.2 Configuração do *Kernel*

Rever funcionalidades apontadas como vilãs da latência!

Algumas funcionalidades do *kernel*, como *debug*, gerenciamento de energia, paginação, e consequentemente acessos a disco, podem comprometer a previsibilidade das aplicações de tempo real, para evitar estes problemas, as funcionalidades de *debug* e gerenciamento e economia de energia do *kernel* foram desabilitadas, os problemas relacionados a paginação e acessos a disco foram resolvidos nas próprias aplicações como veremos mais adiante. Embora esta configuração não tenha apresentado problemas no *hardware* de teste, a ausência de recursos de gerenciamento de energia inviabilizou o carregamento do *kernel* em outras configurações de *hardware*. Como este trabalho trata do teste de soluções de tempo real que executam sobre sistemas monoprocessados a funcionalidade do *kernel* que concede suporte a *SMP* foi desabilitada.

Para que um sistema operacional possa executar aplicações de tempo real é necessário que o sistema possua suporte a relógios com uma boa granularidade e precisão, assim as opções do *kernel* relacionadas aos relógios de alta precisão (High Resolution Timer Support) foram habilitadas. Como sistemas de tempo real normalmente são sistemas reativos, seguindo as recomendações da configuração do kernel, a opção Clock Frequency foi configurada para 1000 Hz.

3.2.3 Erros e Falhas Encontradas

3.3 *Benchmarks*

4 RESULTADOS

5 CONCLUSÕES

5.1 Trabalhos Futuros

Bibliografia

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

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

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.

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

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.

LITAYEM, Nabil; 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.

LIU, Chung Laung; LAYLAND, James W. Scheduling algorithms for multiprogramming in a hard-real-time environment. **Journal of the ACM (JACM)**, ACM, v. 20, n. 1, p. 46–61, 1973.

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.

REGNIER, Paul; LIMA, George; BARRETO, Luciano. Evaluation of interrupt handling timeliness in real-time linux operating systems. **ACM SIGOPS Operating Systems Review**, ACM, v. 42, n. 6, p. 52–63, 2008.

SCHILDT, Herbert. **C Completo e Total**. [S.l.]: Makron Books, 1996.

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