AVALIAÇÃO DE SISTEMAS OPERACIONAIS DE TEMPO REAL BASEADOS EM LINUX

Tiago CHAVES (1); Diego TENORIO (2); Raimundo JUNIOR (3); Anderson MOREIRA (4)
Instituto Federal de Pernambuco, Av. Prof. Luiz Freire, 500, Cidade Universitária – Recife – PE, e-mail: (1) tsc@dase.recife.ifpe.edu.br, (2) dst@dase.recife.ifpe.edu.br, (3) rmlj@dase.recife.ifpe.edu.br, (4) anderson.moreira@recife.ifpe.edu.br

RESUMO

Sistemas Operacionais tem como função básica gerenciar os recursos de *hardware* e de *software* do computador, servindo de interface com o usuário que os usa. Entre os Sistemas Operacionais podemos destacar os Sistemas Operacionais de Propósito Geral (SOPG) que possuem dificuldades em atender as demandas específicas quando diz respeito a tempo, pois estes são desenvolvidos para um melhor desempenho médio, não sendo ideais para aplicações críticas. Para suprir essas dificuldades e que possam atender tais aplicações utilizam-se os Sistemas Operacionais de Tempo Real (SOTR). Consequência disso é que a adoção dos SOTR têm aumentado significativamente, principalmente entre os que são baseados em Linux. O presente trabalho, apresentado nesse artigo, tem como ideal facilitar o profissional de computação a escolher entre dois sistemas de código aberto: RTLinux e RTAI. Para avaliação dos sistemas foi utilizado um *benchmark* modificado do MiBench e Hartstone aplicada regras estatísticas de avaliação de desempenho de sistemas. Um estudo de caso, utilizando um braço robótico, foi aplicado utilizando ambos os SOTR citados. Com essas informações, pode se verificar a real maturidade de tais sistemas e se estes possuem realmente características de sistemas de tempo real, entre eles a previsibilidade e a corretude temporal. Tais sistemas, no futuro, poderão ser utilizados no parque tecnológico do Instituto Federal de Pernambuco (IFPE) para desenvolvimento e hospedagem de aplicações críticas de tempo real, que beneficiará diretamente o curso de Automação Industrial.

Palavras-chave: sistemas operacionais, sistemas operacionais de tempo real, avaliação de desempenho

1. INTRODUÇÃO

O ambiente de sistemas embarcados, principalmente na área de controle automotivo industrial e de robótica, vem se tornando cada vez mais complexo. Tipicamente isso consiste em muitos computadores operando em múltiplos níveis de controle ou supervisionando diferentes dispositivos eletrônicos, cada um com uma função diferente. A partir de uma determinada plataforma de *hardware,* a execução eficiente de uma aplicação de tempo real requer do desenvolvedor conhecimento necessário para um gerenciamento eficiente dos códigos fontes, trabalho com tarefas, escalonamento, carga do sistema e programação concorrente. No entanto um SOTR deve prover ao desenvolvedor primitivas que casem com esses conhecimentos citados além de comunicação entre tarefas e operações com dispositivos, entre outros.

A escolha por usar um determinado SOTR em que as características devam oferecer suporte a um domínio específico de sistemas de tempo real é muitas vezes feita de forma não condizente ao que realmente o desenvolvedor quer, ou por motivos de indicações de terceiros ou por reputação no mercado.

Por outro lado os SOTR estão cada vez mais completos e com uma série de recursos que muitas vezes não são utilizados pelo projetista. Assim, também são importantes fatores como flexibilidade para alterações, confiança no desenvolvedor, garantia de funcionamento, facilidade de suporte, entre outros.

O presente trabalho de Iniciação Científica, apresentado nesse artigo e financiado pela Diretoria de Análise e Desenvolvimento de Tecnologias do IFPE, tem como ideal facilitar o projetista de tempo real a escolher entre sistemas de código aberto RTLinux e RTAI, sendo que estes têm tido uma importância muito grande nos últimos tempos já que fazem uso do Linux como sistema hospedeiro e o mesmo carrega consigo grandes facilidades de desenvolvimento. Não ocorreu nenhuma comparação com sistemas proprietários tais como o *VxWorks* (VXWORKS, 2008) ou o *Windows* CE (YAO, 2009), pois o trabalho desenvolvido é verificar se pode utilizar sistemas baseados em Linux de Tempo Real no parque tecnológico do IFPE.

Este documento está estruturado nos seguintes tópicos: a Seção 2 cita o que são sistemas operacionais de tempo real e os principais trabalhos relacionados com o assunto tratado neste artigo; na Seção 3 são explorados os SOTR em estudo e a metodologia utilizada; na Seção 4 ilustra a idéia do *benchmark* desenvolvido e das técnicas de avaliação com definições formais. Também são apresentados os resultados obtidos; na Seção 5 mostra-se o estudo de caso que foram utilizados nos SOTR em questão no controle de um braço robótico; e na Seção 6 são expostas as conclusões e os projetos futuros que poderão ser criados.

2. SISTEMAS OPERACIONAIS DE TEMPO REAL E ANÁLISE

Sistemas de Tempo Real (STR) são, em geral, reativos ao ambiente que estão inseridos, o que faz com que a prova dos requisitos críticos (lógica e temporal) sejam extremamente complexos (KOPETZ, 1997). Atualmente encontramos STR em uma gama bem diversificada de aplicações, desde eletrodomésticos e videogames até complexos sistemas de controle de tráfego aéreo (CTA). De acordo com FARINES, 2000 o problema de tempo real é fazer com que as aplicações tenham um comportamento previsível, atendendo as características temporais impostas pelo ambiente ou pelo usuário, mesmo com recursos limitados.

O objetivo principal dos STR não é executar o aplicativo o mais rápido possível e sim executar no tempo estabelecido e momento correto (STANKOVIC, 1988). Quanto à solução os STR podem ser classificados em **Sistemas Operacionais de Tempo Real** e **Núcleos de Tempo Real**.

Para FARINES, 2000, SOPG possuem dificuldades em atender as demandas específicas das aplicações de tempo real, não possuindo em nenhum momento preocupação com a previsibilidade temporal, requisito indispensável para o bom funcionamento de tais aplicações. Porém sistemas atuais como o Linux já na versão 2.6.8.1, já possuem em seu escalonador propriedades para a utilização de aplicações não-críticas, especificadas pelo padrão POSIX2 1003.1b e 1003.1c3 (POSIX, 2000).

Os SOPG tipicamente são projetados para um melhor desempenho médio, ou seja, para que a maioria das tarefas sejam atendidas justamente entre os processos e que estas utilizem os recursos da máquina de forma igualitária. Certas funções do SOPG são escondidas dos projetistas e/ou programadores de sistemas, fazendo com que certas funcionalidades não possam ser alteradas de forma direta.

Em SOTR as mesmas funcionalidades de auxílio dos SOPG são encontradas, porém são, em geral, sistemas abertos onde os mecanismos que antes eram escondidos nos SOPG agora estão acessíveis. Isso se deve às propriedades inerentes aos sistemas de tempo real que são a corretude lógica (*correctness*) e a corretude temporal (*timeliness*), sendo as duas de igual importância para garantir a previsibilidade que pode ser considerada o coração de sistemas dessa natureza.

Essa característica se aplica devido ao tempo estabelecido, pois as tarefas que estão sendo executadas devem ser cumpridas em prazos previamente conhecidos. Visibilidade e controle de recursos de *hardware* são fundamentais em tais sistemas. Várias funcionalidades facilitadoras encontradas em SOPG tornam o comportamento dos SOTR pouco previsíveis, tais como o uso de memória *cache* que gera um tempo adicional.

Já em Núcleos de Tempo Real encontramos uma solução que tenham garantias temporais bem maiores do que em SOTR. Alguns autores de trabalhos de análise em SOTR como SHIN E KIM, 2001, estabelecem quatro grandes grupos para comparar sistemas operacionais desse tipo, são eles: Desempenho, Segurança, Flexibilidade e Custo. Essas características em comparação ao presente trabalho não tem grande importância, podendo ser vistas em algum trabalho futuro.

O artigo de STANKONVIC E RAJKUMAR, 2004, descreve uma série de características que podem ser encontradas em SOTR. Essas características são agrupadas seguindo diferentes classificações. Também são descritas técnicas de escalonamento que podem ser utilizadas em sistemas de tempo real. No entanto o trabalho deles não fala de um SOTR específico e nem de características que possam ajudar a escolher um determinado sistema. Já no trabalho de WEINBERG E CESATI, 2001, encontra-se a descrição de uma aproximação para medir e analisar o tempo de execução de um *kernel* de tempo real em uma plataforma de *hardware* específica. Além de descrever os passos necessários para mudar de uma solução proprietária de SOTR para Linux com funções de tempo real. Porém nenhuma análise é descrita para apresentar a importância de fazer tal mudança e o motivo não é bem explicado, levando apenas em consideração a questão do valor e de que o Linux é um sistema muito utilizado mundialmente.

3. METODOLOGIA E FUNDAMENTAÇÃO

3.1 Avaliação de Sistemas

Avaliação de Sistemas são técnicas e metodologias específicas para comparar ou criar escalas de dados em cima de um determinado sistema (NIST, 2010). Esses dados servirão de insumos para poder dizer se um sistema é bom para quê ou ruim para quê, definir estratégias de projetos ou entender bem o seu funcionamento. Essas avaliações podem ser por medições através de *benchmark*, prototipação ou mensuração direta. Também podemos usar de modelagem para avaliar através de simulação e analise de probabilidade. Um projeto de avaliação o projetista deve ter em mente as seguintes questões: o que deve ser avaliado; como avaliar; quem vai avaliar; para quê avaliar; onde avaliar. Além de definir um passo-a-passo para essas questões.

A avaliação também pode ser feita através da execução de um programa específico. Este está diretamente ligada ao gerenciamento que o sistema operacional faz, pois é ele quem vai definir em que momento um dispositivo pode ser acessado por um programa, se um ou vários processos vão ser executados no computador ou qual a sua melhor forma de uso. Isso é genérico para qualquer sistema operacional seja ele *Windows*, Linux ou MacOS.

3.2 Ambiente de execução

Quando comparamos diferentes SOTR é importante que a configuração do *hardware* escolhido seja equivalente ou igual, para que obtenha resultados compatíveis. As plataformas usadas para o trabalho são da arquitetura x86 e as configurações:

- Computador do tipo PC, placa-mãe ASUS A7N8X-X, CPU AMD de 1.4 GHz (Giga hertz) com 256 KB de cache L2, placa gráfica de 32 MB (Mega *Byte*) no barramento PCI, memória RAM de 512 MB do tipo DDR 400 sem *buffer* e disco rígido de 80 GB (Giga *Byte*);
- Computador do tipo PC, placa-mãe SiS 610, CPU Pentium IV de 1.5 Ghz com 256 KB de cache L2, placa gráfica de 32 MB no barramento PCI, memória RAM de 512 MB do tipo DDR 400 sem *buffer* e disco rígido de 80 GB.

3.3 Sistemas Operacionais de Tempo Real utilizados e suas características

O SOTR, RTLinux 3.2 surgiu da idéia de usar o sistema UNIX como base para ser utilizado por aplicações de tempo real. O princípio era criar um SOTR que fosse aberto para modificações, eficiente e que suportasse multitarefa para aplicações de tempo real do tipo *hard* (que são aquelas que têm problemas graves caso não sejam completadas no tempo previsto). A escolha do Linux pela empresa desenvolvedora do RTLinux, se deu porque a licença de uso não era restrita como do UNIX, e o seu código fonte se encontra disponível para alterações. Disponibilizar o código fonte do sistema operacional para o desenvolvedor é essencial para a verificação, análise e correção de problemas do sistema como um todo.

O sistema operacional RTLinux é um *microkernel* de tempo real que trabalha junto com o *kernel* do Linux (ver Figura 1), para alguns autores esse esquema também é conhecido como Dual kernel (STANKONVIC E RAJKUMAR, 2004). Atualmente o SOTR possui duas vertentes: uma aberta para desenvolvimento à comunidade mundial, que é a escolhida no presente estudo, o RTLinux/Open e a versão comercial que possui várias outras funcionalidades que a versão aberta não possui, o RTLinux/Pro.

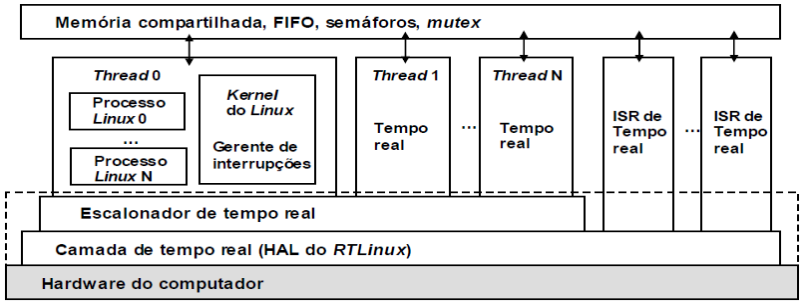

Figura 1 - Estrutura principal do *RTLinux* para sistemas de tempo real (FSMLABS, 2009)

Um dos aspectos mais importantes do RTLinux é o modo em que ele é executado. A HAL (Camada de abstração de *Hardware)* é carregada em uma versão qualquer do Linux executando como módulo desse sistema. Quando o código do SOTR é carregado todas as funcionalidades para tempo real ficam prontas, mas ainda mantém o *kernel* do Linux executando como se fosse uma tarefa ociosa, ou seja, quando não há tarefas de tempo real executando o sistema operacional padrão assume as funções.

Já o RTAI 3.3 (*Real-time Application Interface*) é um projeto que surgiu como sendo uma variante do RTLinux, pelo *Dipartimento di Ingeneria Aerospaziale da Politecnico di Milano* (RTAI, 2009). Tem sua base ainda o uso de uma interface entre o *hardware* e o *kernel* do Linux, porém possuí algumas partes complementares para suprir algumas dificuldades encontradas no RTLinux, como tratamento de várias tarefas ao mesmo tempo e controle de interrupções. A arquitetura do RTAI é similar a do RTLinux, porém não se limita a alterar o *kernel* padrão apenas adicionando a HAL e capturando as interrupções oriundas deste, o SOTR acrescenta interrupções por software chamadas `hard_sti` e `hard_cli`, estas assumem as funcionalidades das interrupções originais do Linux, `cli` e `sti` incluindo ao sistema, as estruturas básicas para execução de aplicações de tempo real e implementa as interrupções de controle de processadores. Uma vantagem desse tipo de codificação é que possibilita o *kernel* padrão do Linux operar normalmente mudando os ponteiros na estrutura HAL para trabalhar com recursos para alocação de processos em diversos processadores e aplicações de tempo real.

3.4 Análise utilizando *benchmark*

Benchmark é uma aplicação ou conjunto de aplicações que exercitam um determinado sistema baseado em diferentes parâmetros de entrada e retornando valores que possam ser analisados com finalidade de trazer melhorias. São usados largamente para analise de desempenho em sistemas computacionais e são encontrados e utilizados nas mais diversas aplicações, como dispositivos de rede, telecomunicações, entretenimento digital entre outras.

Nos testes realizados optou-se em utilizar *benchmarks* de conceitos comuns em sistemas de tempo real: troca de contexto, tratamento de interrupções, chamadas de sistemas (*system calls*), tempo de execução de uma *thread* em um determinado tipo de tarefa e outras intervenções geradas pelo sistema operacional. Foi criado um *benchmark* que foi fundamentado em outras duas soluções para testes de sistemas de tempo real o MiBench (GUTHAUS et al, 2001) e o Hartstone (WEIDERMAN E KAMENOFF, 1992). Verificou também a resposta do sistema na capacidade de executar tarefas do tipo *hard* e a manipulação de interrupções com a execução de tarefas em dois momentos, o primeiro que agem de forma periódica e não-harmônica e o segundo momento que executa de maneira periódica com uma carga de tarefa aperiódica.

4. TÉCNICA DE AVALIAÇÃO E COMPARATIVO

Definido o conjunto de testes e suas aplicações, faz necessário saber como estas serão utilizadas no *benchmark* e para isso deve-se gerar uma série de testes de execução. Os requisitos e como estes devem ser executados para a verificação das funcionalidades e eficiência do SOTR foram descritas pelo sistema Hartstone (WEIDERMAN e KAMENOFF, 1992) na qual representa uma forma de como os requisitos de tempo real (algoritmo, implementação de compilador, sistema operacional e componente de *hardware)* devem ser representados.

Características de tempo real como atividades periódicas, processamento aperiódico gerado por interrupções ou por intervenção de usuário, sincronização entre as mais diversas atividades, acesso a dados compartilhados, mudança de modo e distribuição de tarefas, devem ser levadas em consideração pela execução do Hartstone já que este precisa ser o mais confiável possível, principalmente para que possa atuar em aplicações reais.

As definições utilizadas do Hartstone são a Série-PH – tarefas periódicas e harmônicas e a Série-AH – tarefas do tipo Série-PH com processamento aperiódico. A primeira tem como objetivo fornecer requisitos de testes simples com uma carga de tarefas que são puramente periódicas e harmônicas em sua forma de execução, esta série pode representar um programa que monitore vários sensores de controle com diferentes taxas e divulga o resultado sem intervenção do usuário. A base do sistema consiste em cinco *threads* periódicas em que a frequência de cada *thread* é múltipla da frequência de *thread* maior. No presente trabalho foi descrita da seguinte forma (ver Figura 2):

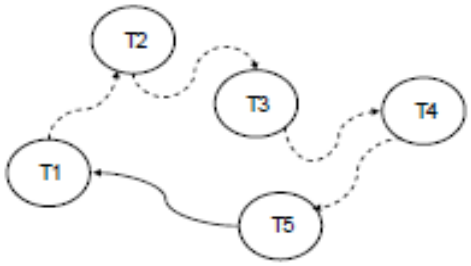

Figura 2 - Ordem de execução das tarefas de acordo com a Série-PH (WEIDERMAN e KAMENOFF, 1992)

Considerando a Figura 2, as frequências das *threads* T1, T2, T3, T4 e T5 são respectivamente, 16Hz, 8Hz, 4Hz, 2Hz e 1Hz. Todas as tarefas devem ser escalonadas para iniciar ao mesmo tempo e o ciclo total deve ser de no máximo 2 milisegundos (ms). Apesar de que na Figura 2 exista uma ordem de precedência entre as tarefas, quem irá determinar qual irá executar primeiro é o escalonador do sistema operacional.

Já na Série-AH, o objetivo dos testes é determinar um conjunto de regras que representam domínios de aplicações em que o sistema responde a eventos externos. Por exemplo, um sistema que possui uma interface com usuário que é dirigida a eventos. As tarefas aperiódicas serão caracterizadas por intervalos de tempo de ativação não conhecidos, geradas aleatoriamente, e que tenham de cumprir com seus tempos limites de finalização. O mesmo esquema de tarefas serão criados com base no esquema da série-PH, porém terá agora duas tarefas aperiódicas para serem executadas em algum momento dos testes (ver Figura 3):

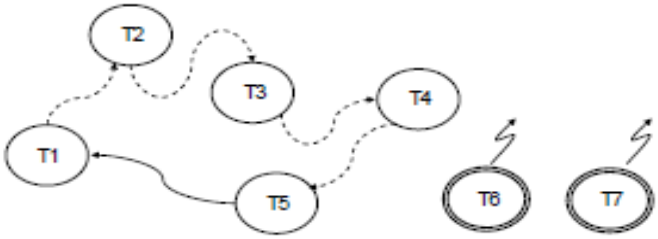

Figura 3 - Ordem de execução das tarefas de acordo com a Série-AH (WEIDERMAN e KAMENOFF, 1992)

Considere na Figura 3 que as tarefas T6 e T7 começam a sua inicialização em um tempo gerado aleatoriamente pela aplicação. E irão executar com uma prioridade maior que as outras tarefas e devem ser executadas e cumprir com a execução em até 20 ms após a sua ativação. A função que estas devem executar é da *thread* enviar um sinal para si mesma. Um manipulador de sinais foi criado e imprime na tela uma mensagem de alerta, cada vez que uma *thread* é executada.

Definido como foi criado o *benchmark* de teste dos SOTR será apresentado a seguir e a avaliação de cada sistema em estudo, o RTlinux e o RTAI.

4.1 Testes no RTLinux e no RTAI

Durante a execução do *benchmark* nos SOTR estudados podemos observar que os mesmos tiveram comportamentos distintos, porém todos atingiram as restrições impostas pelo programa. No sistema RTLinux, quando executado não ocorreu nenhuma anomalia ao sistema operacional e todos os testes foram bem sucedidos. Porém para não gerar mais latência aos resultados, utilizando funções de impressão na tela, estes foram dirigidos diretamente para o relatório (log) do sistema.

Na execução da sequência da série-PH o sistema manteve a média de execução das cinco tarefas, dentro do ciclo total, ou seja, dentro do espaço de 2 ms. Não teve resposta negativa nenhum dos testes e o mesmo foi repetido durante 2 horas. Não sofrendo influência das tarefas nativas do Linux que se encontravam acionadas no sistema operacional. O tempo de execução de cada tarefa também não sofreu alterações bruscas. Na série-AH também manteve a mesma ordem de grandeza na execução das tarefas.

No RTAI, tivemos algumas particularidades. Como forma de execução o *benchmark* foi executado em modo *kernel* assim como no outro SOTR. Também não foram utilizadas funções de impressão na tela, estes foram dirigidos diretamente para o log do sistema. Na execução da sequência da série-PH o sistema manteve a média de execução das cinco tarefas, dentro do ciclo total de 2 ms. Este também não teve resposta negativa em nenhum dos testes e o mesmo foi repetido durante 2 horas, porém sofreu influencia das tarefas nativas do Linux que se encontravam em execução no sistema operacional. No entanto o tempo de execução de cada tarefa também não sofreu alterações bruscas. Na série-AH também manteve a mesma ordem de grandeza na execução das tarefas.

A seguir a Tabela 1 apresenta os resultados obtidos durante a execução:

Tabela 1 – Resultados obtidos durante a execução do benchmark

SOTR	Tempo de Latência	Tempo para Série-PH	Tempo para Série-AH
RTLinux	4 µs	~ 34 µs	~ 42 µs
RTAI	3 µs	~ 36 µs	~ 47 µs

Pela Tabela 1 o SOTR que apresenta os melhores resultados de execução do conjunto de tarefas é o RTLinux em comparação ao RTAI. Porém se considerado o RTAI com a configuração manual das resoluções de frequência, por causa da latência, muito provavelmente obteria melhores resultados que o RTLinux.

5. ESTUDO DE CASO

Aplicações de robótica fornecem uma forte ferramenta de validação dos componentes de SOTR, já que consiste de vários dispositivos comunicantes entre si e alguns com controles independentes que fazem uso de gerenciamento dinâmico de memória até processo de comunicação entre processos como semáforos, barreiras e rede. A aplicação consiste de um sistema computacional entre um braço robótico e um motor ambos controlados por SOTR e a comunicação entre os sistemas é feito através de uma rede *Ethernet,* assim como observado na Figura 4.

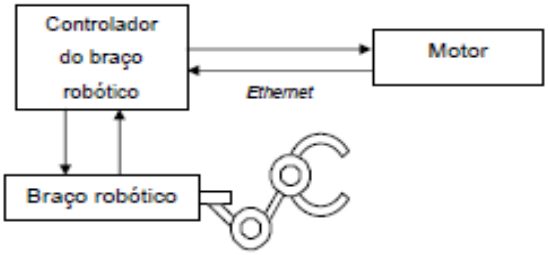

Figura 4 - Descrição do sistema robótico

O usuário do sistema pode interagir com o mesmo usando um ambiente de controle localizado no motor, sendo que para inicializar o braço robótico deve-se usar um sistema interativo que verifique a correta inicialização através de mensagens ao usuário e o motor é inicializado verificando se existe a sincronização entre o mesmo e o controle do braço. Caso falhe na verificação mensagens de erro devem ser mostradas ao usuário. No ambiente do motor uma tela deve desenhar o atual estado do braço e se este se encontra próximo a algum obstáculo. As tarefas no controle do braço são ignoradas e leva-se em conta apenas a forma de comunicação e se o ambiente pode utilizar um SOTR para controle.

A verificação deve ser a cada 20 ms e a sincronização entre os componentes envolvidos não deve ser maior que 30 ms. Para o controlador do braço robótico o RTLinux é mais confiável que o RTAI, já que possui uma API concisa e estável tanto de recursos de comunicação quanto de recursos de alocação de memória. Podemos observar essas características já que as estruturas de acessos e criação de *threads* são bem simples e evitam acessos desnecessários a algumas partes do sistema operacional. Também a forma de modificar o escalonador do sistema é bem simples, precisando apenas modificar um único arquivo (*schedule.c*) que possui todas as políticas de escalonamento implementadas. Um tamanho ideal de escalonamento que não implica muito *overhead* no sistema é em torno de 15 *threads*.

Por padrão o sistema aceita no máximo 128 *threads*, sendo que este número pode ser modificado durante a compilação na função RTL_PTHREAD_THREADS_MAXN. No entanto o RTAI fornece baixa latência e rápida troca de contexto entre as tarefas, a resolução em nanosegundos dos temporizadores pode ser bastante

úteis para este cenário. Durante a execução do estudo de caso podemos observar que o SOTR, possui uma estrutura de definição de relógios e temporizadores chamada `RTIME,` que é expressa em nanosegundos. Essa definição é mais simples, favorecendo assim o acesso mais rápido, por processos, a dados da camada de tempo real. A função `rt_get_time`, retorna o tempo mensurado delimitadas pela estrutura `rt_start_time` e expressa em nanosegundos. Uma função similar é a `rt_get_time_ns`. Além disso, o sistema possui o recurso de *watchdog* como módulo programável para executar determinadas ações na ocorrência de excessos de uma tarefa. Isso é útil, caso em um sistema de automação tiver um excesso de tarefas com a mesma prioridade porém em tempos de execução diferentes. Outras características foram observadas durante o projeto e não foram citadas no artigo.

A seguir a Tabela 2 com as vantagens e desvantagens de cada SOTR:

Tabela 2 – Comparação de acordo com vantagens e desvantagens na implementação do Estudo de Caso.

Estudo de Caso	RTLinux	RTAI
Braço Robótico	**Desvantagens** – Motor: API gráfica de difícil desenvolvimento; falta recursos de comunicação com garantias de previsibilidade; Braço: falta de precisão (operações com ponto flutuante); latência alta. **Vantagens** – uso de *pthreads* (POSIX) facilita o desenvolvimento do Braço, já que usando essa função já se enquadram como de tempo real.	**Desvantagens** – Motor: API gráfica de difícil desenvolvimento; Redundância de determinadas funções da API; interrupções do sistema hospedeiro (Linux) geram tempo não previsível; **Vantagens** – Processos executando em modo usuário; uso de recursos de rede; Braço: fácil conexão com outros SOTR que utilizam recursos de rede; baixa latência e rápida troca de contexto.

6. CONCLUSÃO

Nos SOTR que foram estudados podemos notar que tanto o ambiente do RTLinux quanto o RTAI são limitados quanto a serviços do padrão POSIX e que são primordialmente ligados a ambientes de gerenciamento de E/S, já que a camada de tempo real faz uso do controle de interrupções do Linux para utilização de aplicações de tempo real. Os ambientes podem enviar e receber dados através de serviços do sistema hospedeiro, mas acesso a disco, arquivos, sistemas de rede e acessos de E/S de dispositivos não são acessíveis totalmente na camada de tempo real.

Também podemos observar que os sistemas não são apropriados para hospedar aplicações de tempo real que sejam muito complexas, como por exemplo, um sistema de simulador de vôo completo entre outras. Por isso os estudos em cima de sistemas de *benchmark* são importantes, pois verifica se os SOTR em questão podem ser apropriados para um determinado tipo de aplicação ou não. Uma forma de verificar esse tipo de aplicações em ambientes de tempo real é usar aplicações críticas em conjunto com aplicações normais e verificar se as críticas sofreram interferência das não críticas, como apresentado na execução da Série-AH.

Um fator importante observado durante a execução dos testes é que os sistemas RTLinux e RTAI reduzem o tempo na ocorrência de interrupções, e estes sistemas disponibilizam um rico ambiente de desenvolvimento. Porém a habilidade de resolver problemas de requisitos de tempo real não se limita apenas em adaptar um sistema Linux para funcionar com um melhor gerenciamento de interrupções.

Observamos também que quanto à compatibilidade com outros padrões o que observamos é que a interface POSIX que é encontrada nos sistemas baseados em Linux permite as aplicações serem portadas para outros sistemas sem a necessidade de ficarem atreladas a um determinado SOTR. Mesmo que a compatibilização, em muitos casos, como no RTLinux e RTAI não sejam totalmente compatível com o padrão POSIX, a maioria das funções de tempo real podem ser utilizadas sem muitos problemas de conversão em outros ambiente. Exemplo disso é a transcrição de aplicações do RTLinux para o SOTR VxWorks.

Uma extensão do presente trabalho é a criação de um sistema de *benchmark* que envolva outras características de latência e que possa ser utilizado como uma aplicação simulada de tempo real, fazendo assim uma melhor observação por parte dos projetistas, e possibilitar levar em consideração outros aspectos na utilização de tais sistemas operacionais. Uma avaliação maior também poderia ser feita, envolvendo outros sistemas comerciais como o QNX ou o VxWorks, comparando com as iniciativas abertas encontradas. Também incluir nos estudos de casos outras aplicações reais, tais como sistemas automotivos e linhas de produção.

Outro trabalho futuro é realizar a mesma pesquisa em diferentes plataformas como SH, MIPS ou Power PC, assim um comparativo mais abrangente pode ser definido e a utilização de tais plataformas apresentará uma analise mais ampla, favorecendo a criação de aplicações de tempo real mais precisas e robustas.

REFERÊNCIAS

FARINES, J., FRAGA, J.S. e OLIVEIRA, R.D. **Sistemas de tempo real, Escola de Computação**, Florianópolis, 2000.

FSMLABS. ***RTLinux Free – A overview***. Disponível em <http://www.fsmlabs.com/rtlinuxfree.html>. Acesso em: 05 set 2009.

GUTHAUS, M.; RINGENBERG, J.; ERNST, D.; AUSTIN T.; MUDGE, T.; BROWN, R., ***MiBench: A free, commercially representative embedded benchmark suite***, *Proceedings of the IEEE International Workshop on Workload Characterization*, vol. 00, ed. IEEE Computer Society, Washington, EUA, pp. 3-14, 2001.

KOPETZ, H. Real-Time Systems: design principles for distributed embedded applications. Kluwer Academic Publishers, 338 p, 1997.

LIU, C. L., LAYLAND, J. W. ***Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment***. *Journal of the Association for Computer Machinery*, v. 20, n. 1, pp. 46-61, 1973.

NIST/SEMATECH, ***e-Handbook of Statistical Methods***, Disponível em <http://www.itl.nist.gov/div898/handbook/>, Acesso em: 18 jan 2010.

POSIX 2003.1b 2000. ***IEEE Standard for Information Technology - Test Methods Specifications for Measuring Conformance to POSIX - Part 1: System Application Program Interface (API) - Amendment 1: Real-time Extension [C Language]***, Nova York, NY, EUA, 384 p, 2000.

RTAI, ***RTAI – Official Website***, Disponível em <https://www.rtai.org/>. Acesso em: 23 ago 2009.

SHIN, D.; KIM, J., ***Intra-task Voltage Scheduling on DVS-Enabled Hard Real-Time Systems***, *IEEE Design and Test of Computers*, Março de 2001.

STANKOVIC, J. ***Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems***. IEEE *Computer Society Press*, vol. 21, no. 10, pp. 10-20, Outubro de 1988.

STANKOVIC, J.; RAJKUMAR, R., ***Real-time Operating Systems***, *Kluwer Real-Time Journal*, vol. 28, no. 2-3, ed. *Springer Netherlands*, pp. 237-253, Novembro de 2004.

VXWORKS, ***General Purpose Platform, VxWorks Edition, Product Overview***. Disponível em: <http://www.windriver.com/products/product-overviews/General-Purpose-Platform-ve-overview.pdf> Acesso em: 15 nov 2008.

WEIDERMAN, N.; KAMENOFF, N., ***Hartstone Uniprocessor Benchmark: Definitions and Experiments for Real-Time Systems***, *The Journal of Real-Time Systems*, pp. 353-382, 1992

WEINBERG, B., CESATI, M. ***Moving from a Proprietary RTOS To Embedded Linux.*** *MontaVista Software, Inc*., pp. 55-58, 2001.

YAO, P. ***Windows CE: Enhanced Real-Time Feature Provide Sophisticated Thread Handling***. Disponível em: <http://msdn.microsoft.com/msdnmag/issues/1100/RealCE/default.asp>. Acesso em: 11 ago 2009.