Método para Diminuir o Tempo de Interferência de Tarefas de Tempo Real

Ítalo Campos de M. Silva[1], Rômulo Silva de Oliveira[1], Luciano Porto Barreto[2]

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88040-900 – Florianópolis – SC – Brasil

[2]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
CEP 40170-110 – Salvador – BA – Brasil

{italo,romulo}@das.ufsc.br, lportoba@ufba.br

Abstract. *Real-time tasks run on Linux with PREEMPT-RT patch won best accuracy from the use of high resolution timers. These timers are processed through interrupts and interfering with all tasks that are running, whether real time or not. Some of these timers are processed in hard irq unnecessarily interfering with real time tasks for a longer time than necessary. This article proposes a method to solve this problem and reduce the time of preemption of these tasks.*

Resumo. *As tarefas de tempo real executadas no Linux com o patch PREEMPT-RT ganharam melhor precisão desde a utilização dos temporizadores de alta resolução. Estes temporizadores são processados através de interrupções, interferindo em todas as tarefas que estejam executando, sejam elas de tempo real ou não. Alguns destes temporizadores são processados em hard irq desnecessariamente, interferindo em tarefas de tempo real por um tempo maior do que necessário. Este artigo propõe um método para resolver este problema e diminuir o tempo de preempção destas tarefas.*

1. Introdução

O Linux vem se tornando cada vez mais popular, pois ele é um sistema operacional de propósito geral que fornece bom desempenho, estabilidade e baixo tempo médio de resposta. Ele está sempre em constante evolução devido ao grande conjunto de desenvolvedores que estudam seu código e tentam melhorá-lo. Devido a ser um sistema operacional de código aberto, p seu *kernel* pode ser estudado e alterado por qualquer pessoa, sendo ele muito utilizado no meio acadêmico por tais motivos.

O desenvolvimento do *kernel* padrão do Linux não tem como prioridade suportar aplicações de tempo real, mas mesmo assim ele já implementa o padrão POSIX.4 [POSIX.13 1998]. Mas para suportar de forma mais completa aplicações de tempo real, existem *patches* que alteram o código do *kernel* padrão. Um deles e o estudado para este artigo é o *PREEMPT-RT*, que tem como objetivo prover determinismo para tarefas de alta prioridade no *kernel* [Molnar 2005].

Este artigo irá propor alterações no *kernel* do Linux com *PREEMPT-RT*, na tentativa de diminuir o tempo de interferência causada sobre tarefas de tempo real em determinadas situações. Demonstrando através de medições e equações matemáticas o problema

existente, logo depois demonstra-se através de exemplos e equações o que deve ocorrer com as alterações propostas.

2. Hard irq e SoftIRQ

O Linux trabalha com o conceito de chamar a atenção do processador quando necessário através de interrupções, as quais são utilizadas para alterar o fluxo de execução normal do sistema [Mauerer 2008]. Quando a interrupção é gerada, o processador pára tudo o que está fazendo e executa um código responsável por tratá-la (tratador de interrupção) em contexto de interrupção. Após esta interrupção ter sido tratada, o processador volta a executar o que estava sendo executado.

O tratador de interrupções em geral é dividido em duas partes: *Top Half* (atividades executadas assim que ocorre a interrupção) e *Bottom Half* (atividades relacionadas à interrupção que podem ser postergadas para executarem em um momento mais oportuno) [Love 2005]. As atividades executadas em *Top Half* também podem ser ditas que estão sendo executadas em *hard irq*. Já a *softIRQ* é uma maneira de postergar trabalho dentro do kernel, ou seja, é um dos tipos de tarefas da *Bottom Half*.

3. Temporizadores de Alta Resolução

O Linux utiliza-se de temporizadores para realizar muitas tarefas, como atualizar a hora do sistema, detectar falhas de envio de pacotes pela rede, escalonar tarefas, entre outras. Algumas destas tarefas não têm necessidade de uma grande resolução temporal, mas outras podem oferecer um serviço bem melhor dependendo da resolução temporal que o kernel forneça a elas [Etsion et al. 2001].

No intuito de melhorar o desempenho de muitas tarefas, as quais necessitavam de melhor precisão temporal, o subsistema de tempo do kernel do Linux foi alterado, facilitando a manutenção e desenvolvimento de temporizadores, tendo também sido desenvolvido um tipo de temporizador com alta resolução (*High Resolution Timer*) [Gleixner and Niehaus 2006].

A diferença fundamental entre estes temporizadores e os já existentes (temporizadores clássicos) é que estes utilizam-se de interrupções de disparo único (*one-shot*). Desta forma um dispositivo de relógio é programado para gerar uma interrupção no momento exato em que o temporizador deve expirar, sendo este dispositivo reprogramado para o momento da próxima expiração sempre que for gerada a interrupção.

Estes temporizadores possuem um ponteiro para uma função, a qual deve executar sempre que eles expirem. Como eles são executados através de interrupções, eles impõem sua execução acima de qualquer tarefa que esteja sendo processada. Mas alguns temporizadores de alta resolução não precisam ser processados em *hard irq*, tendo seu processamento postergado, sendo executados através de *softIRQs*. Assim os temporizadores de alta resolução na sua maioria aumentam de forma considerável o tempo gasto no processamento de interrupções, garantindo que as funções ligadas a eles são processadas no menor tempo possível depois que eles expiram.

4. Descrição do Problema

Os temporizadores de alta resolução preemptam as tarefas que estão sendo executadas no sistema por um tempo maior ou menor, dependendo da quantidade de temporizado-

res expirando ao mesmo tempo e se eles devem ser executados via *hard irq* ou *softIRQ*. Mas ainda existem alguns temporizadores dos que são executados via *hard irq* que são utilizados como *sleeps* por determinadas tarefas, ou seja, utilizados para acordar a tarefa depois de um período de tempo. Através de medições realizadas no kernel do Linux, verificou-se que o maior número de temporizadores de alta resolução são os executados em *hard IRQ*, mas que não são definidos explicitamente como temporizadores de *sleep*, tendo na sua maioria de execuções 475 temporizadores por segundo como pode ser verificado na figura 1. Seguido deles estão os que são definidos explicitamente como *sleep*, com uma maioria de 100 temporizadores por segundo e os executados via *softIRQ* são pouquíssimos.

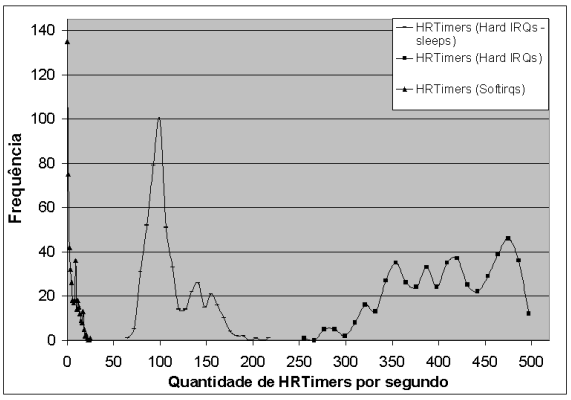

Figure 1. Quantidade de HRTimers executados por segundo

Os temporizadores definidos explicitamente como *sleep*, tem o objetivo de retirar uma tarefa da fila de espera do processador e colocá-la na fila de prontos, desta forma a tarefa será escalonada pelo processador e entrará em execução assim que for possível. Mas esta tarefa que será acordada pelo temporizador possui uma prioridade, a qual é utilizada para definir quando ela será executada pelo processador. Desta forma mesmo ela sendo acordada pelo temporizador em *hard IRQ*, ela deverá esperar ser executada de acordo com sua prioridade. Sendo assim, caso existam outras tarefas de maior prioridade na fila de prontos, elas serão executadas antes desta tarefa. Assim acordá-la neste momento não causará nenhum benefício para ela, pois ela terá de aguardar sua execução de qualquer forma e ainda atrapalha por um tempo maior as tarefas de prioridades maiores que a dela.

Como exemplo supõe-se três tarefas escalonadas pela política de escalonamento para tempo real SCHED_FIFO (T1, T2 e T3), as quais têm prioridades 90, 80 e 70 respectivamente. Considera-se a tarefa T1 com *deadline* e período igual a 32 e as tarefas T2 e T3 com *deadline* e período igual a 30. Considera-se também o tratador de interrupção dos temporizadores como uma tarefa de mais alta prioridade, já que executa em *hard IRQ*. Então como demonstrado na figura 2, T1 começa a executar e logo após dois temporizadores responsáveis por acordar T2 e T3 expiram, gerando uma interrupção no processador que executa o tratador de interrupção, o qual preempta todas as tarefas em execução e conduz T2 e T3 para a fila de prontos. Depois da interrupção ser tratada as tarefas voltam a executar de acordo com suas prioridades. Pode-se verificar na figura 2 que o tratador de interrupção gasta um tempo considerável, justamente para acordar as tarefas T2 e T3 que não vão poder executar ainda, preemptando de forma desnecessária T1.

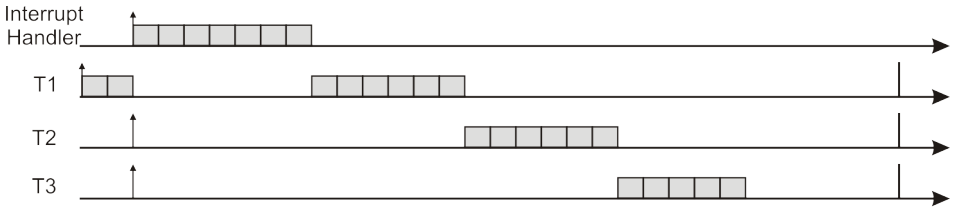

Figure 2. Exemplo de como o tratador de interrupção atrapalha tarefas de alta prioridade

Pode-se calcular o tempo que o tratador de interrupção dos temporizadores gasta, preemptando as tarefas em execução no sistema, utilizando a equação (1).

$$C_{IRQ} = C_{TC} + C_{FG} + C_{TIH} \tag{1}$$

Onde C_{IRQ} é o tempo total gasto pela chamada de interrupção do processador, C_{TC} é o tempo gasto na troca de contexto, C_{FG} é o tempo gasto nas funções gerais de interrupção (como atualizar estatísticas e variáveis) e C_{TIH} é o tempo gasto pelo tratador de interrupção dos temporizadores de alta resolução, o qual pode ser calculado pela equação (2).

$$C_{TIH} = \sum_{i=1}^{n} \Big(C_{FT}(i) \Big) \tag{2}$$

Onde $C_{FT}(i)$ é o tempo de computação da função atrelada ao temporizador i, o qual varia bastante de acordo com o que ele executar, podendo ter vários fatores responsáveis pela variação da mesma função.

5. Abordagem Proposta

Este trabalho propõe diminuir o tempo de preempção que uma tarefa pode sofrer pelo tratador de interrupções em alguns casos. Mais especificamente quando a interrupção é para tratar temporizadores expirados e que tenham como função acordar alguma tarefa. Pois a proposta é fazer com que as funções ligadas a estes temporizadores sejam executadas em momentos posteriores, não atrapalhando assim as tarefas com prioridades maiores do que a que criou o temporizador. Mas ainda assim acordar a devida tarefa no momento que ela possa ser executada sem atrapalhar outras de prioridade maior.

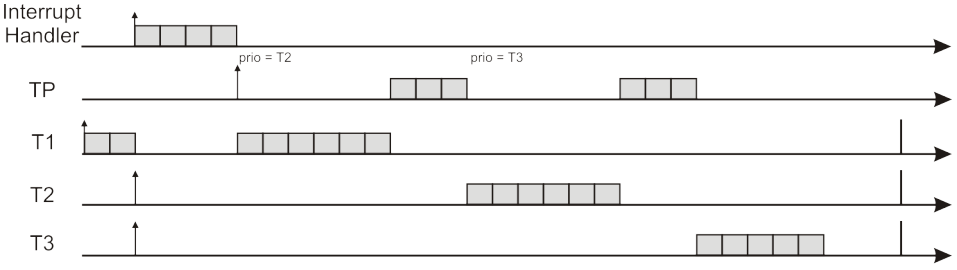

Figure 3. Exemplo de como deve funcionar após a implementação

A proposta é criar uma tarefa com prioridade dinâmica, ou seja, sempre que necessário ela terá sua prioridade modificada. Ela será responsável por executar todas as funções ligadas aos temporizadores definidos explicitamente como *sleeps*. Esta tarefa então assumirá a prioridade da outra que criou o temporizador, executando assim apenas quando não existir nenhuma tarefa com prioridade maior. Desta forma ela acordará a devida tarefa no momento que esta já possa executar, respeitando assim as prioridades.

Para isso ser possível devem ser feitas algumas alterações no kernel do Linux, como criar uma árvore vermelha e preta, a qual armazenará estes temporizadores pela prioridade ligada a eles. Então em vez de processar esses temporizadores expirados em *hard IRQ*, eles devem ser transferidos da árvore vermelha e preta onde são ordenados por tempo de expiração para a árvore criada. Logo em seguida a tarefa criada para processar estes temporizadores deve ser acordada, tendo sua prioridade alterada de acordo com os temporizadores expirados.

Desta forma, caso haja uma tarefa T com prioridade P executando no processador e existam n temporizadores com prioridades menores que P, as quais irão expirar antes que a tarefa T termine de executar, esta tarefa não vai sofrer uma preempção tão grande. Isso pode ser verificado pela alteração do cálculo da variável C_{TIH} da equação (1) demonstrado pela equação (3).

$$C_{TIH} = \sum_{i=1}^{n} \Big(C_{TA}(i) \Big) \tag{3}$$

Onde $C_{TA}(i)$ é o tempo gasto para trocar o temporizador i de uma árvore vermelha e preta para outra. Lembrando que o tempo para remover ou inserir um temporizador em uma árvore vermelha e preta é $log(n)$, mas para a árvore criada para este trabalho o n tem um valor máximo de 100 nodos ($log(100) = 2$), pois os temporizadores são classificados entre as 100 prioridades de tempo real. Desta forma o tempo para realizar esta troca varia mais devido a quantidade de nodos na primeira árvore. Através da figura 4, pode-se verificar que o tempo gasto para trocar o temporizador de fila é bem menor que o tempo gasto para acordar a tarefa, como isso faz parte da proposta deste trabalho, realmente o tempo gasto em *hard irq* vai ser diminuído.

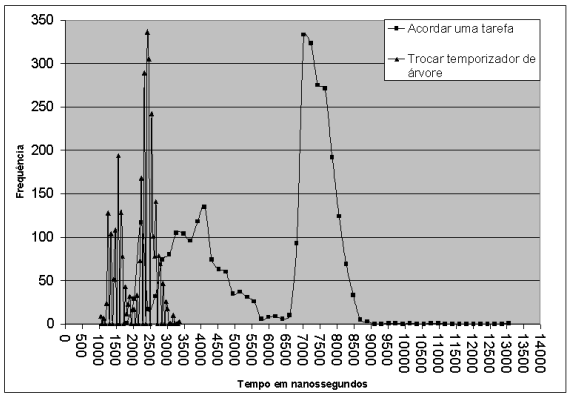

Figure 4. Variação de tempo para acordar uma tarefa e trocar um temporizador de árvore

Com estas alterações pretende-se diminuir o tempo em certas execuções do tratador de interrupção, diminuindo assim o tempo de preempção que ele causa às tarefas. Mas as tarefas que devem ser acordadas por estes temporizadores acabam recebendo algum acréscimo no atraso de sua liberação (*release jitter*) como pode ser visto na figura 3. Este acréscimo se dá porque a tarefa que já deveria estar na fila de prontos, ainda está na fila de espera e ainda deve mudar de fila através da execução da tarefa criada para isso. Então devido a execução desta tarefa que processará o temporizador, o atraso na liberação deverá ser pouco incrementado.

Aplicando-se as alterações sugeridas ao exemplo da figura 2, pode-se supor que o escalonamento deste exemplo ficaria semelhante ao exemplo da figura 3. Onde TP é a tarefa proposta para este trabalho e durante sua execução no exemplo, ele tem sua prioridade alterada para a prioridade da tarefa T2 e T3 respectivamente. Então através do exemplo pode-se verificar a diminuição do tempo de preempção que a tarefa T1 sofre, já a tarefa T2 não sofre alteração na sua execução, mas a tarefa T3 sofre um atraso maior na sua liberação.

6. Conclusão

Neste artigo foi proposto um método de postergar a execução de alguns temporizadores utilizados para acordar tarefas. De acordo com os cálculos apresentados neste artigo, pode-se ter uma prévia da diminuição do tempo de preempção de algumas tarefas, onde quanto maior a carga de temporizadores deste tipo, melhor se aplica este método, pois diminui consideravelmente o tempo gasto no tratador de interrupção.

O próximo passo deste trabalho será realizar as alterações necessárias no kernel do Linux, como criar a tarefa responsável por processar os temporizadores, adicionar alguns campos na estrutura dos temporizadores de alta resolução e alterar alguns trechos de código relacionado a eles. Depois testar, medir os tempos de execução, verificar a diferença na execução entre o *kernel* base e o alterado e por fim validar os cálculos baseado nos tempos da alteração realizada.

References

Etsion, Y., Tsafrir, D., and Feitelson, D. G. (2001). Effects of clock resolution on the scheduling of real-time and interactive processes. *School of Computer Science and Engineering*.

Gleixner, T. and Niehaus, D. (2006). Hrtimers and beyond: Transforming the linux time subsystems. *Proceedings of the Linux Symposium*.

Love, R. (2005). *Linux Kernel Development*. Novell.

Mauerer, W. (2008). *Professional Linux Kernel Architecture*. Wiley Publishing, Inc., Indianapolis.

Molnar, I. (2005). *PREEMPT-RT*. Disponível em: http://www.kernel.org/pub/linux/kernel/projects/rt - Último acesso em: 27 fev 2010.

POSIX.13 (1998). *IEEE Std. 1003.13-1998. Information Technology -Standardized Application Environment Profile-POSIX Realtime Application Support (AEP)*.