Implementação da política EDF no Xenomai

José Augusto Santos[1], **George Lima**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Campus de Ondina, 40170-110, Salvador-BA, Brasil

verne@dcc.ufba.br,gmlima@ufba.br

Abstract. *The scheduler of Xenomai, a Linux-based real-time operating system, is extended to provide support to the Earliest Deadline First (EDF) scheduling policy. The described EDF implementation is simple, compatible with the current Xenomai scheduler and provides support to hierarchical scheduling. The proposed implementation is evaluated by experiments, which show that task activation latency is not significantly altered.*

Resumo. *O escalonador do Xenomai, um sistema operacional de tempo real baseado em Linux, é estendido para dar suporte a política de escalonamento Earliest Deadline First (EDF). A implementação descrita do EDF é simples, compatível com o atual escalonador do Xenomai e fornece suporte para escalonamento hierárquico. A proposta de implementação é avaliada por experimentos, que mostram que a latência de ativação da tarefa não é significativamente alterada.*

1. Introdução

Nos últimos anos, mecanismos de tempo real têm sido introduzidos nos sistemas operacionais de propósito geral (SOPG) a fim de torná-los capazes de suportar aplicações com requisitos temporais. Neste contexto, extensões de Linux têm sido empregadas em sistemas embarcados e de tempo real. Uma destas iniciativas é o projeto Xenomai [Gerum 2004, Xenomai 2008b]. Este sistema operacional de tempo real (SOTR) emprega o conceito de *nanokernel*, o qual executa de forma cooperativa com o *kernel* do Linux, oferecendo alto grau de previsibilidade temporal [Regnier et al. 2008a, Regnier et al. 2008b]. Xenomai suporta aplicações Linux existentes, a integração destas com aplicações de tempo real e provê uma interface padronizada de auxílio ao desenvolvedor.

Um importante mecanismo de qualquer SOTR é o escalonador, pois este é peça-chave na garantia da correção temporal do sistema. Sua função é escolher qual tarefa deve ser executada em função do tempo. Como a grande maioria dos sistemas operacionais, o Xenomai utiliza políticas de escalonamento restritivas sob o ponto de vista de escalonamento. Por exemplo, atualmente, são oferecidos suporte a política FIFO, *round-robin* (RR) e prioridade fixa. Políticas baseadas em prioridade dinâmica, como o *Earliest Deadline First* (EDF) [Liu and Layland 1973] não são atualmente suportadas no Xenomai. De acordo com o EDF, a tarefa mais prioritária é aquela que possui o seu prazo máximo para execução (deadline) mais próximo a expirar. Instâncias de uma mesma tarefa, portanto, podem ter várias prioridades ao longo da execução do sistema, o que pode exigir maior custo de manutenção das filas de prioridade. No entanto, a política EDF tem sido recomendada. De fato, é sabido que EDF é uma política ótima em termos de escalonabilidade em sistemas monoprocessados, o que não é verdade para políticas baseadas em

prioridade fixa [Liu and Layland 1973, Stankovic et al. 1998]. Em outras palavras, se um sistema (monoprocessado) não é escalonável por EDF ele não será escalonável por nenhuma outra política de escalonamento. Outros aspectos relevantes relativos ao uso de EDF, tais como menor número de preempções, suporte à adaptabilidade do sistema etc. são citados em outras referências [Buttazzo 2005]. Neste contexto, portanto, é importante incorporar EDF ao suporte existente no SOTR.

Neste artigo descrevemos uma implementação da política EDF no Xenomai. Poucas alterações nas funções relacionadas ao escalonamento foram necessárias. Além disso, com a implementação proposta, é possível oferecer suporte a múltiplas políticas de escalonamento simultaneamente, característica recomendada por alguns autores [Harbour and Palencia 2003]. Resultados experimentais indicam que o desempenho da implementação EDF é compatível com aquele encontrado para a política de prioridade fixa nativa do Xenomai.

O restante do artigo é organizado como segue. A Seção 2 resume alguns projetos de sistemas operacionais de tempo real que oferecem suporte EDF. Alguns conceitos do Xenomai necessários para o entendimento da implementação proposta são resumidos na Seção 3. A descrição da implementação é abordada na Seção 4. Resultados experimentais são discutidos na Seção 5. Finalmente, na Seção 6 comentários finais são apresentados.

2. Trabalhos Relacionados

Existem diversos projetos que provêem suporte a sistemas de tempo real com licença não-proprietária, muitos dos quais estão focados em Linux. A maioria dos projetos existentes em uso comercial, contudo, possuem implementações restritivas sob o ponto de vista de escalonamento. Resumiremos aqui alguns destes projetos e recentes propostas de implementação de EDF em distribuições de SOTR com código aberto.

O Real-Time Executive for Multiprocessor Systems (RTEMS) [RTEMS 2009] é destinado à implementação de sistemas embarcados. Recentemente, uma implementação EDF para este sistema foi proposta [Molnar 2006]. Ao invés de aproveitar o escalonamento nativo, baseado em prioridades fixas, o escalonador proposto utiliza uma estrutura baseada em árvores vermelho-preto [Cormen et al. 2002] para ordenar as tarefas na fila de prontos. Apesar de melhorar o desempenho, em geral, esta opção de implementação não oferece suporte a múltiplas políticas de escalonamento.

O sistema operacional *microkernel* QNX [QNX 2009] é comercializado pela QNX Software and Systems e voltado para ambientes industriais, de telecomunicações e medicina. Seu modelo de distribuição é fechado, apesar da empresa ter disponibilizado o código. Escalonamento hierárquico é oferecido através de uma implementação de servidores esporádicos [QNX 2002]. No entanto, não há suporte à escalonamento baseado em políticas dinâmicas. Uma proposta de escalonador para outro sistema comercial, VxWorks [VxWorks 2009], foi recentemente publicada [Behnam et al. 2008]. O objetivo desta proposta foi incorporar escalonamento hierárquico e EDF. Porém, o mapeamento de prioridades dinâmicas para as estáticas, nativas do sistema, é computacionalmente cara.

Diversas implementações de EDF para Linux podem ser encontradas [Checchoni et al. 2009, Barreto 2004]. Para a maioria delas, como não há suporte a tarefas de tempo real críticas, não é possível fornecer garantias absolutas de cumprimento

de prazos de execução. No entanto, várias destas abordagens podem ser incorporadas no Preempt-RT [Preempt-RT 2008], uma extensão preemptível do Linux com altos níveis de previsibilidade temporal.

O Xenomai [Xenomai 2008b], RTAI [RTAI 2008] e Real-Time Linux (RT-Linux) [RT-Linux 2008b] são extensões do Linux que oferecem suporte a tarefas com restrições temporais críticas. Estes sistemas utilizam a abordagem de redirecionamento de interrupções, conhecida como *Interrupt abstraction*, onde um *kernel* de tempo real executa independente do *kernel* do SOPG. Atualmente o RT-Linux é mantido pela empresa FSMLabs e possui licença proprietária. A versão com licença GPL, mais defasada, está disponível [RT-Linux 2008a]. O Xenomai e RTAI são de código aberto e licença não-proprietária. Destes três sistemas operacionais, apenas o Xenomai não traz atualmente uma implementação do EDF. A implementação do EDF no RTAI é destinada a tarefas periódicas, que assumem a mais alta prioridade entre as demais tarefas da aplicação. Diferentemente desta abordagem e semelhantemente à implementação do EDF no RT-Linux, a proposta neste artigo é mais flexível, pois oferece suporte a escalonamento hierárquico [Harbour and Palencia 2003]. O EDF atua sobre as tarefas associadas à mesma fila de prioridade e outro escalonador pode ser usado em outra fila. A implementação aqui descrita suporta tanto tarefas periódicas quanto aperiódicas e a prioridade da fila EDF pode ser qualquer uma das disponíveis.

3. Visão Geral do Xenomai

Como mencionado, o Xenomai utiliza a abordagem conhecida como *Interrupt abstraction*, onde um *kernel* de tempo real executa independente do *kernel* do SOPG (Figura 1). A idéia é que estes dois sistemas executem lado-a-lado (na mesma máquina), utilizando um *microkernel*, o ADEOS [Xenomai 2005], que gerencia as interrupções geradas pelo hardware. O *kernel* do SOPG passa a ter prioridade mais baixa, comparada com as tarefas gerenciadas pelo SOTR. Assim, as interrupções são tratadas primeiro no SOTR e, caso não sejam destinadas a este, passam para o SOPG.

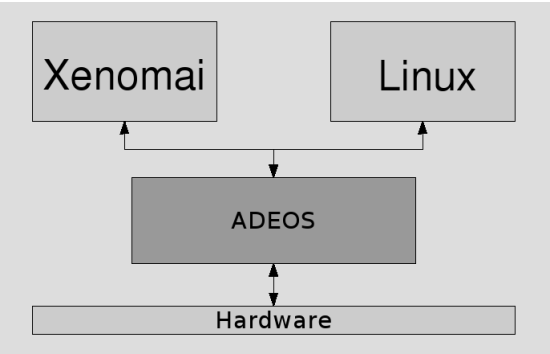

Figura 1. ADEOS e Xenomai.

A portabilidade dos programas é possível graças ao fato de o Xenomai ser, relativamente, independente em relação à API que pode ser utilizada, visto que ele permite que as tarefas de tempo real utilizem outras APIs para se comunicar com o seu *kernel*. Estas APIs são agrupadas num determinado conjunto, conhecido como *skin*. Assim, desde que haja uma *skin* para um determinado sistema, as aplicações desenvolvidas para este devem

funcionar adequadamente. Atualmente o Xenomai possui *skins* de diversos SOTR, tais como VxWorks, VRTX [VRTX 2009], RTAI, para o padrão POSIX e uma *skin* própria, denominada *Native*. Neste trabalho, a *skin Native* foi estendida para prover suporte ao EDF.

No Xenomai, uma tarefa de tempo real é representada por um *thread*. Usaremos então os termos *thread* e tarefa como sinônimos. As funções responsáveis pelas escolhas críticas de gerenciamento (*housekeeping*) e do escalonamento dos *threads* estão localizadas nos arquivos do *PrOcess Domain* (pod), `pod.c` e `pod.h`. Neles estão a função principal do escalonador (`xnpod_schedule`) e o *pod abstraction* [Osman et al. 2002](`xnpod_t`). Originalmente, este sistema possui implementações de escalonadores baseados em prioridade fixa, *round-robin* (RR) e FIFO. Por padrão, os *threads* são escalonados de acordo com um algoritmo de prioridade fixa preemptiva, onde entre os *threads* que possuem a mesma prioridade são escalonados segundo FIFO. Há também um suporte para escalonamento RR entre um grupo de *threads* tendo a mesma prioridade. É possível usar ainda ordenação crescente ou decrescente pela prioridade do *thread*, dependendo da configuração do sistema.

O Xenomai provê um objeto *thread* (`xnthread_t`), o qual representa as informações dos *threads*. O escalonador seleciona o *thread* de maior prioridade que esteja pronto para executar. Se houver mais de um *thread* nesta condição, aquele que está a mais tempo pronto para executar é selecionado. Quando tem sua execução interrompida por um *thread* de maior prioridade, o *thread* em execução é colocado no inicio da fila de *threads* prontos, desde que não tenha sido suspenso ou bloqueado. Assim, o *thread* interrompido deve recuperar a posse do processador assim que nenhuma outra atividade de prioridade mais elevada esteja elegível para execução.

Na próxima seção serão discutidas as alterações realizadas nas estruturas existentes no Xenomai e a inserção das novas funcionalidades neste sistema. Informações mais detalhadas sobre o Xenomai são dadas a medida que forem sendo necessárias.

4. Implementação do EDF no Xenomai

A implementação do EDF proposta neste artigo pode ser implantada através de um *patch*, que incorpora todas as alterações necessárias no Xenomai, versão número 2.4.0. Manteve-se a compatibilidade de configuração e instalação do sistema. As alterações foram feitas em 10 arquivos, totalizando, aproximadamente, 200 linhas de código. A maior parte destas alterações foram realizadas nos arquivos do pod.

O projeto de implementação buscou atender dois requisitos básicos. Primeiro, objetivou-se manter a compatibilidade com as políticas de prioridade fixa e chamadas de sistema existentes no Xenomai. Para tanto, usou-se a abordagem de aplicar a nova política numa fila de prioridade especificada pelo usuário, similarmente ao modo como RR e FIFO são incorporadas atualmente no Xenomai. O segundo requisito foi incorporar escalonamento hierárquico, cujo arcabouço teórico de análise vem sendo tratado por alguns autores [Harbour and Palencia 2003]. Por exemplo, com a implementação proposta, seria possível dividir as tarefas da aplicação em classes, cada classe alocada numa fila de prioridades distintas, e em cada fila pode-se usar EDF ou outra política como escalonador. Tal escalonamento em dois níveis não deixa que possíveis sobrecargas em classes menos prioritárias afetem as mais prioritárias. Como será visto posteriormente, os experimentos

realizados neste trabalho usam tal escalonamento em dois níveis.

No Xenomai os aplicativos geralmente são desenvolvidos utilizando uma *skin*, sendo a *Native* [Xenomai 2006], sua *skin* original. Na extensão desta *skin*, para suporte à nova política, foram criadas três novas chamadas de sistema, `rt_task_activate_edf`, `rt_task_deactivate_edf` e `rt_task_set_deadline`. A primeira e a segunda são destinadas a ativar e desativar, respectivamente, uma das filas manipuladas pelo Xenomai para usar a política EDF. A ativação para o EDF deve ser realizada antes da criação de qualquer tarefa escalonada por EDF. A última chamada de sistema tem como função atribuir o deadline relativo à tarefa e deve ser chamada para cada tarefa a ser escalonada por EDF, caso contrário ela será a menos prioritária. Para exemplificar, suponha que a tarefa `demo_task`, com código `demo` e deadline `dl`, será escalonada por EDF na fila de prioridade de número `pri`. Neste caso, após executar `rt_task_activate_edf(pri)`, a seguinte sequência deve ser executada:

1. `rt_task_create(&demo_task, "trivial", 0, pri, 0)`
2. `rt_task_set_deadline(&demo_task,dl)`
3. `rt_task_start(&demo_task, &demo, NULL).`

Em outras palavras, foi necessário apenas incluir o segundo passo quando comparamos com a sequência de criação de tarefas da *skin Native*. Os demais parâmetros acima, não descritos aqui, correspondem à interface mais simples para criação de tarefas. Estas chamadas da *skin*, na verdade, encapsulam as funções correspondentes do *kernel* e agregam algum mecanismo de tratamento para os parâmetros e checagem de contexto, visto que algumas funções só podem executar em determinado contexto. Desta forma, foi necessário ainda criar três funções do *kernel*, correspondentes àquelas da *skin*, são elas `xnpod_activate_edf`, `xnpod_deactivate_edf` e `xnpod_set_thread_deadline`. Além destas, foi criada uma função de inserção de *threads* por ordem de deadlines, chamada `insertpqo`. Nas seções seguintes, tais funções serão descritas.

4.1. Representação do Deadline

Para implantar a política EDF, foi necessário a inserção de dois novos atributos em `xnthread_t` (ver Seção 3) para representar o deadline relativo (`rdeadline`) e o absoluto (`adeadline`) dos *threads*. Note que tais atributos são necessários para definir a ordem de execução dos *threads*. De fato, a cada ativação de um *thread* no instante t, o valor de seu deadline, que definirá sua prioridade, será `adeadline` $= t +$ `rdeadline`. Ambos atributos são do tipo `xnticks_t`, usado para representar informações temporais. Além disso, algumas funções que manipulam atributos de *threads* foram modificadas.

A alteração do valor do campo `rdeadline` é realizada pela função `xnpod_set_thread_deadline`. Notar que o valor de `rdeadline` poderia ser definido na função `xnpod_init_thread`, responsável por criar um *thread*. Porém, neste caso, haveria a necessidade de alterar o código da aplicação caso fosse necessário alterar seu deadline relativo. Preferimos, portanto, usar a função `xnpod_set_thread_deadline`, semelhante ao modo de definição dos *threads* periódicos no Xenomai.

Como o valor de `adeadline` depende do instante de ativação do *thread*, sua atualização é realizada pelas funções que ativam os *threads* e suas instâncias no Xenomai, respectivamente, `xnpod_start_thread` e `xnpod_wait_thread_periodic`. A primeira é responsável por colocar um *thread* na fila de prontos quando ele é iniciado. Já a segunda função é chamada para iniciar as demais instâncias de *threads* periódicos ou esporádicos. Neste caso, quando um *thread* periódico é iniciado, a função `xnpod_start_thread` é chamada. A ativação das demais instâncias desse *thread* é realizada pela função `xnpod_wait_thread_periodic`. No caso de *threads* aperiódicos, apenas a função `xnpod_start_thread` é utilizada, pois assumimos que tais *threads* possuem uma única instância. Em outras palavras, ativações de instâncias de um *thread* aperiódico são modeladas como recriações do mesmo *thread*.

Mantivemos a compatibilidade com o Xenomai no que diz respeito à violação de *deadline* dos *threads*. Desta forma, a função `xnpod_wait_thread_periodic` retorna um código de erro caso o `adeadline` não seja cumprido, tal como ocorre no Xenomai quando a ativação de uma nova instância de um *thread* periódico ocorre antes da instância anterior finalizar.

A configuração inicial dos atributos de *threads* é realizada pela função `xnthread_init`, que também teve que ser adaptada para considerar os campos `rdeadline` e `adeadline`. O valor inicial do deadline relativo é nulo para indicar que este atributo ainda não foi configurado. O deadline absoluto recebe inicialmente valor infinito, representado pelo maior número que é suportado por `xnticks_t`. Assim, os *threads* que estão na fila do EDF para os quais não tenham sido informados seus deadlines não interferem na execução daqueles que já os possuem.

4.2. Seleção e Manipulação da Fila EDF

Como mencionado, a política de escalonamento padrão no Xenomai é baseada em prioridade fixa, com uma fila sendo associada a cada prioridade. A definição da política de escalonamento para *threads* na mesma fila é feita no instante da criação dos *threads*, através da função `xnpod_init_thread`, que indica se a política para o *thread* criado será RR ou FIFO. O *quantum* do RR é informado como parâmetro de `xnpod_activate_rr`. Desta forma, pode haver a coexistência de FIFO e RR numa mesma fila. A implementação de EDF seguiu o mesmo padrão, mas com algumas alterações. De fato, não faz sentido ter numa mesma fila *threads* escalonadas por EDF e outra política, pois isto seria uma fonte de imprevisibilidade no sistema.

Para informar a fila associada ao EDF, o atributo `prio_readyq_edf` foi adicionado a `xnsched_t`. Este tipo, contido na definição do tipo `xnpod_t`, representa informações do escalonador. O valor inicial (nulo) deste novo campo é definido por `xnpod_init`, função responsável por atribuir os valores iniciais aos atributos de `xnpod_t`. Desta forma, indica-se que inicialmente a política EDF não está associada a nenhuma fila. Quando a fila EDF é definida, o que é feito pela função `xnpod_activate_edf`, o valor `prio_readyq_edf` armazena o número da fila associado ao EDF. A função `xnpod_deactivate_edf` retorna o valor de `prio_readyq_edf` para nulo novamente. Atualmente, `prio_readyq_edf` é um inteiro. Futuramente, este tipo será um vetor de bits, o que permitirá que várias filas possam estar associadas a política EDF.

Para implantar preempções por prioridade numa mesma fila, funcionalidade necessária para EDF, foi preciso alterar duas funções do Xenomai, `xnpod_preempt_thread` e `xnpod_schedule`. Tais funções eram originalmente configuradas para tratar preempção apenas considerando filas distintas.

As filas de escalonamento do Xenomai, do tipo `xnpqueue_t`, são construídas como listas duplamente encadeadas. Os *threads* são inseridos numa fila com determinada prioridade através de funções como `insertpqf` (ordem FIFO) e `insertpql` (ordem LIFO). Estas serviram como base para a função `insertpqo`, um método de inserção ordenada, a qual é utilizada para inserir *threads* em ordem EDF.

5. Avaliações Experimentais

Testes experimentais foram realizados para avaliar a variação no desempenho introduzida pela implementação proposta. O Xenomai possui uma aplicação chamada `latency`, que consiste basicamente em medir a latência de ativação sofrida por uma tarefa periódica, escolhida para ser monitorada. Usando esta aplicação, dois tipos de experimentos foram conduzidos. Em ambos a tarefa monitorada foi definida para ter período de $1\,ms$. Para o primeiro tipo de experimento, nenhuma outra tarefa de tempo real além daquela monitorada foi considerada. No segundo tipo outras tarefas foram consideradas de forma a avaliar o efeito de carga no desempenho do escalonador. Os experimentos foram realizados numa máquina Pentium IV (monoprocessada), com frequência 2.66 GHz, 512 MB de memória RAM. O Xenomai, versão 2.4.0, foi integrado ao SOPG Debian Etch.

Os resultados experimentais encontrados são mostrados em tabelas, onde cada linha corresponde a $10\,min$ de execução do experimento reportado e seus valores estão em μs. Em outras palavras, cada linha de cada tabela exibe os resultados de aproximadamente 6×10^5 ativações da tarefa monitorada.

Observou-se em geral que a diferença entre os valores médios obtidos para os experimentos executados com configurações diferentes é muito pequena e próxima à precisão das medições (em ns). Tal fato, muitas vezes, dificulta a avaliação dos resultados. Portanto, consideramos como semelhantes as configurações que apresentam diferenças de resultados médios menores que $0,1\mu s$.

5.1. Resultados sem Carga

Inicialmente, a influência do *patch* proposto foi avaliada. A latência de ativação da tarefa monitorada foi medida em três configurações: (a) sem o *patch* proposto; (b) com o *patch*, mas com o escalonador EDF desativado; e (c) com o escalonador ativado e a tarefa escalona por EDF. Três níveis de prioridade para a tarefa monitorada foram considerados: baixa (prioridade 10), média (prioridade 50) e alta (prioridade 90). Os resultados são apresentados nas Tabelas 1 e 2.

Os resultados da configuração (a) servem como base de comparação para avaliar os valores obtidos nas configurações (b) e (c). Como pode ser notado na Tabela 1, o valor médio da latência de ativação é bem próximo do valor mínimo observado e o desvio padrão obtido é inferior a $0,1\mu s$.

Os resultados apresentados na Tabela 2 indicam que não houve aumento significativo dos valores de latência de ativação. Por exemplo, a latência média observada sofreu

	Latência (μs)			
Prioridade	Mínima	Máxima	Média	Desvio
Baixa	$1,013$	$11,696$	$1,237$	$0,090$
Média	$0,992$	$11,899$	$1,297$	$0,088$
Alta	$1,112$	$11,907$	$1,325$	$0,091$

Tabela 1. Latência de ativação sem aplicar o *patch* proposto.

um aumento de cerca de 6% se o *patch* for aplicado e de cerca de 8% caso o EDF esteja ativado. Estes valores referem-se ao código adicional incluído para manipulação da fila EDF. Notar que a latência máxima observada não foi superior a $12,2\mu s$ e o recomendado é que esta latência seja inferior a $100\mu s$ [Xenomai 2008a]. Outro ponto que vale a pena notar é que os desvios padrões observados continuaram muito pequenos.

	Latência (μs)							
	EDF desativado				EDF ativado			
Prioridade	Mínima	Máxima	Média	Desvio	Mínima	Máxima	Média	Desvio
Baixa	$1,111$	$11,770$	$1,310$	$0,092$	$1,130$	$12,188$	$1,360$	$0,100$
Média	$1,082$	$11,963$	$1,289$	$0,086$	$1,151$	$11,703$	$1,400$	$0,092$
Alta	$1,042$	$11,370$	$1,287$	$0,088$	$1,163$	$11,828$	$1,374$	$0,089$

Tabela 2. Latência de ativação com o *patch* proposto.

5.2. Resultados com Carga

Para medir a latência de ativação da tarefa monitorada em condições de carga, outras tarefas de tempo real, chamadas tarefas de carga, foram consideradas no experimento. Cada tarefa de carga possui tempo de execução de, aproximadamente, $2,5ms$ e período de $100ms$. Considerou-se dois níveis de carga, referentes a conjuntos de 5 e 15 tarefas. Desta forma, as tarefas de carga consomem cerca de 38% de processador. A intenção aqui não é sobrecarregar o sistema e sim verificar o desempenho de gerenciamento da fila EDF e sua adequação ao mecanismo de escalonamento já existente no Xenomai.

Ao longo do experimento a tarefa monitorada foi mantida com a mesma prioridade (nível 50) enquanto que as tarefas de carga assumiram prioridades baixa (nível 10), alta (nível 90) ou igual à prioridade da tarefa monitorada. Neste último caso, as tarefas de carga são escalonadas juntamente com a tarefa monitorada de acordo com EDF.

As Tabelas 3 e 4 mostram os resultados encontrados sem aplicar e aplicando o *patch* EDF proposto. Pode-se observar que os valores de latência aumentaram em ambas as tabelas quando comparamos com os resultados descritos na seção anterior. O aumento máximo observado não é maior que 20%, indicando que o desempenho é afetado em função do número de tarefas escalonadas, mesmo sem aplicar o *patch* EDF. Quando comparamos os valores com e sem a aplicação do *patch* nas Tabelas 3 e 4, no entanto, observamos que não há diferença significativa entre os valores observados, o que indica que o *patch* EDF não introduz alteração de desempenho relevante. É importante notar que as possíveis fontes de diminuição de desempenho são devido às alterações em `xnpod_schedule` e a utilização de `insertpqo`, funções que manipulam a fila EDF, o que é esperado. Por

exemplo, observa-se que quando a tarefa monitorada compartilha a mesma fila das tarefas de carga, o valor da latência passa para $1,612\mu s$, um aumento de $0,225\mu s$ quando o *patch* não é aplicado. De fato, com o compartilhamento da fila EDF, a manutenção da ordem de prioridades requer mais recursos computacionais para o EDF. Para as demais políticas do Xenomai, a inserção na fila é uma operação mais simples. Ainda assim, a latência máxima observada continuou inferior ao recomendado [Xenomai 2008a] e os desvios padrões observados mantiveram-se em décimos de μs.

	Latência (μs)							
	5 tarefas				15 tarefas			
Prioridade	Mínima	Máxima	Média	Desvio	Mínima	Máxima	Média	Desvio
Baixa	1,090	11,560	1,319	0,117	1,067	13,746	1,294	0,111
Igual	1,134	12,580	1,435	0,109	1,074	14,788	1,387	0,133
Alta	1,069	12,280	1,372	0,125	1,052	13,667	1,375	0,125

Tabela 3. Latência de ativação com carga e sem a aplicação do *patch*.

	Latência (μs)							
	5 tarefas				15 tarefas			
Prioridade	Mínima	Máxima	Média	Desvio	Mínima	Máxima	Média	Desvio
Baixa	1,109	12,499	1,427	0,131	1,033	14,455	1,439	0,151
Igual	1,063	11,427	1,341	0,105	1,161	14,017	1,612	0,192
Alta	1,158	12,256	1,456	0,127	1,102	13,587	1,531	0,168

Tabela 4. Latência de ativação com carga e aplicado o *patch*.

Como pode ser notado, os experimentos relatados acima comprovam que a implementação não induz uma diminuição significativa do desempenho em termos de manipulação da fila de prioridades. Desta forma, acreditamos que, devido à otimalidade do EDF [Liu and Layland 1973], os resultados para a aplicação em termos de escalonabilidade sejam promissores. Este aspecto deve ser melhor investigado em trabalhos futuros. Para tal avaliação, é necessário considerar diferentes tipos de aplicação e seus requisitos temporais.

6. Conclusões e Trabalhos Futuros

Apesar de ser uma política de escalonamento amplamente difundida, EDF não é oferecida em diversos sistemas operacionais de tempo real. Neste trabalho descrevemos uma implementação de EDF no Xenomai. A implementação proposta preserva a compatibilidade com os mecanismos de escalonamento existentes neste sistema e pode ser usada para prover suporte a escalonamento hierárquico. Resultados experimentais indicaram que o desempenho do escalonador não é significativamente afetado pela implementação proposta. É necessário ainda avaliar o comportamento de aplicações de tempo real usando o *patch* e as políticas de escalonamento atualmente existentes no Xenomai, pois os resultados aqui apresentados referem-se apenas ao desempenho do escalonador. Além disso, trabalhos futuros devem incluir tratamento de tarefas aperiódicas com proteção temporal, recursos compartilhados e suporte a sistemas multiprocessados ou multinúcleos. Neste contexto, os resultados aqui apresentados devem servir de ponto de partida para tais trabalhos.

Agradecimentos

Este trabalho recebeu o apoio do PIBIC e da FAPESB, projeto número 7320/2007. Agradecemos a Luciano Barreto e aos revisores anônimos do WSO 2009 pelos comentários que ajudaram no desenvolvimento deste artigo.

Referências

Barreto, L. (2004). "Uma Arquitetura Baseada em Eventos para Desenvolvimento de Políticas de Escalonamento de Processos". *1º Workshop de Sistemas Operacionais*.

Behnam, M., Nolte, T., Shin, I., and Asberg, M. (2008). "Towards Hierarchical Scheduling in VxWorks". In *4th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 63–71.

Buttazzo, G. C. (2005). "Rate Monotonic vs. EDF: Judgment Day". *Real-Time Systems*, 29(1):5–26.

Checchoni, F., Trimarchi, M., and Faggioli, D. (2009). "An Implementation of the Earliest Deadline First Algorithm in Linux". In *24th ACM Symposium on Applied Computing*, pages 1984–1989.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2002). *Algoritmos: teoria e prática*. Editora Campus.

Gerum, P. (2004). "Xenomai - Implementing a RTOS Emulation Framework on GNU/Linux". Technical report, Xenomai, `http://www.xenomai.org`.

Harbour, M. G. and Palencia, J. C. (2003). "Response Time Analysis for Tasks Scheduled Under EDF Within Fixed Priorities". In *24th IEEE International Real-Time Systems Symposium*, page 200. IEEE Computer Society.

Liu, C. L. and Layland, J. W. (1973). "Scheduling Algorithms for Multiprogram in a Hard Real-Time Environment". *Journal of ACM*, 20(1):46–61.

Molnar, M. (2006). The EDF Scheduler Implementation in RTEMS Operating System. Master's thesis, Czech Technical University in Prague. Faculty of Electrical Engineering.

Osman, S., Subhraveti, D., Su, G., and Nieh, J. (2002). "The Design and Implementation of Zap: A System for Migrating Computing Environments". *ACM SIGOPS Operating Systems Review*, 36(SI):361–376.

Preempt-RT (2008). Preempt-RT howto. `http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO`.

QNX (2002). "Maximizing Performance with SMP: Design Application to Run on Multiple,Tightly Coupled Processors". Technical report, QNX Software System.

QNX (2009). Qnx Home Page. `http://www.qnx.com`.

Regnier, P., Lima, G., and Barreto, L. (2008a). "Avaliação do Determinismo Temporal no Tratamento de Interrupções em Plataformas de Tempo Real Linux". In *5º Workshop de Sistemas Operacionais*, pages 13–24.

Regnier, P., Lima, G., and Barreto, L. (2008b). "Evaluation of Interrupt Handling Timeliness in Real-Time Linux Operating Systems". *ACM SIGOPS Operating Systems Review*, 42(6):52–63.

RT-Linux (2008a). RT-Linux GPL Home Page. http://www.rtlinux-gpl.org.

RT-Linux (2008b). RT-Linux Home Page. http://www.rtlinux.org.

RTAI (2008). RTAI Home Page. http://www.rtai.org.

RTEMS (2009). Rtems Home Page. http://www.rtems.com.

Stankovic, J. A., Spuri, M., Ramamritham., K., and Buttazzo, G. C. (1998). *Deadline Scheduling For Real-Time Systems EDF and Related Algorithms*. Kluwer Academic Publishers, 1^{st} edition.

VRTX (2009). Mentor Graphics Home Page. http://www.mentor.com.

VxWorks (2009). Wind river Systems Home Page. http://www.windriver.com.

Xenomai (2005). "Life with Adeos". Technical report, Xenomai, http://www.xenomai.org.

Xenomai (2006). "A Tour of the Native API". Technical report, Xenomai, http://www.xenomai.org.

Xenomai (2008a). Xenomai-core Mailing List. https://mail.gna.org/listinfo/xenomai-core.

Xenomai (2008b). Xenomai Home Page. http://www.xenomai.org.