Avaliação do determinismo temporal no tratamento de interrupções em plataformas de tempo real Linux *

Paul Regnier [1] **George Lima** [1] **Luciano Barreto** [1]

{pregnier, gmlima, lportoba}@ufba.br

[1] Distributed Systems Laboratory (LaSiD) - Computer Science Department (DCC)
Pos-Graduation Program on Mechatronics - Federal University of Bahia
Campus de Ondina, 40170-110, Salvador-BA, Brazil

Abstract. *Several real-time Linux extensions can be found nowadays. Two of them have received special attention recently, the patches Preempt-RT and Xenomai. This paper evaluates to what extent they provide deterministic guarantees when reacting to external events, an essential characteristic when it comes to real-time systems. To do that we define a simple but effective experimental approach. Obtained results indicate that Preempt-RT is more prone to temporal variations than Xenomai when the system is subject to overload scenarios.*

Resumo. *Várias extensões Linux para tempo real podem ser encontradas hoje em dia. Duas delas têm recebido atenção especial recentemente: os* patches *Preempt-RT e Xenomai. Este artigo avalia em que medida elas fornecem garantias determinísticas quando reagem a eventos externos, uma característica essencial quando se trata de sistemas de tempo real. Para tanto, definimos uma abordagem experimental simples e eficaz. Os resultados obtidos indicam que Preempt-RT é mais propenso a variações temporais que Xenomai quando o sistema está sujeito a cenários de sobrecarga.*

1. Introdução

Os sistemas de tempo real englobam diversas aplicações ligadas às áreas de telecomunicações, multimídia, indústria, transporte, medicina, etc. Para tais sistemas, a escolha adequada de Sistemas Operacionais de Tempo Real (SOTR) constitui aspecto fundamental de projeto. Apesar da importância da evolução tecnológica do hardware, certas inovações podem introduzir empecilhos para a construção de SOTR. Por exemplo, os advents de memória *cache*, acesso direto à memória (DMA), co-processamento, predição de instruções, unidades *multicore*, *pipelines* e execução fora de ordem constituem fontes não-desprezíveis de indeterminismo [Liu 2000, Pratt and Heger 2004]. Assim, a construção de um sistema operacional de uso genérico com foco em previsibilidade temporal permanece um desafio de pesquisa atual.

Apesar de sua difusão e popularidade, o *kernel* padrão do Linux [D. P. Bovet 2005] falha na oferta de garantias temporais típicas dos sistemas de tempo real críticos [Marchesotti et al. 2006, Abeni et al. 2002]. Para contornar esse problema, várias abordagens foram desenvolvidas com o intuito de aumentar o grau de previsibilidade temporal do Linux [I. Molnar et al. 2008, P. Gerum et al. 2008,

*Este trabalho recebeu o apoio da FAPESB e do CNPq

Dozio et al 2003, V. Yodaiken et al. 2008, Fry and West 2007, Calandrino et al. 2006]. A evolução rápida e a diversidade das soluções disponíveis apela para estudos comparativos que permitam avaliar o determinismo oferecido por cada plataforma de forma a auxiliar o projetista de sistemas de tempo real na escolha apropriada da distribuição a ser utilizada.

Este trabalho tem por objetivo apresentar e comparar os *patches* do *kernel* Linux Preempt-RT (Linux$^{\mathbf{Prt}}$) [I. Molnar et al. 2008] e Xenomai (Linux$^{\mathbf{Xen}}$) [P. Gerum et al. 2008] , desenvolvidos para aumentar a previsibilidade do Linux. As principais contribuições deste trabalho são: (i) a proposta de uma metodologia de avaliação simples baseada em software e hardware de prateleira, que utiliza uma sobrecarga do processador através de carga de processamento, de entrada e saída, e de interrupção; (ii) a obtenção de resultados que confirmam a capacidade de Linux$^{\mathbf{Xen}}$ em oferecer garantias temporais críticas, e (iii) a confirmação que, em situação de carga intensa, Linux$^{\mathbf{Prt}}$ não consegue oferecer garantias temporais de maneira tão determinística quanto Linux$^{\mathbf{Xen}}$.

A Seção 2 descreve alguns fatores de imprevisibilidade do Linux e define as métricas adotadas para a comparação de Linux$^{\mathbf{Prt}}$ e Linux$^{\mathbf{Xen}}$. Estas duas plataformas são descritas nas Seções 3 e 4. Em seguida, a descrição da metodologia de experimentação na Seção 5 precede a apresentação dos resultados experimentais na Seção 6. Finalmente, a Seção 7 apresenta brevemente os trabalhos relacionados e a Seção 8 conclui este trabalho.

2. Métricas de comparação

O método convencional utilizado para minimizar o impacto das interrupções sobre a execução dos processos consiste em dividir a execução do tratador de interrupção em duas partes. A primeira parte executa operações críticas de forma imediata e com as interrupções desabilitadas, o que constitui a **seção crítica** do tratador. Eventualmente, pode-se reabilitar as interrupções de modo a permitir preempção, tomando-se o cuidado de garantir o acesso controlado aos dados compartilhados através de *locks*. Na segunda parte, as operações não-críticas são possivelmente adiadas e executadas com as interrupções habilitadas. No Linux, estas execuções postergadas são chamadas de ***softirqs***.

2.1. Latência de interrupção

Uma requisição de interrupção, ou simplesmente **interrupção**, do processador por um dispositivo de *hardware* é assíncrona e pode acontecer em qualquer momento do ciclo de execução do processador. Em particular, tal requisição pode ocorrer enquanto a seção crítica do tratador de outra interrupção estiver executando, com as interrupções desabilitadas. Tal cenário pode provocar uma latência não determinística para a detecção da requisição de interrupção pelo processador. O tempo decorrido entre o instante no qual uma requisição de interrupção acontece e o início da execução do tratador associado é chamado de **latência de interrupção**. Esta grandeza foi contemplada como métrica para efeito de comparação das plataformas estudadas, pois caracteriza a capacidade do sistema para reagir a eventos externos.

2.2. Latência de ativação

No *kernel* Linux, logo após o término da seção crítica do tratador de interrupção, o *softirq* correspondente está apto a executar. No entanto, entre o instante no qual a seção crítica termina e o instante no qual o *softirq* começa a executar, outras interrupções podem acontecer, provocando um possível atraso na execução dos *softirqs*.

Nas plataformas de tempo real, eventos de temporizadores ou de *hardware* são utilizados para disparar tarefas, num modelo similar aos *softirqs*. Tal tarefa, muitas vezes periódica, fica suspensa a espera de um evento. Quando o evento ocorre, a requisição de interrupção associada aciona o tratador correspondente que, por sua vez, acorda a tarefa. O intervalo de tempo entre os instantes de ocorrência do evento e o início da execução da tarefa associada é chamada de **latência de ativação**. Assim como no caso dos *softirqs*, a latência de ativação pode ser aumentada pela ocorrência de interrupções. Além disso, a execução de outros *softirqs* pode ser escalonada de acordo com alguma política (ex: FIFO, prioridade fixa), o que pode também gerar interferências na latência de ativação. Assim como a latência de interrupção, a latência de ativação caracteriza a capacidade de um sistema para reagir a eventos externos.

3. Linux$^{\mathbf{Prt}}$

Para prover precisão temporal, Linux$^{\mathbf{Prt}}$ [McKenney 2005, Rostedt and Hart 2007] utiliza uma nova implementação dos temporizadores de alta resolução desenvolvida por Thomas Gleixner [L. Torvalds et al. 2008]. Baseado no valor no registrador *Time Stamp Counter* (TSC) da arquitetura Intel ou em relógios de alta resolução, esta implementação oferece uma API que permite obter valores temporais com resolução de micro-segundos. De acordo com resultados apresentados [Rostedt and Hart 2007, Siro et al. 2007], os tempos de latência de ativação obtidos usando esta API são da ordem de algumas dezenas de μs nos computadores atuais.

Linux$^{\mathbf{Prt}}$ comporta várias modificações que tornam o *kernel* totalmente preemptível. Dessa forma, assim que um processo de mais alta prioridade é desbloqueado, este consegue adquirir o processador com latência mínima, sem necessidade de espera pelo fim da execução de um processo de menor prioridade, mesmo que este esteja executando em modo *kernel*. Para limitar os efeitos de imprevisibilidade causados por recursos compartilhados, Linux$^{\mathbf{Prt}}$ modifica as primitivas de sincronização de maneira a permitir a implementação de um protocolo baseado em herança de prioridade [Sha et al. 1990].

Em relação à latência de interrupção, Linux$^{\mathbf{Prt}}$ utiliza *threads* de interrupções. Quando uma linha de interrupção é iniciada, um *thread* é criado para gerenciar as requisições de interrupção associadas a esta linha. Na ocorrência de uma requisição, o tratador associado mascara a requisição, acorda o *thread* da interrupção e volta para o código interrompido. Desta forma, a parte crítica do tratador de interrupção é reduzida ao seu mínimo e a latência causada pela sua execução, além de ser breve, é determinística. Em algum momento futuro, o *thread* de interrupção acordado é escalonado, de acordo com a sua prioridade, dando espaço para indeterminismo, o que será objeto do presente estudo.

Usando programas corretamente escritos, respeitando as regras de programação de Linux$^{\mathbf{Prt}}$ e alocando os recursos de acordo com os requisitos temporais, a solução Linux$^{\mathbf{Prt}}$ tem a vantagem de oferecer o ambiente de programação do sistema Linux, dando acesso às bibliotecas C e ao conjunto de software disponível para este sistema.

4. Linux$^{\mathbf{Xen}}$

Diferentemente de Linux$^{\mathbf{Prt}}$, a plataforma Linux$^{\mathbf{Xen}}$ utiliza uma abordagem baseada no mecanismo de indireção das interrupções introduzido na técnica de "proteção otimista das interrupções" [Stodolsky et al. 1993]. Quando uma requisição de interrupção acontece, a

camada de indireção, também chamada de *nanokernel*, identifica se esta é relativa a uma tarefa de tempo real ou se é destinada a um processo do Linux. No primeiro caso, o tratador da interrupção é executado imediatamente. Caso contrário, a requisição é enfileirada e, em algum momento futuro, entregue para o Linux quando inexistirem tarefas de tempo real. Quando o Linux precisa desabilitar as interrupções, o *nanokernel* apenas deixa o Linux acreditar que as interrupções estão desabilitadas. No entanto, o *nanokernel* continua a interceptar qualquer interrupção de *hardware*. Nesta ocorrência, a interrupção é tratada imediatamente se for destinada a uma tarefa de tempo real. Caso contrário, a interrupção é enfileirada, até que o *kernel* Linux reabilite suas interrupções.

Para a implementação do *nanokernel*, Linux$^{\mathbf{Xen}}$ utiliza uma camada de virtualização dos recursos chamada Adeos (*Adaptative Domain Environment for Operating Systems*) [Yaghmour 2001]. O *patch* Adeos facilita o compartilhamento e o uso dos recursos de *hardware* e oferece uma interface de programação simples e independente da arquitetura. Resumidamente, Adeos é baseado nos conceitos de domínio e de canal hierárquico de interrupção. Um domínio caracteriza um ambiente de execução isolado, no qual pode-se executar programas ou até mesmo sistemas operacionais completos. O canal hierárquico de interrupção, chamado *ipipe*, serve para priorizar a entrega da interrupções entre os domínios. Quando um domínio se registra no Adeos, ele é colocado numa posição no *ipipe* de acordo com os seus requisitos temporais. Adeos utiliza então o mecanismo de indireção das interrupções para organizar a entrega hierárquica das interrupções, seguindo a ordem de prioridade dos domínios.

Os serviços de tempo real de Linux$^{\mathbf{Xen}}$ correspondem ao domínio mais prioritário do *ipipe*, chamado "domínio primário". Este domínio corresponde, portanto, ao núcleo de tempo real no qual as tarefas são executadas em modo protegido. O "domínio secundário", por sua vez, corresponde ao *kernel* Linux, no qual o conjunto de bibliotecas e software usual do Linux está disponível. Em contrapartida, as garantias temporais são mais fracas, dado que o código pode utilizar as chamadas de sistemas bloqueantes do Linux.

5. Metodologia experimental

Em geral, realizar medições precisas de tempo no nível dos tratadores de interrupção pode não ser tão simples. De fato, o instante exato no qual uma requisição de interrupção acontece é de difícil medição, pois tal evento é assíncrono e pode ser causado por qualquer dispositivo de *hardware*. Para obter medidas das latências de interrupção e ativação confiáveis com alto grau de precisão, aparelhos externos, tais como osciloscópios ou outros computadores, são necessários. Visto que o objetivo do presente trabalho foi caracterizar e comparar o grau de determinismo das plataformas operacionais estudadas, adotou-se uma metodologia experimental simples e efetiva, que pode ser reproduzida facilmente em outros contextos.

5.1. Configuração do experimento

O dispositivo experimental utilizou três estações: (1) a estação de medição E_M, na qual os dados foram coletados e onde temos uma tarefa de tempo real τ à espera de eventos externos; (2) a estação de disparo E_D, que foi utilizada para enviar pacotes Ethernet com uma freqüência fixa à estação E_M; e (3) a estação de carga E_C, utilizada para criar uma carga de interrupção na estação E_M. As estações de disparo e carga foram conectadas

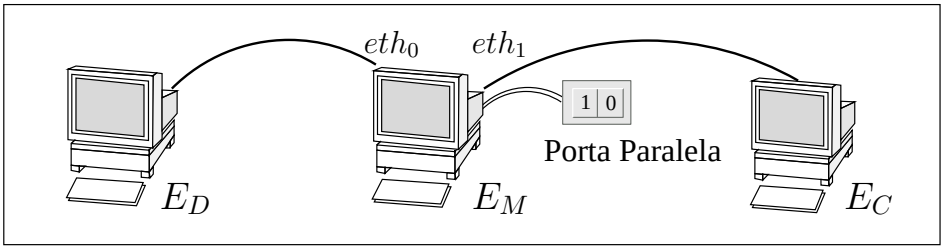


Figura 1: Configuração do experimento.

a estação de medição por duas redes Ethernet distintas, utilizando duas placas de redes (eth_0 e eth_1), conforme ilustrado no diagrama da Figura 1.

As chegadas dos pacotes enviados por E_D em E_M servem para disparar uma cascata de eventos na estação E_M, permitindo a simulação de eventos externos via sua porta paralela (PP). Mais explicitamente, cada chegada de um pacote Ethernet em eth_0 foi utilizada para disparar uma requisição de interrupção na PP, escrevendo no pino de interrupção desta porta (ver Figura 2). Esta escrita foi realizada pelo próprio tratador T_{eth_0} de interrupção da placa eth_0. O instante t_1 de escrita no pino de interrupção da PP pelo tratador T_{eth_0} constitui então o início da seqüência de eventos utilizados para medir as latências de interrupção (Lat_{irq}) e de ativação (Lat_{ativ}). Vale observar que não existe relação entre a chegada de pacotes na placa eth_0 de E_M e a atividade sendo executada nesta estação.

As medidas seguiram o seguinte roteiro, ilustrado pela Figura 2:

- A estação E_D envia pacotes Ethernet para a placa eth_0 da estação E_M, provocando interrupções assíncronas em relação às aplicações executando em E_M.
- A interrupção associada à chegada de um pacote provoca a preempção da aplicação em execução no processador pelo tratador de interrupção T_{eth_0}.
- O tratador T_{eth_0} foi restrito ao seu mínimo: ele apenas escreve no pino de interrupção da PP e armazena o instante t_1 na memória. Este instante corresponde, portanto, ao valor lido no relógio local, no instante na escrita da PP, logo após a chegada de uma pacote Ethernet.
- A requisição de interrupção associada à escrita no pino de interrupção da PP provoca a preempção da aplicação em execução no processador pelo tratador de interrupção T_{PP}.

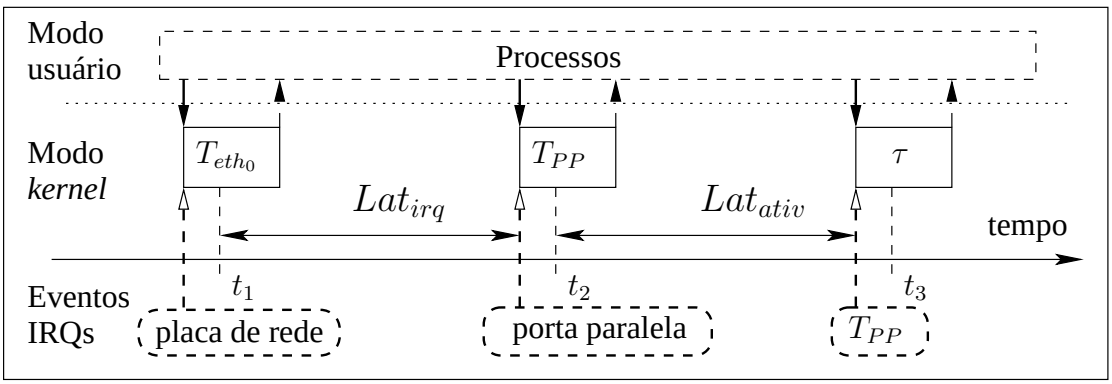


Figura 2: Cálculo das latências de interrupção e ativação na estação E_M.

- T_{PP} grava o instante t_2 e acorda a tarefa τ. Este valor t_2 corresponde ao valor do relógio local, logo após o início de T_{PP}.
- No momento que a tarefa τ acorda, ela grava o instante t_3 e volta a ficar suspensa até a próxima interrupção na PP. Portanto, t_3 corresponde ao tempo no qual a tarefa τ começa a executar, no final da cascata de eventos provocada pela chegada de um pacote na placa de rede.

Como representado na Figura 2, Lat_{irq} corresponde à diferença $t_2 - t_1$ e Lat_{ativ} a diferença $t_3 - t_2$. No decorrer do experimento, a transferência das medições da memória para o sistema de arquivos foi realizada por um canal FIFO lido por um processo usuário de forma a impedir qualquer interferência entre a aquisição dos dados e seu armazenamento no sistema de arquivos. Além da prioridade deste processo ser menor que as demais tarefas e tratadores de interrupções executados em modo *kernel*, os eventos de transferências de dados eram suficientemente raros (20 por segundo) para não interferir nas medidas realizadas.

5.2. Cargas de I/O, processamento e interrupção

Num primeiro momento, realizou-se experimentos com uma carga mínima no processador da estação E_M (modo *single*). Desta forma, observou-se o comportamento temporal das três plataformas em situação favorável. Em seguida, dois tipos de cargas foram utilizados simultaneamente para sobrecarregar E_M. Tais sobrecargas tiveram por objetivo avaliar a capacidade de cada plataforma em garantir uma latência determinística no tratamento das interrupções e na ativação de tarefas de tempo real, apesar da existência de outras atividades não-críticas. As cargas de I/O e processamento foram realizadas executando as instruções seguintes na estação E_M:

```
while "true"; do
      dd if=/dev/hda2 of=/dev/null bs=1M count=1000
      find / -name "*.c" | xargs egrep include
      tar -cjf /tmp/root.tbz2 /usr/src/linux-xenomai
      cd /usr/src/linux-preempt; make clean; make
done
```

Um outro estresse de interrupção foi criado utilizando uma comunicação UDP entre a estação E_M configurada como servidor e a estação E_C configurada como cliente. Para isolar esta comunicação da comunicação entre E_M e E_D, utilizou-se a segunda placa de rede eth_1 de E_M, assim com ilustrado pelo diagrama da Figura 2. Durante o experimento, o processo cliente hospedado pela estação E_C transmitiu pequenos pacotes de 64 *bytes* na freqüência máxima permitida pela rede, ou seja, com uma freqüência superior a $200kHz$ (um pacote a cada $10\mu s$). Desta forma, mais de 100.000 interrupções por segundo foram geradas pela placa eth_1 de E_M. A placa $eth1$ foi registrada na linha de interrupção 18 de E_M, cuja prioridade é menor que a prioridade da porta paralela. Nos experimentos com cargas, os dois tipos de estresses foram aplicados simultaneamente e as medições só foram iniciadas alguns segundos depois.

6. Avaliação de Linux$^{\mathbf{Prt}}$ e Linux$^{\mathbf{Xen}}$

6.1. Configuração

Os experimentos foram realizados em computadores Pentium 4 com processadores de 2.6 GHz e 512 Mb de memória, com o objetivo de ilustrar o comportamento temporal das três

plataformas seguintes:

- **Linux$^{\mathbf{Std}}$:** Linux padrão - *kernel* versão 2.6.23.9 (opção *low-latency*);
- **Linux$^{\mathbf{Prt}}$:** Linux com o *patch* Preempt-RT (rt12) - *kernel* versão 2.6.23.9;
- **Linux$^{\mathbf{Xen}}$:** Linux com o *patch* Xenomai - versão 2.4-rc5 - *kernel* versão 2.6.19.7.

Utilizou-se a configuração Linux$^{\mathbf{Std}}$ para realizar experimentos de referência para efeito de comparação com as duas plataformas de tempo real Linux$^{\mathbf{Prt}}$ e Linux$^{\mathbf{Xen}}$. A versão estável do *kernel* 2.6.23.9, disponibilizada em dezembro de 2007 foi escolhida para o estudo de Linux$^{\mathbf{Prt}}$, pois este *patch* tem evoluído rapidamente desde sua primeira versão publicada há dois anos. No entanto, utilizou-se a versão do *kernel* 2.6.19.7 para o estudo da versão 2.4-rc5 de Xenomai. De fato, considerou-se desnecessário atualizar a versão do *kernel*, pois Xenomai é baseado no Adeos (ver Seção 4) e, portanto, as garantias temporais oferecidas para as aplicações executando no primeiro domínio dependem apenas da versão do Xenomai e do *patch* Adeos associado, e não, da versão do *kernel* Linux.

As medidas das latências de interrupção e de ativação foram realizadas pela consulta do *Time Stamp Counter* (TSC), permitindo uma precisão abaixo de $30ns$ (88 ciclos), verificada experimentalmente. Utilizou-se uma freqüência de disparo dos eventos pela estação E_D de 20Hz. Para cada plataforma, dois experimentos de 10 minutos foram realizados. O primeiro sem carga nenhuma do sistema e o segundo aplicando os estresses apresentados na Seção 5.2.

6.2. Resultados experimentais

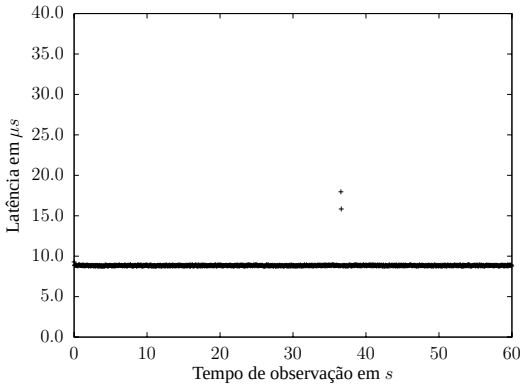
Os resultados experimentais são apresentados nas Figuras 3 e 4, onde o eixo horizontal representa o instante de observação variando de 0 a 60 segundos e o eixo vertical representa as latências medidas em μs (tais valores podem ser multiplicados por 2.610^3 para obter o número de ciclos do TSC). Apesar de cada experimento ter durado no mínimo uma hora, escolheu-se apresentar apenas resultados para um intervalo de $60s$, pois este intervalo é suficiente para observar o padrão de comportamento de cada plataforma. Neste intervalo, o total de eventos por experimentos é 1200, pois a freqüência de chegada de pacotes utilizada foi de $20Hz$.

Abaixo de cada Figura, os seguintes valores são indicados: Valor Médio (VM), desvio padrão (DP), valor mínimo (Min) e valor máximo (Max). Estes valores foram obtidos considerando a duração de uma hora de cada experimento. Na medida do possível, utilizou-se a mesma escala vertical para todos os gráficos. Conseqüentemente, alguns valores altos podem ter ficado fora das Figuras. Tal ocorrência foi representada por um triângulo próximo do valor máximo do eixo vertical.

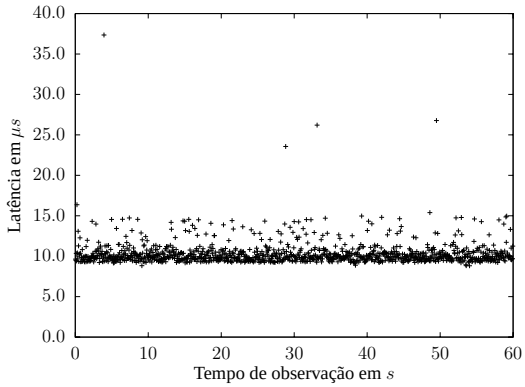
6.2.1. Latência de interrupção

A Figura 3 apresenta as latências de interrupção medidas, com e sem estresse do sistema. Como pode ser observado, sem carga, o Linux$^{\mathbf{Std}}$ e o Linux$^{\mathbf{Xen}}$ tem comportamentos parecidos. Com carga, observa-se uma variação significativa do Linux$^{\mathbf{Std}}$, como esperado.

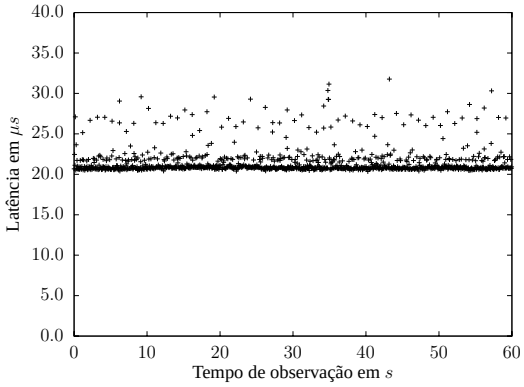
Com relação ao Linux$^{\mathbf{Prt}}$, dois resultados chamam atenção. Primeiro, o comportamento do sistema sem carga exibe latências da ordem de 20 μs. Isto é causado pela implementação dos *threads* de interrupção vista na Seção 3. Segundo, contradizendo as



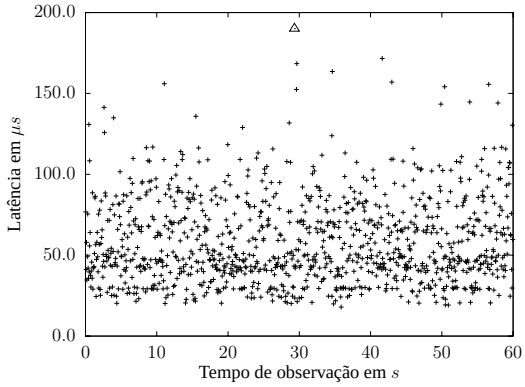
(a) **Linux$^{\mathbf{Std}}$ - Sem carga**
VM: 8.9, DP: 0.3, Min: 8.7, Max: 18.4



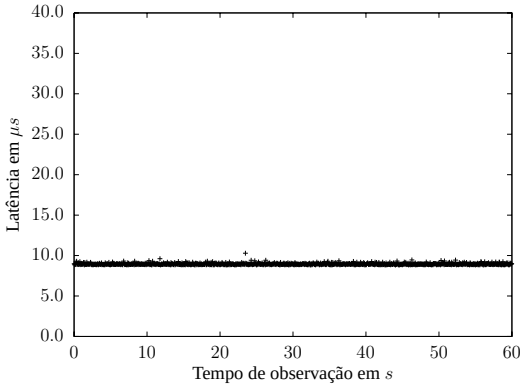
(b) **Linux$^{\mathbf{Std}}$ - Com carga**
VM: 10.4, DP: 1.9, Min: 8.8, Max: 67.7



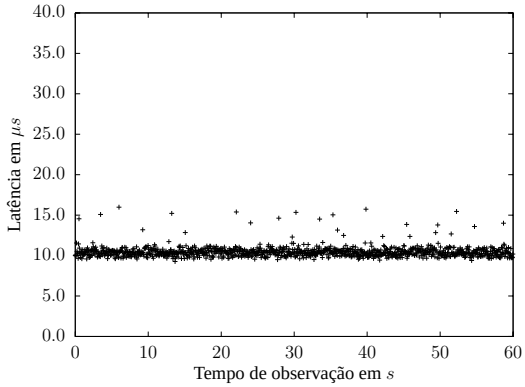
(c) **Linux$^{\mathbf{Prt}}$ - Sem carga**
VM: 21.5, DP: 1.7,
Min: 20.3, Max: 45.1



(d) **Linux$^{\mathbf{Prt}}$ - Com carga**
VM: 58.5, DP: 26.4,
Min: 17.2, Max: 245.9



(e) **Linux$^{\mathbf{Xen}}$ - Sem carga**
VM: 9.0, DP: 0.1, Min: 8.8, Max: 11.1



(f) **Linux$^{\mathbf{Xen}}$ - Com carga**
VM: 10.2, DP: 0.1, Min: 8.8, Max: 20.8

Figura 3: Latência de interrupção com freqüência de escrita na PP de $20Hz$.

expectativas, a aplicação do estresses teve um impacto significativo, provocando uma alta variabilidade das latências. De fato, entre o instante no qual o tratador T_{PP} requer a interrupção do processador e o instante no qual este *thread* acorda efetivamente, uma ou várias interrupções podem ocorrer. Neste caso, a execução dos tratadores associados pode provocar o atraso da execução de T_{PP}.

Para cancelar esta variabilidade indesejável, é possível usar o Linux$^{\mathbf{Prt}}$ sem utilizar a implementação de *threads* de interrupção. Para tanto, usa-se a opção `IRQF_NODELAY` na requisição inicial da linha de interrupção. Utilizando esta opção na definição da linha de interrupção da porta paralela, o comportamento do Linux$^{\mathbf{Prt}}$ passa a ser semelhante ao Linux$^{\mathbf{Std}}$.

6.2.2. Latência de ativação

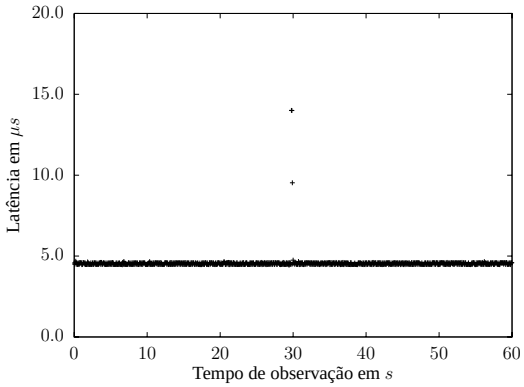
A Figura 4 apresenta os resultados para as latências de ativação sem estresse e com estresse do processador. Como pode ser observado, o comportamento de Linux$^{\mathbf{Std}}$ é inadequado para atender os requisitos de tempo real. Linux$^{\mathbf{Prt}}$ e Linux$^{\mathbf{Xen}}$, por outro lado, apresentam valores de latências dentro dos padrões esperados. Vale a pena notar o comportamento destes sistemas com carga. Apesar do valor médio encontrado para Linux$^{\mathbf{Xen}}$ ($8,7\mu s$) ser superior ao do Linux$^{\mathbf{Prt}}$ ($3,8\mu s$), o desvio padrão é significativamente menor em favor de Linux$^{\mathbf{Xen}}$, característica desejável nos sistemas de tempo real críticos. De fato, para tais sistemas, deseja-se que o pior caso seja próximo do caso médio.

É interessante ainda comparar o comportamento de Linux$^{\mathbf{Prt}}$ sem utilizar o contexto de *threads* de interrupção, isto é, com a opção `IRQF_NODELAY`, comentada anteriormente. Como pode ser observado na Figura 5, apesar das latências de ativação sem estresse apresentar bons resultados em comparação ao Linux$^{\mathbf{Prt}}$, seus valores com estresse indicam um comportamento menos previsível que o Linux$^{\mathbf{Xen}}$.

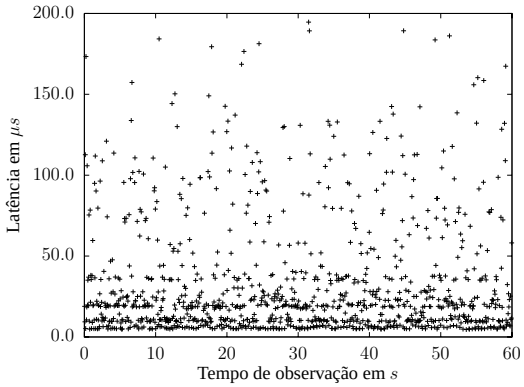
7. Trabalhos relacionados

Alguns resultados experimentais comparando Linux$^{\mathbf{Prt}}$ com Linux$^{\mathbf{Std}}$ são apresentados por [Rostedt and Hart 2007]. Duas métricas são usadas, latências de interrupção e de escalonamento, relacionadas ao escalonamento de uma tarefa periódica. No entanto, os experimentos foram realizados sem sobrecarga do processador e a metodologia usada não foi precisamente descrita. [Siro et al. 2007] realizam um estudo comparativo do Linux$^{\mathbf{Prt}}$, de RT-Linux [V. Yodaiken et al. 2008] e de Linux$^{\mathbf{RTAI}}$ [P. Mantegazza et al. 2008] no qual eles utilizam conjuntamente o *benchmark* LMbench [McVoy and Staelin 1996] e medidas de desvios na execução de uma tarefa periódica. Nestes experimentos, os autores aplicaram uma sobrecarga "média" do processador, sem considerar carga de interrupção. Num outro trabalho, divulgado apenas na Internet [Benoit and Yaghmour 2005], os desenvolvedores do projeto Adeos apresentam resultados comparativos relativos ao Linux com os *patches* Preempt-RT e Adeos. Esta avaliação, bastante abrangente, utiliza o *benchmark* lmbench [McVoy and Staelin 1996] para caracterizar o desempenho das duas plataformas e apresenta resultados de medidas de latências de interrupção realizadas com a porta paralela.

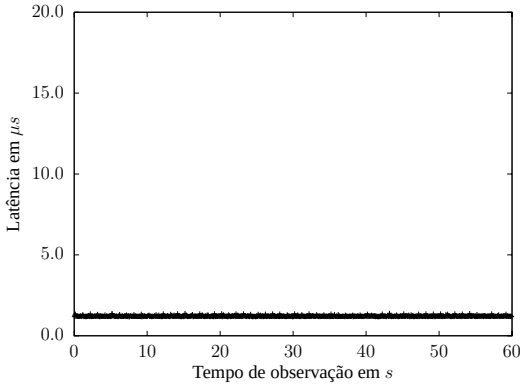
O presente trabalho apresenta resultados de latência de interrupção que confirma os resultados obtidos em [Benoit and Yaghmour 2005] para a plataforma Linux$^{\mathbf{Xen}}$. Já



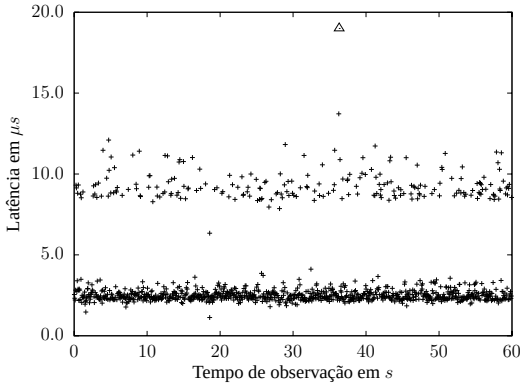
(a) **Linux$^{\mathbf{Std}}$ - Sem carga**
VM: 4.6, DP: 0.4, Min: 4.4, Max: 16.2



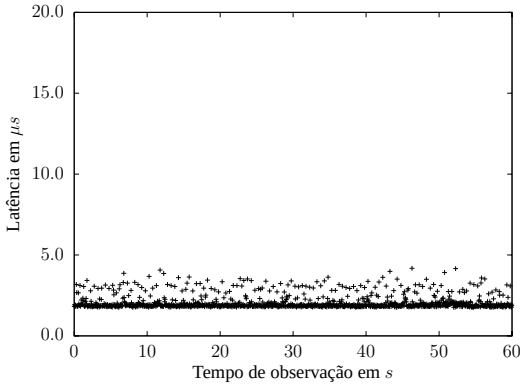
(b) **Linux$^{\mathbf{Std}}$ - Com carga**
VM: 37.3, DP: 48.2,
Min: 4.6, Max: 617.5



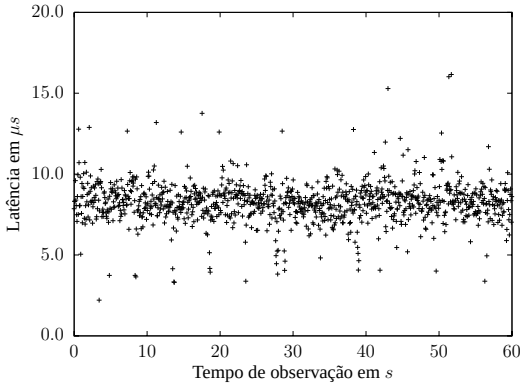
(c) **Linux$^{\mathbf{Prt}}$ - Sem carga**
VM: 2.1, DP: 0.2, Min: 1.2, Max: 9.4



(d) **Linux$^{\mathbf{Prt}}$ - Com carga**
VM: 3.8, DP: 2.8, Min: 1.1, Max: 27.4

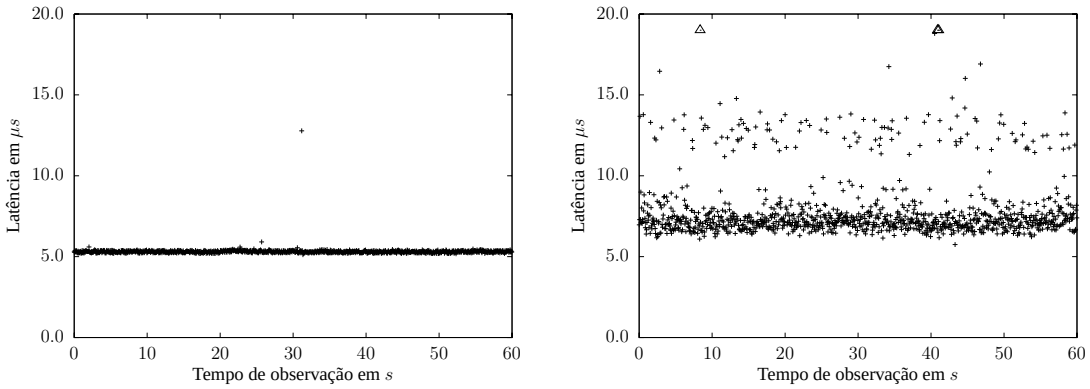


(e) **Linux$^{\mathbf{Xen}}$ - Sem carga**
VM: 2.1, DP: 0.5, Min: 1.8, Max: 8.4



(f) **Linux$^{\mathbf{Xen}}$ - Com carga**
VM: 8.7, DP: 0.3, Min: 1.8, Max: 18.7

Figura 4: Latência de ativação com freqüência de escrita na PP de $20Hz$.



(a) **Linux$^{\mathbf{Prt}}$ - Sem carga**
VM: 5.3, DP: 0.3, Min: 5.0, Max: 13.1

(b) **Linux$^{\mathbf{Prt}}$ - Com carga**
VM: 8.0, DP: 2.0, Min: 5.2, Max: 31.0

Figura 5: Latência de ativação do Linux$^{\mathbf{Prt}}$ desabilitando o *thread* associado as interrupções da PP (opção `IRQF_NODELAY`).

os resultados encontrados aqui para Linux$^{\mathbf{Prt}}$, sem a opção `IRQF_NODELAY`, diferiram dos apresentados por [Benoit and Yaghmour 2005], pois uma degradação das garantias temporais por esta plataforma foi observada, tal como visto na Seção 6.2.1. Em relação às latências de ativação, não temos conhecimento de nenhum outro trabalho comparativo. Experimentos idênticos aos relatados aqui foram conduzidos para a plataforma Linux$^{\mathbf{RTAI}}$ [Regnier 2008] e os resultados encontrados são semelhantes aos apresentados para Linux$^{\mathbf{Xen}}$, dado que ambas plataformas utilizam o mesmo *nanokernel*.

8. Conclusão

Neste trabalho, a avaliação de duas soluções de SOTR baseadas em Linux foi realizada. A metodologia experimental permitiu medir as latências de interrupção e de ativação, em situações de carga variável, tanto do processador quanto de eventos externos tratados por interrupção. Enquanto o Linux padrão apresentou latências no pior caso acima de $100\mu s$, as plataformas Linux$^{\mathbf{Prt}}$ e Linux$^{\mathbf{Xen}}$ conseguiram prover garantias temporais com uma precisão abaixo de $20\mu s$. No entanto, para se conseguir este comportamento em relação ao Linux$^{\mathbf{Prt}}$, foi necessário desabilitar *threads* de interrupção, tornando o sistema menos flexível. Com tais *threads*, o comportamento de Linux$^{\mathbf{Prt}}$ sofre considerável degradação da sua previsibilidade temporal. A plataforma Linux$^{\mathbf{Xen}}$ se mostrou mais adequada, pois tanto oferece um ambiente de programação em modo usuário, quanto consegue previsibilidade temporal característica de sistema de tempo real.

Referências

Abeni, L., Goel, A., Krasic, C., Snow, J., and Walpole, J. (2002). A measurement-based analysis of the real-time performance of the linux kernel. In *Proc. of the Real-Time Technology and Applications Symposium (RTAS02)*, pages 1–4.

Benoit, K. and Yaghmour, K. (2005). Preempt-RT and I-pipe: the numbers. http://marc.info/?l=linux-kernel&m=112086443319815&w=2. Last access 03/08.

Calandrino, J., Leontyev, H., Block, A., Devi, U., and Anderson, J. (2006). Litmus^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers. In *In Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–126.

D. P. Bovet, M. C. (2005). *Understanding the Linux Kernel*. O'Reilly, 3rd edition.

Dozio, L. and Mantegazza, P. (2003). Linux real time application interface (RTAI) in low cost high performance motion control. In *Proceedings of the conference of ANIPLA, Associazione Nazionale Italiana per l'Automazione*.

Fry, G. and West, R. (2007). On the integration of real-time asynchronous event handling mechanisms with existing operating system services. In *Proceedings of the International Conference on Embedded Systems and Applications (ESA'07)*.

I. Molnar et al. (2008). PreemptRT. http://rt.wiki.kernel.org - Last access jan. 08.

L. Torvalds et al. (2008). Kernel. http://www.kernel.org - Last access jan. 08.

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice-Hall.

Marchesotti, M., Migliardi, M., and Podestà, R. (2006). A measurement-based analysis of the responsiveness of the Linux kernel. In *Proc. of the 13th Int. Symposium and Workshop on Engineering of Computer Based Systems*, volume 0, pages 397–408. IEEE Computer Society.

McKenney, P. (2005). A realtime preemption overview. http://lwn.net/Articles/146861/ - Last access dez. 07.

McVoy, L. W. and Staelin, C. (1996). lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, pages 279–294.

P. Gerum et al. (2008). Xenomai. http://www.xenomai.org - Last access jan. 08.

P. Mantegazza et al. (2008). RTAI. http://www.rtai.org - Last access jan. 08.

Pratt, S. L. and Heger, D. A. (2004). Workload dependent performance evaluation of the linux 2.6 i/o schedulers. In *Proceedings of the Linux Symposium*, pages 425–448.

Regnier, P. (2008). Especificação formal, verificação e implementação de um protocolo de comunicação determinista, baseado em ethernet. Master's thesis, Universidade Federal da Bahia.

Rostedt, S. and Hart, D. V. (2007). Internals of the rt patch. In *Proceedings of the Linux Symposium*, pages 161–172.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority Inheritance Protocols: An approach to real-time synchronisation. *IEEE Transaction on Computers*, 39(9):1175–1185.

Siro, A., Emde, C., and McGuire, N. (2007). Assessment of the realtime preemption patches (RT-Preempt) and their impact on the general purpose performance of the system. In *Proceedings of the 9th Real-Time Linux Workshop*.

Stodolsky, D., Chen, J., and Bershad, B. (1993). Fast interrupt priority management in operating systems. In *Proc. of the USENIX Symposium on Microkernels and Other Kernel Architectures*, pages 105–110.

V. Yodaiken et al. (2008). RT-Linux. http://www.rtlinuxfree.com - Last access jan. 08.

Yaghmour, K. (2001). The real-time application interface. In *Proceedings of the Linux Symposium*.