Gerência de tempo no Sistema Operacional EPOS

Giovani Gracioli, Danillo Moura Santos, Roberto de Matos, Lucas Francisco Wanner, Antônio Augusto Fröhlich

[1]Laboratório de Integração Software e Hardware (LISHA)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476, 88049-900, Florianópolis, SC, Brasil

{giovani,danillo,roberto,lucas,guto}@lisha.ufsc.br

Resumo. *Uma das tarefas de um sistema operacional é o tratamento de eventos de tempo. Tradicionalmente a gerência de tempo é baseada em interrupções periódicas de um dos relógios de hardware do sistema (ticks). Porém, esta abordagem apresenta limitações, como falta de precisão, maior custo computacional e maior consumo de energia. Isso motivou a procura por novas soluções, e o emprego de técnicas de temporizadores não periódicos (ex.: one-shot timers) tornou-se freqüente, principalmente em sistemas operacionais de propósito específico, que possuem algum tipo de restrição de tempo, energia ou processamento (ex.: sistemas embarcados, de tempo real e multimídia). Este trabalho faz uma comparação entre as implementações de temporizadores de disparo único e temporizadores periódicos na gerência de tempo do sistema operacional EPOS. São apresentados os impactos no tamanho de código do sistema (Footprint), número de trocas de contextos, número de execuções do tratador de interrupção e tempo de computação em diferentes cenários de execução.*

Abstract. *One of the tasks of an operating system is to handle time events. Traditionally, time management is based on periodical interrupts from one of the system's hardware timers (ticks). However, this approach presents limitations, such as lack of precision, large overhead, and large power consumption. These limitations have motivated the use of non-periodical timers (e.g. one-shot timers), specially in specific-purpose operating systems with timing restrictions, such as embedded, real-time, and multimedia systems. This work presents a comparison between single-shot and periodical timer implementations in the time management abstractions in the EPOS operating system. We present the impact of the different implementation in terms of memory footprint, number of context switches, number of interrupt handler executions and run time in different execution scenarios.*

1. Introdução

Tradicionalmente os sistemas operacionais de propósito geral implementam temporizações baseadas em interrupções periódicas de um dos relógios de hardware do sistema (*ticks*). Com a proliferação dos sistemas computacionais e o emprego desses nas mais diversas aplicações, foram revelados problemas nessas abordagens que utilizam gerenciamento de tempo periódico [Tsafrir et al. 2005, Farines et al. 2000, Aron and Druschel 2000], entre eles: a falta de precisão, maior custo computacional e

maior consumo de energia. Isso motivou a procura por novas soluções e o emprego da técnica de temporizadores não periódicos, como por exemplo, os temporizadores de disparo único (*one-shot timers*). Estas técnicas passaram a ser empregadas em sistemas operacionais de propósito específico, tais como sistemas embarcados, de tempo real e multimídia, os quais possuem algum tipo de restrição de tempo, energia ou processamento.

Os temporizadores periódicos são implementados baseados em interrupções de um relógio de hardware (*ticks*). A cada *tick*, o kernel executa tarefas administrativas como verificação de alarmes a serem disparados, contagem de tempo da tarefa que está executando no processador, análise de escalonamento, preempções e outros serviços periódicos do sistema. Esse tipo de abordagem revela vários problemas [Tsafrir et al. 2005]: desperdício de energia no caso de sistemas embarcados e móveis, problemas de segurança, tarefas *soft real-time* e multimídia ficam sujeitas a precisão limitada do relógio do sistema.

Além desses problemas, a implementação de temporizadores periódicos pode gerar erros grosseiros nos serviços de temporização providos pelo SO às tarefas [Farines et al. 2000]. Por exemplo, no sistema operacional Linux, a resolução do temporizador é de 10ms. Suponha que uma aplicação solicite uma temporização de 15ms logo após o retorno de interrupção do temporizador. Nesse cenário, a tarefa deve esperar até a próxima interrupção, onde o sistema começa a contagem do tempo e ainda esperar 20ms, duas interrupções de 10ms. Assim, uma tarefa que solicitou uma temporização de 15ms deverá esperar 30ms para ter sua temporização atendida, o que para muitas aplicações é inaceitável.

Uma das alternativas para solucionar esses problemas é a utilização da gerência de tempo não periódica, tendo como abordagem mais comum, os temporizadores de disparo único (*one-shot timers*), ou seja, a interrupção do temporizador acontecerá somente no momento para o qual foi agendado. Esses temporizadores são diferentes dos temporizadores periódicos, onde um temporizador é disparado com uma freqüência predefinida e a lógica de controle de tempo é feita em software. Temporizadores periódicos possuem um período, tempo levado para um disparo, que é a unidade mínima de temporização do sistema que o utiliza. Por sua vez, temporizadores de disparo único são programados em hardware, por isso a resolução máxima de tempo suportada é a resolução do próprio temporizador em hardware.

Este trabalho faz uma análise de duas implementações de gerência de tempo no sistema operacional EPOS, uma baseada em temporizadores de disparo único e a outra baseada em temporizadores periódicos. São apresentados os impactos no tamanho de código do sistema *(Footprint)*, número de trocas de contextos, número de execuções do tratador de interrupção, e no tempo de computação das *Threads*. Esta análise visa discutir os pontos positivos e negativos da implementação baseada em temporizadores de disparo único em relação à implementação baseada em temporizadores periódicos.

Esse artigo está organizado da seguinte forma: na seção 2 são apresentados os trabalhos relacionados e as diferentes abordagens na gerência de tempo em um SO encontradas na literatura; na seção 3 é descrita a gerência de tempo do sistema operacional EPOS já existente e a abordagem baseada em (*one-shot timers*); a apresentação dos resultados e sua análise são feitas na seção 4 e as considerações finais aparecem na seção 5.

2. Trabalhos Relacionados

Em [Tsafrir et al. 2005] é apresentada uma análise dos problemas da falta de precisão, da assincronicidade e consumo de energia na utilização dos temporizadores periódicos e propõe uma solução baseada em uma abordagem chamada *Smart Timers*. Os *Smart Timers* são definidos com três características básicas: 1) Temporização precisa com limite configurável da latência máxima; 2) *Overhead* reduzido agregando eventos próximos e 3) *Overhead* reduzido evitando a sobrecarga desnecessária de eventos periódicos.

Kohout [Kohout et al. 2003] apresenta estratégias utilizadas para suportar de maneira eficiente sistemas operacionais de tempo real com alguns componentes implementados em hardware. O intuito é diminuir o impacto causado pelo sistema operacional de tempo real na aplicação. Este impacto é medido em termos de tempo de resposta e utilização da CPU. Neste trabalho é proposto um gerenciador de tarefas Tempo Real (do inglês *Real Time Task Manager - RTM*), que é uma espécie de banco de dados de tarefas em hardware, sendo implementado como um periférico no mesmo chip do processador (desde que o processador esteja em uma FPGA). O RTM suporta funções de núcleo, como no caso do gerenciamento de tempo, causando uma redução de até 10% (com 24 tarefas) na utilização do processador para tratamento de *ticks* do *timer* do sistema.

Soft Timer [Aron and Druschel 2000] é uma implementação de temporizadores que não é vinculada somente as interrupções do relógio de hardware. Essa abordagem aproveita o retorno de chamadas do sistema para verificar se existe alguma tarefa pendente e assim liberá-la. Característica que possibilita a diminuição do número das trocas de contexto, do número e do *overhead* das interrupções do relógio físico, já que as chamadas ao sistema acontecem a uma taxa muito maior que as interrupções do relógio. Por outro lado, como a freqüência exata das chamadas ao sistema é imprevisível, não existe uma garantia de precisão e a estrutura de temporizadores periódicos tradicionais são utilizados para certificar o funcionamento mínimo do sistema.

Firm Timer [Goel et al. 2002] combina três abordagens diferentes: temporizadores de disparo único, *soft timers*, [Aron and Druschel 2000] e temporizadores periódicos para prover um mecanismo eficiente de alta resolução e baixo *overhead*. Essa combinação permite reduzir a necessidade de interrupções, atenuando o risco de *overheads* excessivos.

A base da maioria desses trabalhos é evitar as intervenções desnecessárias do sistema operacional causadas pelo tratamento de interrupções do temporizador, onde na maior parte do tempo a única ação é incrementar o contador de *ticks* do sistema. Com a abordagem dos temporizadores de disparo único a interrupção acontecerá somente no momento para o qual foi agendado, evitando o *overhead* do tratamento de interrupções periódicas. Mesmo com o tratamento dessa interrupção de agendamento sendo mais complexo, devido a necessidade de re-programação do temporizador para a próxima tarefa da fila, na maioria dos casos, onde o intervalo entre acordar as tarefas é maior que o de um *tick* do sistema, os temporizadores de disparo único possuem melhor desempenho. Nas próximas seções serão apresentadas as duas implementações da gerência de tempo (periódica e *one-shot timer*) no sistema operacional EPOS e os impactos causados no sistema. Na próxima seção são apresentados os resultados comparativos da nova implementação.

3. Gerência de Tempo no EPOS

O EPOS *(Embedded Parallel Operating System)* é um sistema operacional para sistemas embarcados desenvolvido seguindo a AOSD [Fröhlich 2001] *(Application-Oriented System Design)*, que faz uso da Engenharia de Domínio para definir componentes que representam entidade significantes de um domínio. O sistema operacional EPOS ainda combina conceitos de FBD *(Family-Based Design)*, AOP *(Aspect-Oriented Programming)*, Orientação a Objetos e Meta-Programação Estática na qual permite a organização de famílias de componentes independentes de cenário. Através de um metaprograma que utiliza regras de composição, o EPOS possui um *framework* de componentes que permite a geração de sistemas adaptados à aplicação. As abstrações do sistema são adaptadas aos cenários de execução utilizando-se técnicas de orientação a aspectos, aplicadas com Adaptadores de cenários [Fröhlich and Schröder-Preikschat 2000]. Regras coordenam as operações realizadas pelo metaprograma, especificando restrições e dependências para a composição das abstrações do sistema. Cada aspecto pode ser aplicado individualmente a cada abstração do sistema, por exemplo, a aplicação do aspecto concorrência em um componente faz com que este tenha seus métodos de acesso guardados por um Mutex ou Semáforo. O uso de metaprogramação para a composição das abstrações do sistema não adiciona *overhead* em tempo de execução.

O uso de mediadores de hardware [Polpeta and Fröhlich 2004] ainda permitem que o mesmo sistema suporte arquiteturas distintas (ex. H8, AVR, POWERPC, SPARCV8, IA32) mantendo uma mesma interface com a aplicação. Basicamente os mediadores de hardware são construções que encapsulam dependências arquiteturais em sistemas projetados seguindo AOSD, dando aos componentes de hardware como CPU, FPU, barramentos entre outros, interfaces comuns de Sistema Operacional. Por exemplo, no caso da gerência de tempo, os temporizadores em hardware apresentam diversas funções distintas, e podem ser configurados de diversas formas diferentes. Um temporizador pode atuar como um modulador de largura de pulsos controlando um circuito analógico, como *Watchdog*, como temporizador de intervalo programável ou um simples temporizador de intervalo fixo. Cada um desses possíveis tipos de temporizadores possuem as suas peculiaridades de configuração.

No nível mais alto do sistema EPOS o tempo é manipulado pela família de abstrações denominada ***Timepiece***, composta pelas abstrações ***Clock***, ***Alarm*** e ***Chronometer***. Cada abstração possui uma função específica no sistema, a abstração ***Clock*** é responsável por armazenar o tempo corrente e está disponível apenas em sistemas que possuem dispositivos de relógios de tempo-real (RTC); a abstração ***Chronometer*** é utilizada para realizar medições de tempo com alta precisão; e a abstração ***Alarm*** pode ser utilizada para gerar eventos, para acordar uma Tarefa/*Thread* ou chamar uma função. Essa família é suportada pelos mediadores de hardware ***Timer***, ***TimeStamp Counters (TSC)*** e ***Real-Time Clocks***. [Marcondes et al. 2006]

A periodicidade do sistema é caracterizada na abstração ***Alarm***, que por sua vez, é suportada pelo mediador ***Timer*** que configura o temporizador de hardware para gerar interrupções a uma freqüência determinada. No atendimento dessas interrupções são contabilizados os *ticks* e executada uma rotina para verificação e liberação de eventos agendados. Além do *overhead* causado pelo atendimento periódico das interrupções, se um evento for agendado para um tempo menor ou não múltiplo que o período do sistema,

a invocação da tarefa agendada pode ser atrasada por erros de arredondamento inerentes a granularidade dos *ticks*. A Figura (1) apresenta o diagrama de classes das duas principais abstrações que compõe o gerenciamento de tempo do sistema EPOS para a arquitetura *AVR*. A Abstração **Alarm** utiliza dois *Timers* em Hardware disponíveis na arquitetura AVR. Um destes *Timers*, o **ATMega128_Timer_3**, é utilizado para controlar a fila de requisições do **Alarm**, este é um *Timer* de 16 bits, o que permite o agendamento de Alarmes de até aproximadamente 9 segundos com o Clock do sistema em $\sim$ 7.2MHz e *PRESCALE* de 1024 no Clock do *Timer*. O outro *Timer*, **ATMega128_Timer_1**, gera interrupções periódicas que acionam o escalonador do sistema, estas interrupções acontecem somente no momento em que o escalonador deve executar e o tempo destas é configurado baseado no *QUANTUM* do sistema.

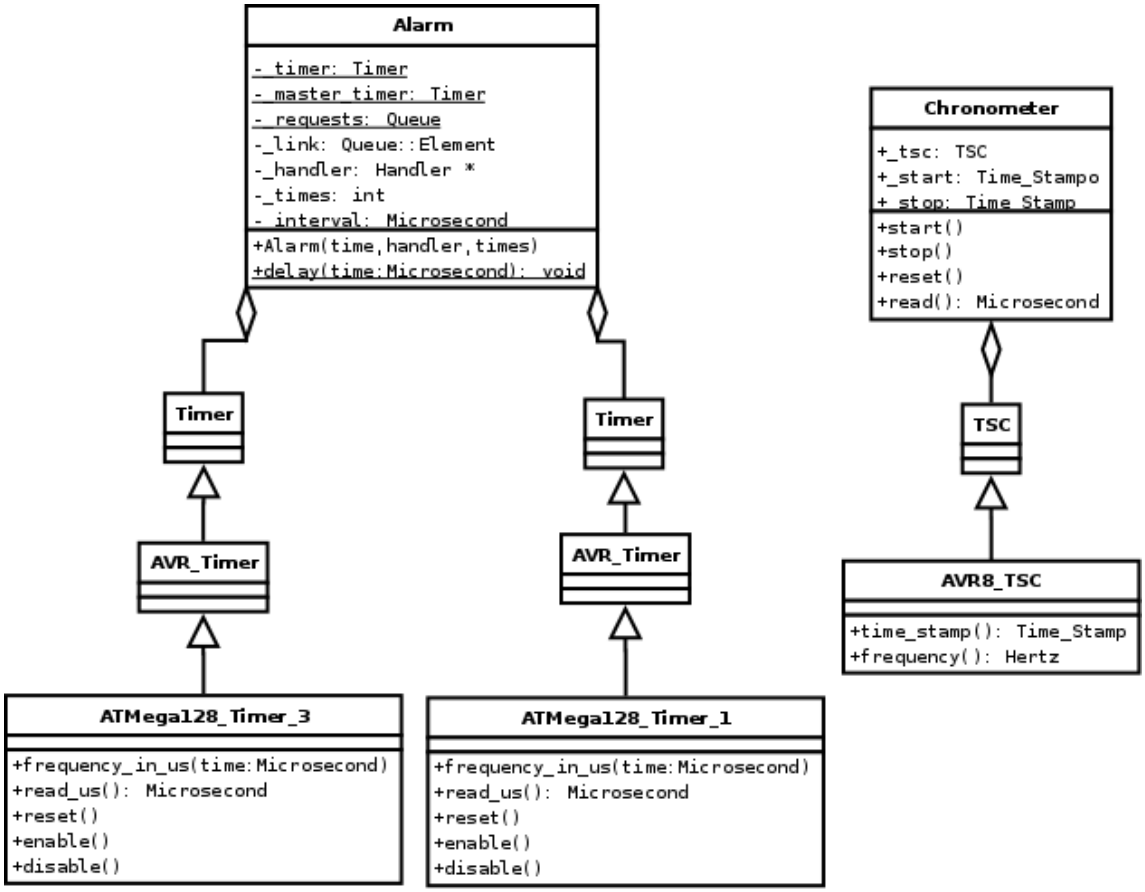


Figura 1. Diagrama de classes de Abstrações e Mediadores de Hardware (Timers) do sistema EPOS com disparo único

O tratamento de uma interrupção no gerenciamento de tempo periódico, além de contabilizar os *ticks* e liberar eventos agendados, também é responsável por chamar o tratamento de tempo do escalonador (*master_handler*). Esta rotina decrementa o contador de *ticks* e verifica se este contador é menor ou igual a zero, chamando a rotina de escalonamento de threads em caso positivo.

As modificações para eliminar as interrupções periódicas programadas no temporizador (contagem dos *ticks*) e para criar a estrutura de temporização baseada na técnica de disparo único se concentraram na abstração **Alarm**. Com a mudança do conceito de

interrupção periódica para intervalo variável do próximo disparo, o tratador de interrupções do ***Alarm*** foi implementado como mostra o diagrama de sequência na Figura 2. Diferente da abordagem periódica, onde a cada *tick* o tratador da interrupção chamava o *master_handler*, a nova implementação utiliza dois temporizadores em hardware independentes. Um relativo às requisições dos alarmes (fila ***_requests***), ou seja, é programado para disparar no exato momento em que foi agendado para acordar uma tarefa e outro ligado ao escalonamento do sistema, programado para disparar a cada *QUANTUM* de escalonamento, como supracitado.

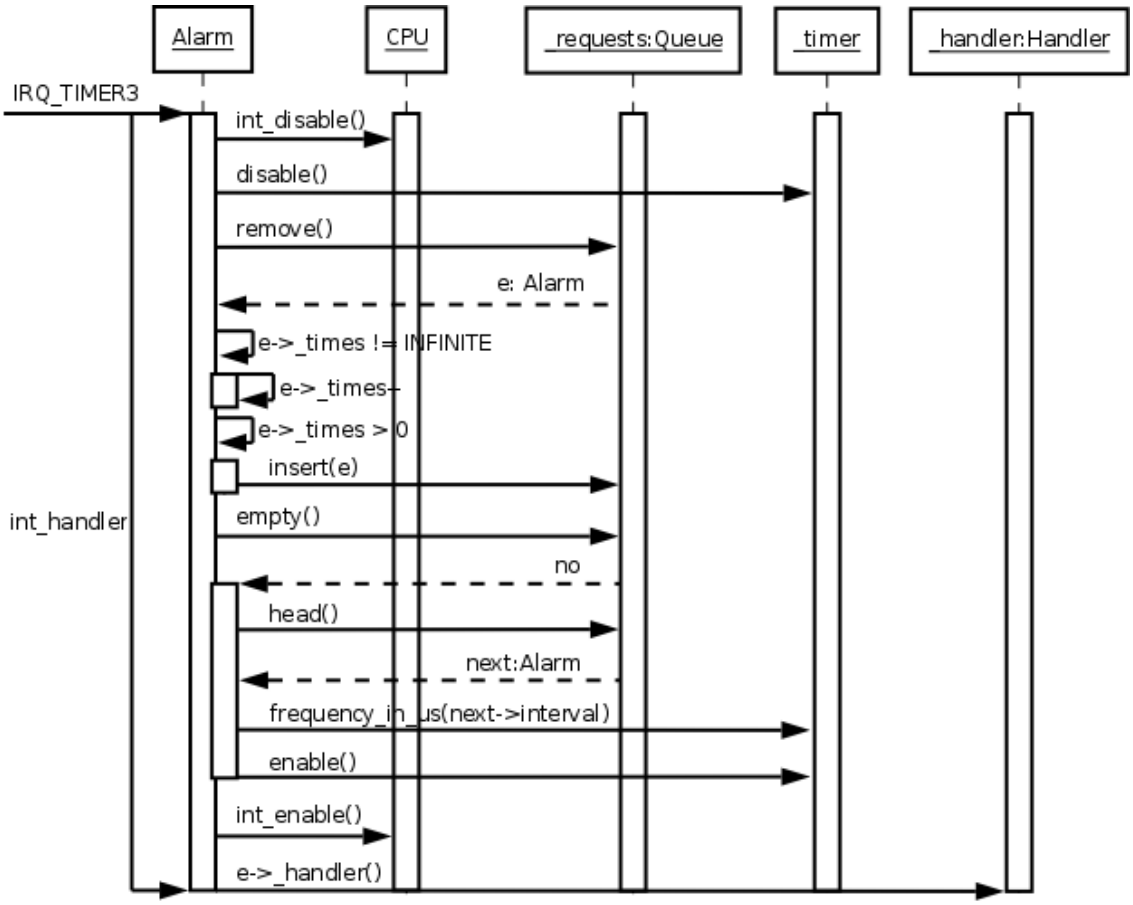


Figura 2. Diagrama de seqüência do tratamento da interrupção do temporizador de disparo único no EPOS

Com essa modificação, o temporizador relacionado ao escalonamento é configurado na inicialização do sistema e ao ser disparado simplesmente chama a rotina de escalonamento. Isto deixa o tratamento da interrupção simples e rápido, pois não necessita fazer o controle do número de *ticks* a cada interrupção e também diminui a influência do escalonador no sistema. Já o temporizador ligado aos alarmes só é inicializado na criação de um **Alarm**. Ao ser criado, o **Alarm** é inserido em uma lista ordenada e relativa, de forma que o valor inserido seja exatamente o valor necessário para a programação do temporizador. Quando chega uma interrupção, o tratador retira o **Alarm** que está na primeira posição da fila, verifica se ainda existem **Alarms** na fila e se caso exista, reprograma o temporizador com o valor do próximo. Este tratador de interrupção é executado muito menos vezes que o tratador periódico, pois só é chamado quando existe um evento de

tempo a ser gerenciado. Podemos ver os impactos nestes números na próxima seção.

4. Análise dos Resultados

Com o intuito de avaliar o impacto da implementação da abordagem de temporizadores de disparo único no sistema operacional EPOS foi utilizada uma aplicação que é uma implementação do problema Jantar dos Filósofos [Tanenbaum 2001], na qual 5 **Threads** (filósofos) alternam dois comportamentos: pensar e comer. Para comer, o filósofo necessita de dois garfos (pois o prato é macarrão) e existem na mesa 5 garfos, portanto, dois filósofos não adjascentes podem comer ao mesmo tempo. Nesta implementação, um filósofo "pensa" (faz um pedido de delay ao *Alarm* de tempo variável e ao acordar, adquiri dois garfos (semáforo p()) e "janta". Quando o filósofo reassume o controle, os semáforos são liberados ((v()). Cada thread é executada 10 vezes.

A aplicação teste foi implementada utilizando um microcontrolador Atmel ATMega128 de 8 bits com Clock de aproximadamente 7.2 MHz, 4 KB de memória RAM, 128 KB de memória flash de programa, 2 temporizadores de 8 bits e 1 temporizador de 16 bits. Como já explicado no quarto parágrafo da seção 3 um temporizador de 8 bits foi utilizado para o disparo do escalonador do sistema e o outro de 16 bits para gerenciamento da fila do **Alarm**. Nesta análise, 4 características foram avaliadas:

1. *Consumo de Memória*: sistemas embarcados são caracterizados pela falta de recursos de processamento, memória e energia. Um sistema operacional para um sistema embarcado deve economizar o máximo de recursos possíveis. Neste contexto, esta métrica avalia e compara o consumo de memória utilizado pela implementação da aplicação teste usando uma versão do EPOS com temporização de disparo único e outra com temporização periódica.
2. *Tempo de Computação*: o tempo de computação de uma thread é o tempo transcorrido entre o início e o término da sua execução. O tempo de computação pode ser influenciado pela interferência sofrida por tarefas de maior prioridade no sistema, ou seja, a tarefa que está em execução é preemptada e a mais prioritária ganha o direito de ser executada. Além disso, na solução de temporização periódica, a tarefa que está em execução também sofre uma interferência a cada interrupção do temporizador, pois na implementação existente o tratador de interrupção tem maior prioridade. Na temporização não-periódica, o tratador da interrupção só será executado quando uma tarefa, como a liberação de alarmes, precisar ser realizada.
3. *Número de Trocas de Contexto*: uma troca ou chaveamento de contexto se refere à troca de uma tarefa/**Thread** que está sendo processada por outra, podendo a primeira não ter sido concluída. É garantido que quando o contexto anterior armazenado seja restaurado, o ponto de execução (contexto de software) volte ao mesmo estado anterior.
4. *Número de Execuções do Tratador de Interrupções*: esta métrica visa comprovar o real ganho de processamento através do menor número de execuções do tratador de interrupção do temporizador. Usando um temporizador de disparo único, o número de execuções do tratador tende a diminuir, pois ele somente irá ser chamado quando alguma ação na gerência de tempo deve ser realmente efetuada.

A Tabela 1 apresenta o consumo de memória da aplicação teste das duas versões do EPOS, com temporizador de disparo único e periódico. A versão de disparo único

Tabela 1. Consumo de memória da aplicação teste com e sem o uso do temporizador de disparo único no EPOS.

Temporização Periódica (bytes)	*Disparo Único (bytes)*	*Diferença (bytes)*
32000	**32504**	**504**

consumiu 504 bytes a mais do que a versão periódica. Isso ocorreu devido ao uso de 2 temporizadores, um relacionado ao escalonador e outro para a gerência de eventos.

Os testes a seguir foram realizados em três cenários. Estes cenários são descritos abaixo, mostrando o tempo que cada Filósofo (*Thread*) passa pensando e depois comendo a cada execução:

- Cenário 1: Pensa por 1000000 μs e come por 5000000 μs
- Cenário 2: Pensa por 100000 μs e come por 500000 μs
- Cenário 3: Pensa por 25000 μs e come por 125000 μs

Estes cenários foram criados para avaliar o impacto do uso de temporizadores de disparo único em aplicações que passam a maior parte do tempo aguardando eventos do *timer*, como será mostrado nas tabelas e gráficos a seguir.

Tabela 2. Número de execuções do tratador de interrupção. Temporização periódica X Disparo Único.

Cenário	*Tratador*	*Temporização Periódica*	*Disparo Único*	*Diferença*
Cenário 1	int_handler	**26005**	**100**	**25905**
Cenário 1	*master_handler*	**26415**	**3559**	**22856**
Cenário 2	int_handler	**9843**	**100**	**9743**
Cenário 2	*master_handler*	**10252**	**320**	**9932**
Cenário 3	int_handler	**3079**	**100**	**2979**
Cenário 3	*master_handler*	**3239**	**68**	**3171**

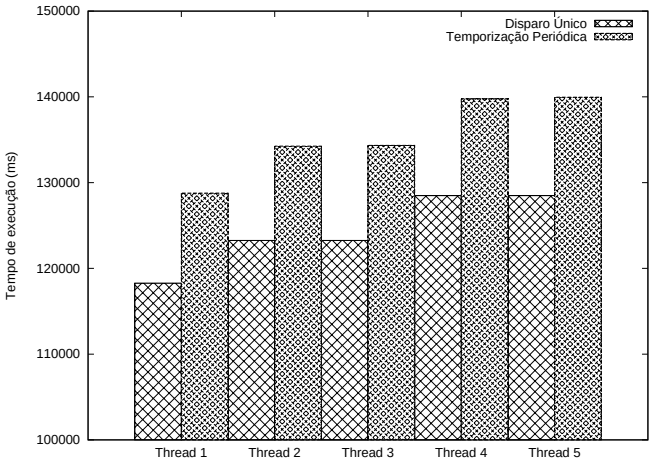
A Tabela 2 apresenta o número de execuções do tratador de interrupção do temporizador. Obviamente, o tratador de interrupção na temporização periódica executa de *Tick* em *Tick* ($\sim$ 1,38 ms), mesmo que nenhum evento esteja pronto para ser atendido. Diferentemente, o tratador do temporizador de disparo único somente é executado quando algum evento deve ser gerenciado, obtendo até 25905 execuções a menos(Cenário 1). A cada *tick*, o tratador periódico chama o tratador do *master_handler* (escalonador), para que este seja executado sempre, mesmo que outro alarme seja disparado no mesmo *tick*, sendo que na versão com temporizador de disparo único o *master_handler* somente é acionado quando o *QUANTUM* é alcançado.

Tabela 3. Número de trocas de contexto entre as threads para os 3 cenários.

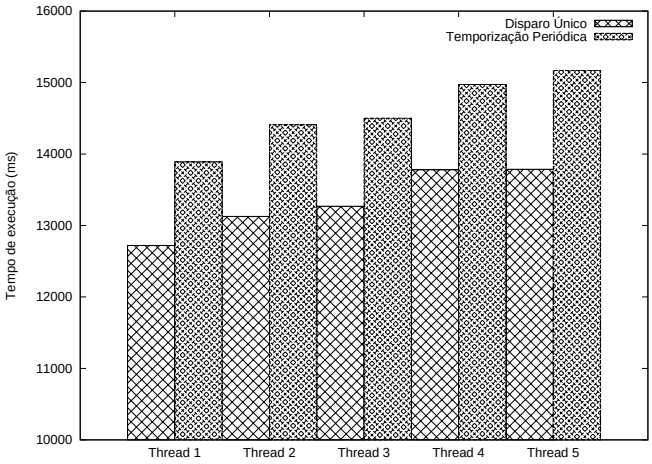
Cenário	*Temporização Periódica*	*Disparo Único*	*Diferença*
Cenário 1	**616**	**344**	**272**
Cenário 2	**615**	**360**	**255**
Cenário 3	**617**	**317**	**300**

Na Tabela 3 é mostrado o número de trocas de contexto das duas versões do EPOS com a aplicação teste. E como esperado a versão com temporizador de disparo único ob-

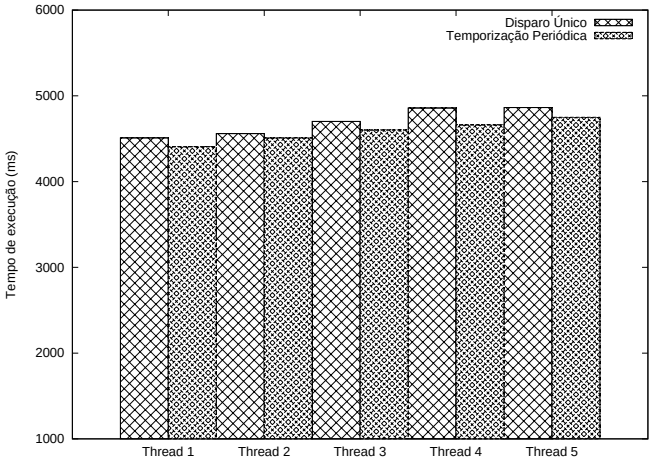
teve até 300 trocas de contexto a menos(Cenário 3), que é um reflexo do número de execuções do tratador de interrupção, que acaba diminuindo o *QUANTUM* efetivo da *Thread*. Com a diminuição do *overhead* do sistema operacional, o *QUANTUM* de tempo dado as tarefas para execução é melhor aproveitado, sofrendo menos preempções do escalonador.



(a) Cenário 1



(b) Cenário 2



(c) Cenário 3

Figura 3. Tempo de computação das threads nos 3 cenários.

Na Figura 3 é mostrado o tempo de computação de cada uma das *Threads* (Filósofos) nos 3 cenários. Nos cenários 1 e 2 vemos que o tempo de computação médio das Threads é menor na implementação com gerenciador de tempo de disparo único. Já no cenário 3, onde os filósofos (*Threads*) passam tempos menores pensando e comendo, vemos que a implementação baseada em *ticks* impacta menos no tempo de computação. Isso acontece pois os ticks processados pelo SO são muito próximos dos intervalos requisitados pelas *Threads*, fazendo que cada tratador de interrupção, praticamente, dispare um alarme. Isso nos leva a concluir que os tempos de computação serão quase os mesmos nas duas implementações, para aplicações com intervalos pequenos (menores que 10000 μs).

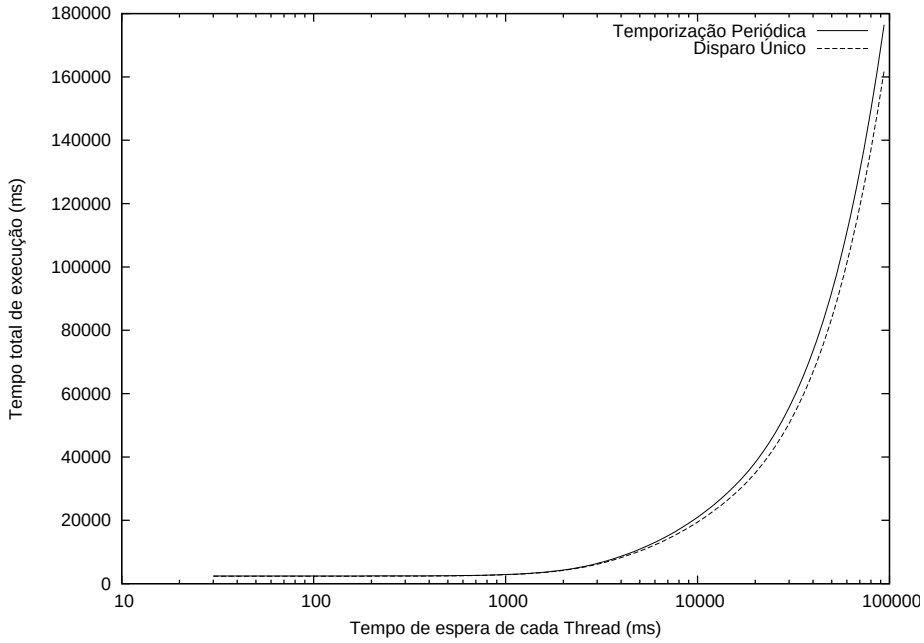


Figura 4. Tempo total de execução variando tempo de espera

Na Figura 4, uma nova série de testes mostra o comportamento do gerenciamento de tempo periódico e baseado em disparo único, quando se varia o tempo total de espera de cada *Thread* (Tempo de pensar e comer de cada filósofo). Esta figura mostra que quando a aplicação passa a maior parte do tempo aguardando eventos de temporizadores, o impacto do sobrecusto no gerenciamento periódico é maior, resultando em maior tempo total de execução. Quando o tempo de espera total é pequeno, há pouca diferença no desempenho das duas alternativas, resultando em tempos de computação bastante parecidos.

5. Conclusões

Este artigo apresentou uma comparação entre as implementações de temporizadores de disparo único e temporizadores periódicos na gerência de tempo do sistema operacional EPOS. Embora a abordagem de temporização de disparo único não seja a mais eficiente dentre os algoritmos de gerenciamento de tempo não-periódicos, esta implementação possibilitou uma primeira visão dos impactos positivos e negativos da classe de temporizadores não periódicos no EPOS.

Quatro métricas foram avaliadas com o intuito de medir os impactos da implementação das abordagens de gerência de tempo periódica e não-periódica no EPOS. O

número de trocas de contexto e o número de execuções do tratador de interrupção obtiveram melhorias significativas no desempenho do sistema. E como esperado, o consumo de memória da implementação não-periódica foi maior, devido ao uso de dois temporizadores. Já o tempo de computação das tarefas sofre menos impacto do gerenciamento de tempo do sistema de disparo único quando a aplicação passa a maior parte do tempo aguardando eventos de timer.

Apesar da melhoria do desempenho do sistema no caso médio, existem situações onde a abordagem não-periódica pode não se tornar uma boa alternativa, por exemplo, quando a freqüência do tratamento de interrupções do temporizador de disparo único tende a freqüência do número de *ticks* de um temporizador periódico. Com isso, o número de chamadas ao tratador é semelhante. A implementação de outros algoritmos de gerenciamento de tempo não-periódicos mais eficientes, como *smart timers*, será alvo de trabalhos futuros.

Referências

Aron, M. and Druschel, P. (2000). Soft timers: efficient microsecond software timer support for network processing. *ACM Trans. Comput. Syst.*, 18(3):197–228.

Farines, J.-M., da Silva Fraga, J., and de Oliveira, R. S. (2000). *Sistemas de Tempo Real*. Escola de Computação: IME-USP, S ao Paulo, SP.

Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin.

Fröhlich, A. A. and Schröder-Preikschat, W. (2000). Scenario Adapters: Efficiently Adapting Components. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, U.S.A.

Goel, A., Abeni, L., Krasic, C., Snow, J., and Walpole, J. (2002). Supporting time-sensitive applications on a commodity os. In *OSDI*.

Kohout, P., Ganesh, B., and Jacob, B. (2003). Hardware support for real-time operating systems. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 45–51, New York, NY, USA. ACM.

Marcondes, H., Hoeller, A., Wanner, L., and Frohlich, A. (20-22 Sept. 2006). Operating systems portability: 8 bits and beyond. *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, pages 124–130.

Polpeta, F. V. and Fröhlich, A. A. (2004). Hardware mediators: A portability artifact for component-based systems. In Yang, L. T., Guo, M., Gao, G. R., and Jha, N. K., editors, *EUC*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280. Springer.

Tanenbaum, A. S. (2001). *Modern Operating Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Tsafrir, D., Etsion, Y., and Feitelson, D. G. (2005). General purpose timing: the failure of periodic timers. Technical Report 2005-6, School of Computer Science and Engineering, the Hebrew University, Jerusalem, Israel.