Escalonamento interativo no kernel Linux

Lucas Correia Villa Real , Felipe Wilhelms Damasio

[1] Universidade do Vale do Rio dos Sinos
Centro de Ciências Exatas e Tecnológicas
São Leopoldo – RS – Brasil

lucasvr@gobolinux.org, felipewd@terra.com.br

Abstract. *The Linux kernel has problems with responsiveness regarding interactive tasks, which are I/O bound processes that have interaction with the user (mainly Desktop). This paper proposes the introduction of heuristics into the Linux process scheduler to deal with interactive tasks. Through the priorization of I/O bound interactive tasks, the responsiveness of the operating system as a whole was highly improved. The achieved improvements are presented through some measurements of interactive applications.*

Resumo. *O kernel Linux apresenta problemas com o tempo de resposta de tarefas interativas, representando as tarefas* I/O bound *que possuem uma interação com o usuário (principalmente Desktop). Este artigo propõe a introdução de heurísticas de tratamento de tarefas interativas no escalonador do kernel Linux. Através da priorização de tarefas I/O bound interativas, o tempo de resposta do sistema operacional como um todo diminui consideravelmente. Os ganhos obtidos são ilustrados através de algumas medições feitas em aplicações interativas.*

1. Introdução

Tradicionalmente, o escalonador do kernel Linux decidia qual processo iria ser executado através de um método de busca linear em uma lista global de processos[Bovet and Cesati, 2002]. Com a introdução do kernel 2.6, essa política foi modificada, e um algoritmo de complexidade constante ($O(1)$) foi adotado, apresentando melhorias substanciais de escalabilidade[Love, 2003b].

Além de escalonar os processos, o novo escalonador introduziu um estimador de interatividade, que busca determinar quais tarefas são interativas e quais são *CPU bound*. Essa determinação é realizada baseado na premissa de que tarefas CPU bound nunca dormem, e usam todo o tempo (fatia de tempo de CPU) oferecido à elas. Por outro lado, tarefas interativas ocasionalmente dormem, liberando a CPU para uma outra tarefa. Tarefas tradicionalmente conhecidas como I/O Bound, ou puramente geradoras de I/O, como editores de texto (que usam seguidamente interrupções de teclado/mouse) ou mesmo aplicações multimídia, como reprodutores de música e vídeo, podem ser consideradas tarefas interativas, já que utilizam grande parte de seu tempo de vida computacional para realizar tarefas designadas diretamente (e com retorno também direto) pelo usuário[Love, 2003a].

Para implementar a política de estimativa de interatividade, o conceito de I/O bound e CPU bound foi utilizado. Todo o processo que passa boa parte de seu tempo computacional fazendo I/O é considerado I/O bound, sendo que os processos que passam a maior parte de sua vida computacional sem serem preemptados (usando toda a sua rajada de CPU) são considerados não-interativos ou CPU bound.

A heurística do estimador de interatividade clássico utilizado no kernel Linux é implementada dando um bônus à tarefas que são constantemente preemptadas por serem I/O bound, e uma penalidade à tarefas CPU bound. Embora na prática seja improvável, caso uma tarefa seja considerada "neutra" (nem possa ser classificada como I/O ou CPU bound) ela não sofre nenhum tipo de alteração em sua política de prioridades.

O padrão do escalonador $O(1)$[Molnar, 2003] é ter até 5 níveis de prioridades de bônus e punições para serem distribuídas aos processos, dependendo do grau de interatividade destes. O valor do bônus (ou da penalidade) corresponde a 25% do total de -20 a 19 do valor de *nice* do processo.

A heurística de determinação do grau de interatividade de um processo usa um *sleep average* **máximo** contra o sleep average de um processo a cada preempção do mesmo. Quanto mais próximo o processo estiver deste máximo, mais interativa ela vai ser considerada e, conseqüentemente, maior será o bônus que irá receber. Ortogonalmente, quanto mais próximo o processo estiver da negação do sleep average máximo, maior será a penalidade recebida[Love, 2003a].

Alguns resultados do estimador clássico de interatividade podem ser vistos a seguir:

USER	NI	PRI	%CPU	STAT	COMMAND
lucasvr	0	15	0.0	S	vim
felipewd	0	18	0.4	S	bash
root	0	25	91.7	R	loop.sh

A coluna `PRI` mostra a prioridade do kernel, que corresponde ao mapeamento do valor padrão do fator nice de prioridade para o valor 20. A menor prioridade possível (do fator nice), 19, é a prioridade 39. A maior prioridade (também do fator nice), -20, é zero. Assim, os menores números da coluna PRI representam as maiores prioridades (para o escalonador). Um editor de texto, `vim`, recebeu o maior nível de bônus em prioridade, devido à sua natureza interativa. Como ele inicialmente tinha um valor nice de zero, ele agora tem a prioridade 15. Similarmente, um programa executando um loop infinito (processo CPU bound), `loop.sh`, recebeu a máxima penalidade do estimador de interatividade, o qual fez com que a prioridade do processo se tornasse 25. A shell bash, que é basicamente interativo mas executa scripts CPU bound em nome do usuário, recebeu um bônus menor e possui prioridade 18.

Processos mais prioritários (aqueles que possuem um valor de PRI menor), são executados antes dos de menor prioridade (que possuem um valor maior de PRI), e ainda recebem uma maior fatia de tempo no(s) processador(es). Isso implica que tarefas interativas estejam geralmente sendo executadas; elas são escalonadas antes e geralmente tem uma fatia de tempo mais do que o suficiente, até fazerem uma operação de I/O. Isso garante que o editor de texto é capaz de responder a uma interrupção de teclado em pouco tempo, diminuindo a latência no **tempo de resposta** mesmo quando o sistema está sob

estresse.

O estimador de interatividade não é aplicado para tarefas de tempo real, que possuem um intervalo de prioridade maior que as tarefas usuais do sistema. Elas também são mantidas em filas separadas, para garantir que seu intervalo de prioridade será sempre respeitado, e para permitir também que se possam usar algoritmos de filas de multinível[Silberschatz et al., 2003] no escalonador $O(1)$. O estimador de interatividade tem como objetivo beneficiar apenas tarefas de usuários desktop, como aplicativos multimídia, editores de texto, entre outros.

2. Problemas com o estimador de interatividade

A (re)estimativa de interatividade é realizada a cada preempção da tarefa, na tentativa de descobrir o seu comportamento à medida em que o processo é executado. O problema é que até que a tarefa atinja um valor de topo para poder ser dita interativa (sleep average máximo) ela ficará, na melhor das hipóteses, com uma prioridade baixa, ou mesmo ainda pode sofrer penalizações por uso de rajadas de CPU[Love, 2003c].

A princípio uma solução poderia ser a diminuíção do sleep average máximo. Caso ela fosse usada, a classificação das tarefas como interativas ou CPU bound seria mais rápida, bem como a determinação do nível de interatividade da mesma. No entanto, a maior parte das aplicações interativas reais comportam-se de maneira dinâmica: apesar de dormirem freqüentemente, elas também usam rajadas de CPU. Essas rajadas rapidamente diminuem o status interativo da tarefa, e ela repentinamente tem sua prioridade reduzida, indo ser escalonada juntamente com as tarefas CPU bound, ou ainda pior, sendo expiradas e tendo que aguardar uma nova chance para serem colocadas na fila de processos prontos para executar.

Além disso, processos CPU bound eventualmente fazem I/O, e nestes casos eles precisam dormir e perdem o uso do processador. Isso faz com que eles tenham seu nível de prioridade aumentado, sendo considerados processos interativos, e acabam assim roubando o lugar de processos que realmente possuem interatividade com o usuário.

O atual estimador de interatividade não prevê estas oscilações no comportamento de aplicações reais de usuários desktop, como clientes de email ou aplicativos de multimídia, o que não permite a classificação de tarefas interativas ou batch de maneira clara e simples para toda a vida computacional da tarefa.

Um exemplo que ilustra este problema são aplicações multimídia. Tomando como exemplo um tocador de mídia (áudio): como é basicamente feito I/O para realizar leitura de dados do disco e para comunicar-se com a placa de som, ele é facilmente identificado como um processo interativo. No entanto, muitas vezes é necessário realizar a decodificação da *stream* via software, o que consome ciclos de CPU[Rubini and Corbet, 2001]. Assim, esse processo pode demorar para ser classificado como interativo, bem como pode oscilar muito entre os níveis de interatividade, sendo que a reprodução do áudio pode sofrer *glitches* (paradas instantâneas na reprodução da stream) por estar concorrendo com processos batch.

Outro exemplo ainda são aplicações que usam o mouse: no caso de alta carga da CPU, o processo gerenciador do mouse, como o servidor X, pode perder prioridade e as

movimentações do cursor podem demorar a serem refletidas no vídeo.

Vale notar que em alguns casos a decodificação de streams pode ser realizada via hardware, não envolvendo software, bem como o cursor do mouse pode ser renderizado diretamente através de um recurso conhecido por cursor de hardware (*hardware cursor*) em placas gráficas. Nestes casos os problemas de interatividade envolvendo estes dispositivos são minimizados, mas assim que uma stream baseada em um formato não implementado no hardware seja reproduzida, será papel do software realizar a decodificação, e o problema tornará a aparecer (no caso das placas de som).

3. Escalonamento dinamicamente interativo

Devido aos problemas de instabilidade da previsão de interatividade de um processo pelo estimador de interatividade atual, se torna necessário um mecanismo de **dinamização** do nível de interatividade, baseado em uma heurística de determinação da prioridade de I/O que está sendo feita por uma determinada tarefa. Desta forma, seria possível resolver os problemas relacionados à má definição de tarefas como interativas ou CPU bound, bem como o problema de oscilação de prioridades de tarefas interativas.

Para tal, nota-se que apenas a visualização da média simples de sleep average de uma dada tarefa contra uma contante máximo de tempo de espera para determinação do nível de interatividade não é ideal, justamente por existirem outros fatores que determinam a classificação de uma tarefa como interativa, pois tarefas podem fazer I/O e não necessariamente possuirem uma `interação` com o usuário.

Com a utilização da monitoração do uso de dispositivos considerados interativos e associação de uma prioridade utilizada pelo estimador de interatividade, pode-se acrescentar o nível de precisão na determinação do fator de interatividade de uma dada tarefa.

O monitoramento dos dispositivos é realizado através da camada VFS (*Virtual File System*), onde a cada abertura de um destes dispositivos, a prioridade do processo relacionado à esta operação é incrementada de acordo com um bônus, como pode ser visto na Tabela 1. Assim que este processo finaliza a utilização do dispositivo, o bônus é removido, e o nível de interatividade da tarefa torna a ser estipulado apenas pelo estimador padrão presente no escalonador $O(1)$.

4. Resultados obtidos

Foram realizados testes de medição de latência de aplicações multimídia (isoladas e integradas no resto do sistema) com um kernel $O(1)$ tradicional e com o kernel com o novo estimador de interatividade, aqui identificado por *devprio*.

Os testes realizados foram a medição do tempo em que uma tarefa permaneceu dormindo a partir do instante em que ela foi preemptada, até ser escolhida pelo escalonador para utilizar o processador novamente[Williams, 2002]. Essa medição foi realizada em tempo de execução da tarefa, ficando os dados relativos às estatísticas disponíveis em entradas no sistema de arquivos virtual *proc*, permitindo a busca das estatísticas em tempo de execução.

Para tal, foram medidas tarefas I/O bound interativas, através do tocador de música

Tabela 1: Prioridades associadas aos dispositivos definidos

Dispositivo	Prioridade
/dev/dsp*	5
/dev/audio*	5
/dev/input/mice	3
/dev/input/mouse*	3
/dev/mga_vid	5
/dev/mouse	3
/dev/psaux	3
/dev/rtc	2
/dev/ttyS0	3
/dev/video*	5

ogg123[bib, 2003], bem como uma aplicação CPU bound composta por um laço infinito de processamento:

```
#include <stdio.h>

int main ()
{
        while (1);
}
```

Essas tarefas foram medidas em diferentes cenários: ambas sendo executadas isoladamente, e num momento de competição pela CPU.

Independente de qual tarefa (ou tarefas) foi testada, o cenário foi o mesmo: Com o kernel 2.6.6 $O(1)$ tradicional, com o kernel $O(1)$ com a política de estimativas do devprio, ambos numa máquina mono-processada. Os resultados são apresentados nas Figura 1 e 2. As medidas feitas estão representadas por *jiffies*, que indicam a quantidade de interrupções do timer desde o instante que o computador foi ligado [Rubini and Corbet, 2001].

O escalonador conseguiu prever mais rapidamente o nível de interatividade da tarefa ogg123, bem como manteve o seu nível mesmo quando ela (por apenas algumas fatias de tempo) fez rajadas de CPU, o que permitiu que o som continuasse a tocar sem interferência no processamento (trocas de contexto para alimentar o buffer de kernel da placa de som) da música.

Na competição pela CPU, o valor da prioridade nice da aplicação interativa foi incrementado, bem como a aplicação CPU bound foi punida, permitindo que a tarefa ogg123 tivesse uma menor latência e uma maior vasão do que a tarefa CPU bound *while1*.

Devido à checagem na camada VFS pela bonificação por uso de dispositivos interativos como parte da heurística acrescentada no estimador de interatividade, pode-se notar que houve um aumento da latência da aplicação puramente CPU bound.

Com isso, pode ser visto que utilizando a heurística de monitoramento de dispositivos interativos juntamente com a análise da média de sleep average da tarefa, conseguiu-se um aumento na freqüência com a qual escalonador disponibilizou a CPU para tarefas

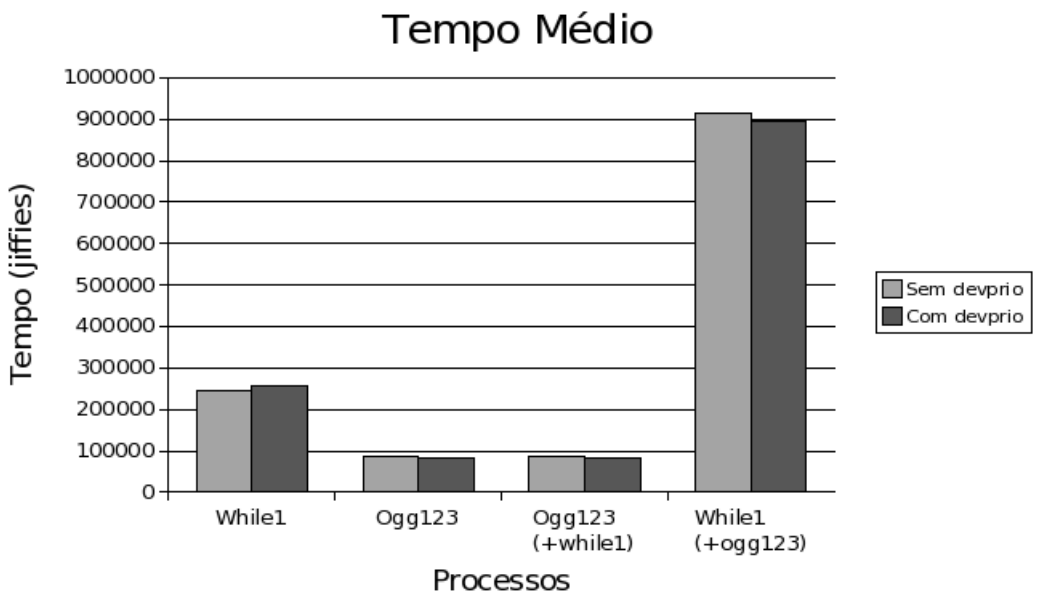

Figura 1: Tempo médio de espera

interativas do sistema.

Além disso, nota-se que como tarefas CPU bound tiveram um tempo médio de espera maior do que as tarefas interativas, demoraram mais para terminar seu processamento. Isto indica que o estimador de interatividade previu corretamente não só o nível de interatividade das tarefas, mas também o nível de **CPU bound** das tarefas, sendo que as tarefas que realmente não fazem I/O (apenas processamento CPU bound) recebem uma punição maior com relação à tarefas menos CPU bound.

5. Conclusões e trabalhos futuros

Através da utilização de um estimador de interatividade no kernel Linux, pôde-se adicionar conceitos de qualidade de serviço para o cenário de usuários de desktop. Porém, a definição simplista utilizada no kernel Linux apenas pela utilização do fator de espera de uma tarefa para determinar o seu nível de interatividade (tarefas I/O bound) se torna não ótima quando entram cenários reais de aplicativos multimídia, naturalmente interativos, e aplicações I/O bound, usadas para manutenção do sistema.

Para tal, torna-se necessário a adição de outros fatores de estimativa do nível de interatividade de uma tarefa, como a proposta de monitoração de dispositivos pré-definidos como interativos, resolvendo também o problema de oscilação de níveis de prioridade no atual escalonador.

Como trabalho futuro, torna-se necessária a dinamização do estimador de interatividade para a classificação, em tempo de execução, de dispositivos, baseado na história dos processos executados. Também se torna imperativo a não utilização da camada VFS para monitoração de dispositivos, pois a interação do escalonador com essa camada adiciona complexidade desnecessária para o escalonador.

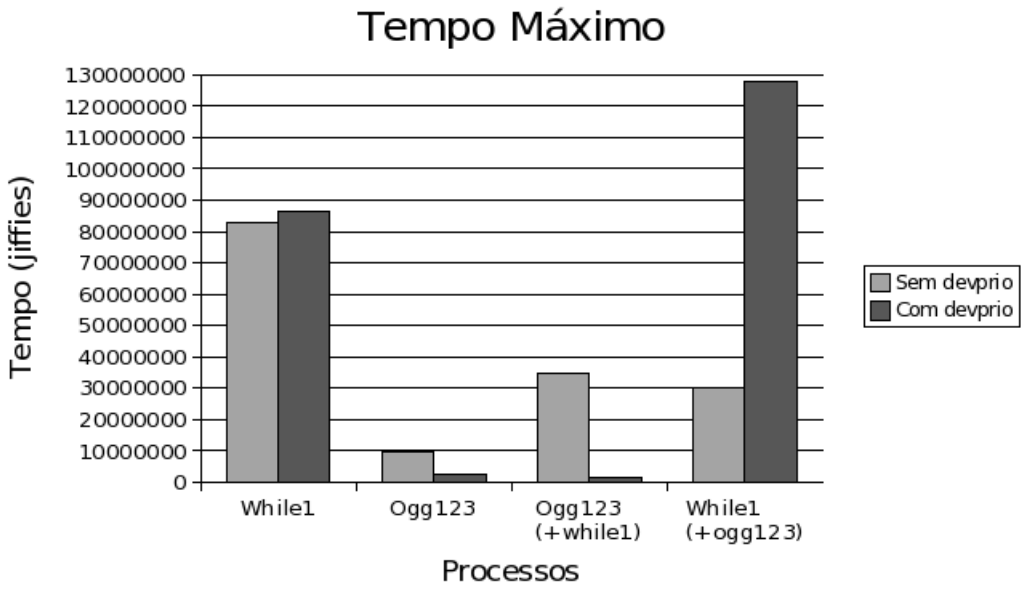

Figura 2: Tempo máximo de espera

Referências

(2003). Vorbis.com - open, free audio. http://www.vorbis.com. Projeto Ogg Vorbis.

Bovet, D. P. and Cesati, M. (2002). *Understanding the linux kernel*. O'Reilly, 2nd edition.

Love, R. M. (2003a). Interactive kernel performance. In *Proceedings of the Linux Symposium*, pages 301–312.

Love, R. M. (2003b). Introducing the 2.6 kernel. *Linux Journal*, pages 52–57.

Love, R. M. (2003c). *Linux Kernel Development*. SAMS, Developer Library Series, 1st edition.

Molnar, I. (2003). O(1) scheduler. kernel/sched.c. Fonte do kernel Linux.

Rubini, A. and Corbet, J. (2001). *Linux Device Drivers*. O'Reilly, 2nd edition.

Silberschatz, A., Galvin, P. B., and Gagne, G. (2003). *Operating system concepts*. John Wiley & Sons, 6th edition.

Williams, C. (2002). Linux kernel latency.