Avaliação do comportamento de sistemas operacionais de mercado em situação de *thrashing*

Douglas Santos, Carlos Maziero

[1] Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná
Curitiba, PR - Brasil

{dsantos,maziero}@ppgia.pucpr.br

Resumo. *A memória virtual oferece aos processos mais memória que aquela fisicamente disponível, usando um disco como extensão de memória. Quando não há memória RAM para atender os processos, o sistema se torna lento, pois gasta seu tempo fazendo troca de páginas, caracterizando o* thrashing. *Este trabalho traz uma avaliação de alguns sistemas operacionais de mercado sob* thrashing. *Uma ferramenta de* benchmark *foi definida para conduzir cada sistema a um* thrashing *controlado e então de volta à operação normal. Além disso, foram identificadas as informações de desempenho disponíveis em cada sistema e os mecanismos usados para coletá-las.*

Abstract. *Virtual memory allows offering to processes more memory than that physically available in the system, using a disc as a memory extension. When there is not enough RAM memory to satisfy the processes, the system becomes slow, because it spends the time doing paging, characterizing memory* thrashing. *This paper presents an evaluation of some commodity operating systems under thrashing. A portable benchmark tool was developed to bring each system to a thrashing and then back to normality. We also identify the performance data available in each system and the mechanisms available to collect them.*

1. Introdução

Em um sistema operacional de propósito geral, os mecanismos de memória virtual permitem oferecer aos processos mais memória que aquela disponível fisicamente, usando um disco externo como extensão da memória. Quando a demanda por memória aumenta, o sistema seleciona páginas e as move para o disco, para liberar memória. Quando essas páginas forem solicitadas por seus donos, o gerenciador as move de volta para a RAM. O movimento de páginas entre memória e disco é denominado *paginação*. Quando a memória RAM é insuficiente para atender as demandas dos processos, a quantidade de paginações aumenta muito e o desempenho do sistema se degrada, pois o sistema gasta seu tempo fazendo trocas de páginas. Caso a demanda por memória seja muito intensa (muita paginação), o sistema entra em *thrashing*, se tornando praticamente inoperante.

Este trabalho traz uma avaliação quantitativa de alguns sistemas operacionais de mercado sob *thrashing*. Foram selecionados sistemas de amplo uso e foi definido um *benchmark* próprio, que induz uma situação de *thrashing* controlada e seu retorno à normalidade. Este texto está assim organizado: a seção 2 discute o conceito de *thrashing*; a seção 3 apresenta algumas ferramentas de *benchmark* de memória; a seção 4 descreve o estudo realizado; a seção 5 discute os resultados obtidos e a seção 6 conclui o artigo.

2. *Thrashing*

O termo *thrashing* [Denning 1968] designa uma situação de consumo excessivo de memória em um computador, quando o elevado número de paginações paralisa os processos e torna o sistema inutilizável. A dinâmica do *thrashing* é simples: se um processo não dispuser de todas as páginas necessárias para processar, ele irá gerar faltas de páginas. O gerenciador de memória irá então carregar suas páginas na memória, mas para isso terá de descarregar páginas de outros processos. Enquanto aquele processo aguarda a carga de suas páginas, outros processos podem ocupar a CPU, gerando mais faltas de páginas, e assim sucessivamente. Como nenhum processos tem páginas que necessita para processar, a taxa de uso da CPU cai drasticamente e o sistema avança muito lentamente.

O fenômeno de *thrashing* é influenciado por vários fatores, como a quantidade de memória RAM do sistema, o número de processos ativos, a localidade de referências de cada processo, o algoritmo de seleção de páginas e a velocidade dos discos. A solução mais óbvia para o problema é aumentar a quantidade de memória RAM do sistema, o que nem sempre é possível. Outras soluções envolvem mudanças nos algoritmos de troca de páginas e/ou no escalonador de processos. Na seqüência são descritos os métodos de tratamento de *thrashing* presentes nas versões mais recentes dos sistemas operacionais sob estudo neste texto.

O sistema **FreeBSD** detecta *thrashing* observando o uso da memória [McKusick and Neville-Neil 2004]. Quando o sistema possuir pouca memória livre e uma taxa elevada de requisições de memória, ele se considera em *thrashing*. O sistema reduz o *thrashing* fazendo com que o último processo que executou não volte ao processador durante algum tempo. Isto permite que o sistema mova as páginas desse processo ao disco, liberando memória para os demais. Se o *thrashing* continuar, outros processos são bloqueados, até que exista memória suficiente para os demais. Os processos bloqueados podem voltar a executar depois de 20 segundos, o que pode provocar novo *thrashing*, levando a novos processos bloqueados, e assim sucessivamente.

O núcleo do **Linux** incorpora uma técnica de *token swap* [Jiang and Zhang 2005] para tratar o *thrashing*: um *token* único é atribuído a um processo no sistema; esse processo não perderá páginas enquanto detiver o *token*, permitindo a ele prosseguir sua execução. A atribuição do *token* a um processo segue algumas regras: a) o *token* não mudou de detentor nos últimos 2 segundos; b) o atual detentor do *token* não gerou faltas de página desde que o recebeu; e c) o processo solicitante não recebeu o *token* recentemente. O Linux também implementa níveis de prioridade em seu algoritmo de seleção de páginas: em casos extremos de *trashing*, o *token* não é mais considerado e o gerenciador de memória escolhe processos "vítimas" para eliminar, até que o problema se amenize.

O gerenciador de memória do sistema **OpenSolaris** trabalha em três estados: *normal*, *soft swap* e *hard swap*. No estado normal, é usado um mecanismo de paginação clássico (*two-handed clock*). Se a quantidade de memória livre cair abaixo de um patamar chamado `desfree` durante mais de 30 segundos, o sistema comuta para o modo *soft swap*, no qual processos inteiros são enviados para o disco. Se ainda assim a atividade de paginação continuar elevada, o sistema entra no estado *hard swap*, no qual o núcleo descarrega todos os seus módulos e memória cache sem uso, além de mover para o disco seqüencialmente os processos que estejam dormindo há mais tempo, até que a atividade de paginação diminua [Mauro and McDougall 2001].

O **Windows XP** usa um arquivo de tamanho variável como área de *swapping*, que pode ser aumentado em situações críticas. Seu algoritmo de troca de páginas é baseado em *Working sets* com agrupamento de páginas [Russinovich and Solomon 2004]. Uma *thread* de núcleo periodicamente revisa os tamanhos dos conjuntos de trabalho, removendo páginas caso haja pouca memória disponível. Outra *thread* de núcleo migra gradativamente processos há muito tempo suspensos para a área de *swap*. Na documentação analisada não foi encontrada menção a uma politica explícita para *thrashing*.

3. Benchmark de sistemas de memória

Em sua maioria, os programas de *benchmarking* de memória permitem avaliar a velocidade das operações envolvendo a memória RAM e seus caches. Duas medidas freqüentes são a *largura de banda* e a *latência* de acesso à memória. Nesta seção são apresentadas algumas ferramentas para *benchmarking* de memória, inicialmente aquelas que executam em mais de um sistema operacional e que possuem código fonte aberto (condição importante para compreender como a ferramenta realiza suas medidas).

O programa *Bandwidth* [bandwidth 2008] mede velocidades de leitura/escrita do cache L2, da memória RAM, da memória de video e a velocidade de execução de funções como `memset`, `memcpy` e `bzero`. O programa *CacheBench* [Mucci et al. 1998] avalia o desempenho dos *caches*, medindo suas larguras de banda. O sistema *LMbench* [McVoy and Staelin 1996] mede apenas a transferência de dados entre CPU, cache, memória, rede e disco, em sistemas UNIX. Já o sistema *nBench* [nbench 2008] avalia o desempenho de CPU e largura de banda de memória, simulando operações usadas por aplicações populares. STREAM [Stream 2008] é uma ferramenta portável para medir o tempo necessário para copiar regiões de memória e medir sua largura de banda.

A *Standard Performance Evaluation Corporation* (SPEC) mantém um conjunto de *benchmarks* para computadores modernos [Henning 2006]. A ferramenta SPEC CPU funciona em sistemas Unix e Windows e permite medir o desempenho do processador, desempenho do compilador e largura de banda de memória. Além dessas ferramentas, existem diversas outras, abertas e comerciais, mas nenhuma das ferramentas pesquisadas oferece funcionalidade para avaliar a resposta de um sistema ao *thrashing*.

4. O estudo realizado

O objetivo deste trabalho é avaliar o comportamento de alguns sistemas operacionais de mercado em condições de *thrashing*. Foram selecionados para este estudo os sistemas operacionais *FreeBSD*, *Linux*, *OpenSolaris* e *Windows XP*, por serem os principais sistemas operacionais *desktop* do mercado executando sobre a plataforma Intel x86 padrão (o que excluiu o Apple Mac OS X). Essa plataforma foi escolhida por ser de fácil acesso e ser bem suportada pelos sistemas escolhidos. Estes sistemas podem operar em modo *desktop* ou como ambientes multi-usuários com terminais gráficos remotos. Nesse contexto, o *thrashing* pode ser particularmente problemático, pois a ação de um usuário pode afetar diretamente a disponibilidade do sistema para os demais.

Como não identificamos ferramentas para avaliar o comportamento de um sistema sob *thrashing*, foi definido um *benchmark* próprio, que conduz o sistema de um estado normal de operação ao *thrashing* e depois de volta a uma situação normal. Assim, pode-se avaliar como cada sistema gerencia a memória durante o *thrashing* e quão rápido consegue

se recuperar dele, quando a demanda por memória se normalizar. O comportamento de cada sistema foi observado através do uso de processador em modo usuário e em modo sistema (%) e da taxa de páginas lidas/escritas em disco (*page-in/page-out*). Cada sistema oferece essas informações através de uma API própria, exigindo um programa de coleta de dados específico para cada um.

4.1. A ferramenta de *benchmark*

A ferramenta desenvolvida possui três componentes: um conjunto de N *Processos consumidores*, responsáveis por provocar o *thrashing*, um *processo de medição*, que coleta dados de desempenho, e um *processo pai*, que lança os demais processos. Cada consumidor aloca uma grande área de memória, onde executa ciclos de escrita, alternados com períodos de espera (Algoritmo 1). As escritas são feitas em posições aleatórias da memória, para forçar uma baixa localidade de referência e não favorecer nenhum sistema. Os parâmetros cuja influência foi observada são: o número de operações de escrita em cada ciclo (W), a duração da espera entre dois ciclos de escrita (t_w) e a defasagem entre processos consumidores (t_c), ou seja, o tempo decorrido entre o início da atividade de dois processos consumidores sucessivos (pc_i e pc_{i+1}). Estes parâmetros foram escolhidos de forma empírica, por serem os mais influentes nos diversos experimentos realizados.

Algoritmo 1 Processo consumidor $pc_i(N, i, W, t_c, t_w)$

Entrada: N (numero total de consumidores), $1 \leq i \leq N$ (índice do processo corrente), t (relógio do sistema)

1: sleep (10 + $i \times t_c$) // cada processo se ativa em um momento próprio
2: mem = malloc (100×1024^2) // aloca 100 MB de memória RAM
3: $t_p = (N \times t_c) + t_c$ // duração da atividade de cada consumidor
4: $t_f = t + t_p$ // data do fim da execução deste processo
5: **while** $t \leq t_f$ **do**
6:     **for** $k = 1$ to W **do**
7:         val = random ($0 \ldots 255$)
8:         pos = random ($0 \ldots 100 \times 1024^2$)
9:         mem [pos] = val // escreve valor aleatório em posição aleatória
10:     **end for**
11:     sleep (t_w) // espera entre ciclos de escritas
12: **end while**
13: free (mem) // libera a memória alocada

Os processos consumidores foram escritos em C, devido a sua portabilidade. Com isso, o processo consumidor é exatamente o mesmo programa nos quatro sistemas avaliados. Os processos de medição também foram escritos em C, com exceção do Windows, onde foi utilizado um programa nativo dessa plataforma. O processo pai foi escrito em Java, pois suas funções de criação de novos processos independem de plataforma.

4.2. Coleta de informações do núcleo

Basicamente, o processo de medição coleta as informações de processador e memória providas pelo núcleo e as salva em um arquivo, a cada segundo. O processo de medição é específico para cada sistema operacional, pois as interfaces de acesso às informações de núcleo são distintas para cada sistema, conforme descrito a seguir.

No sistema **FreeBSD**, a ferramenta *sysctl* permite acessar/ajustar mais de 500 variáveis do núcleo, organizadas hierarquicamente. As variáveis referentes à utilização de CPU (em *ticks* de relógio) e de memória são: `kern.cp_time` (vetor com tempos de CPU em modo usuário e sistema), `vm.stats.vm.v_swappgsin` (páginas lidas no último segundo) e `vm.stats.vm.v_swappgsout` (páginas escritas no último segundo). O **Linux** implementa o sistema de arquivos virtual */proc* [Cowardin 1997], que oferece uma hierarquia de informações do núcleo através da abstração de arquivos e diretórios. Muitas ferramentas de monitoração, como o *top* e *vmstat* usam o sistema */proc* como fonte de informação. Em nossos experimentos, os dados de consumo de CPU foram obtidos do arquivo `/proc/stat`, enquanto os dados sobre operações de paginação foram obtidos do arquivo `/proc/vmstat` (campos `pgpgin` e `pgpgout`).

O **OpenSolaris** oferece funções e estruturas de dados para tratar informações do núcleo e de módulos, denominada *kstat* e representada pelo dispositivo virtual `/dev/kstat`. A função `kstat_data_lookup` permite localizar uma variável específica e obter seu valor. As variáveis usadas são: `cpu_ticks_user` (tempo de CPU em modo usuário), `cpu_ticks_kernel` (tempo de CPU em modo sistema), `pgswapin` (páginas lidas no último segundo) e `pgswapout` (páginas escritas no último segundo). O **Windows** não oferece uma API pública para coletar dados do núcleo. Ao invés disso, uma ferramenta chamada *perfmon* (Monitor do Sistema) permite acessar dados de desempenho do sistema. O *perfmon* classifica os recursos do computador como objetos, aos quais são associados contadores. Os contadores considerados para o objeto CPU (em %) e para o objeto memória foram: `%User Time` (tempo de CPU em modo usuário), `%Privileged Time` (idem, modo sistema), `Pages Input/sec` (páginas lidas no último segundo) e `Pages Output/sec` (páginas escritas no último segundo).

4.3. O ambiente de experimentação

Nos experimentos foi usado um PC IBM ThinkCentre S50, com processador Intel Pentium 4 2.6 GHz, placa-mãe Intel 865G com componentes *on-board* e barramento frontal de 533 MHz. A máquina possui 1 GB de memória RAM Kingston DDR PC2700, com *clock* de 333 MHz e tempo de acesso de 6 ns. O disco rígido usado foi um Seagate Barracuda ST340014A de 40 GB IDE (conector ATA-100, 2 MB de buffer interno, velocidade 7200 RPM, taxa de transferência de 100 MB/s e tempo de busca de 8.5 ms). O disco foi dividido em 4 partições iguais. Um espaço de 2 GB foi criado em arquivo para a área de *swap* em cada partição, com a finalidade de tornar a estrutura dos sistemas o mais próxima possível, uma vez que a plataforma Windows não utiliza partição de *swap*.

Foi feita uma instalação *desktop* default para cada sistema operacional e nenhum outro processo, a não ser os processos default do sistema operacional, estava sendo executado durante a experimentação. O compilador escolhido foi o GCC (*GNU Compiler Collection*), por ser amplamente usado e possuir versões nos sistemas operacionais avaliados. Nenhuma configuração específica de otimização no compilador foi utilizada. A Tabela 1 indica a configuração de cada sistema, incluindo o número de processos ativos, a quantidade de memória livre e de área de *swap* usada logo após a inicialização.

5. Resultados obtidos

Nos experimentos, foi avaliada a influência dos parâmetros na seção 4.1, ou seja, o número de escritas na memória, a espera entre ciclos de escrita e a espera entre a atividade

Tabela 1. Dados dos sistemas operacionais sob estudo

Sistema	Versão	Compilador	Interface	Proc	RAM / Cache	Swap
FreeBSD	PC-BSD 7.0	GCC 4.2.1	KDE 4	66	630 / 70 MB	0
Linux	OpenSUSE 10.3	GCC 4.2.1	Gnome 2.20	62	730 / 140 MB	0
OpenSolaris	2008.11	GCC 3.4.3	Gnome 2.22	79	370 / 100 MB	0
Windows	XP Pro SP3	GCC 4.2.3	nativa	16	830 / 70 MB	80 MB

de dois processos. Para cada parâmetro, foram registrados os valores de consumo de CPU por processos e pelo sistema e o número de páginas lidas (*page in*) e de páginas escritas (*page out*). Esses dados foram coletados a cada segundo por um processo de medição executando com prioridade normal, porém com permissões de super-usuário. Em todos os experimentos, foram criados 25 processos, cada um alocando 100 MB de memória RAM. A máquina foi reiniciada após cada medição.

Em todos os sistemas estudados, os dados observados apresentam forte variação entre uma medida e outra, tornando sua visualização e interpretação inviáveis. Por esta razão, foi adotado um procedimento de suavização (curvas de Bézier) para a representação gráfica dos resultados (Figura 1). Assim, os gráficos apresentados nas próximas seções não indicam os valores exatos observados para cada cada variável, mas sua tendência, que se manteve estável nas várias repetições de cada experimento.

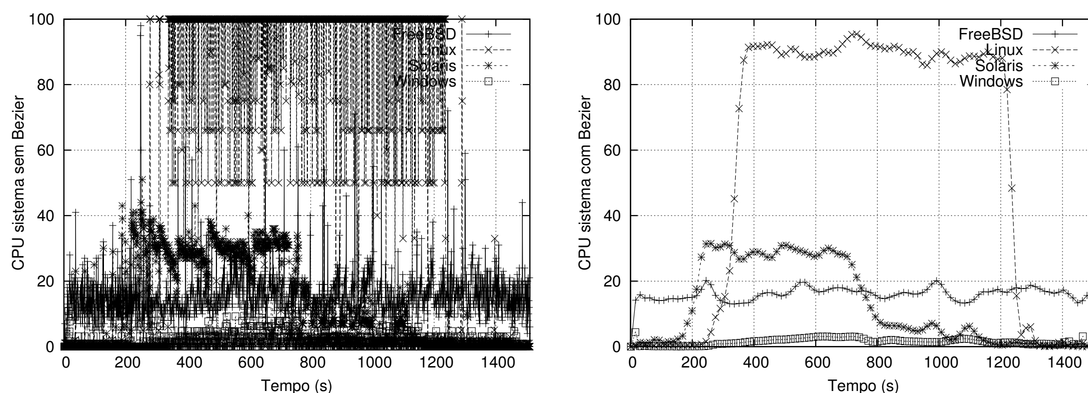

Figura 1. Plotagem dos dados brutos (esq.) e com suavização (dir.)

5.1. Influência do número de escritas por ciclo

As Figuras 2 e 3 mostram o consumo de CPU para $W = 1.000$ e 50.000 escritas/ciclo, mantendo o tempo de espera entre dois ciclos $t_w = 100ms$ e o tempo de espera entre dois processos $t_c = 30s$. As Figuras 4 e 5 indicam o número de *page in/out* nas duas situações.

O consumo de CPU em modo usuário foi baixo quando avaliado em 1.000 escritas/ciclo, com exceção do OpenSolaris, que apresentou picos de 40% de CPU no início e final da execução. O FreeBSD apresentou um consumo de CPU consistentemente acima dos demais, tanto em modo usuário quanto em modo sistema. Já com 50.000 escritas/ciclo, o comportamento do Linux se distancia dos demais, sobretudo em modo sistema: a atividade de processador no Linux chega a 100%, considerando-se a soma dos dois modos de operação. O FreeBSD conserva seu comportamento estável, enquanto o OpenSolaris apresenta um consumo inicial de aproximadamente 30% da CPU, que vai decrescendo ao longo do tempo, sem os picos de processamento do caso anterior.

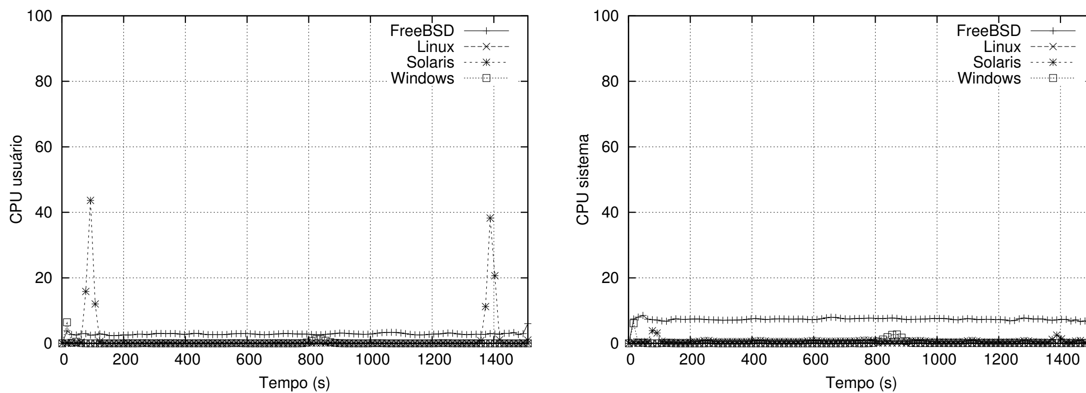

Figura 2. Consumo de CPU para 1.000 escritas/ciclo

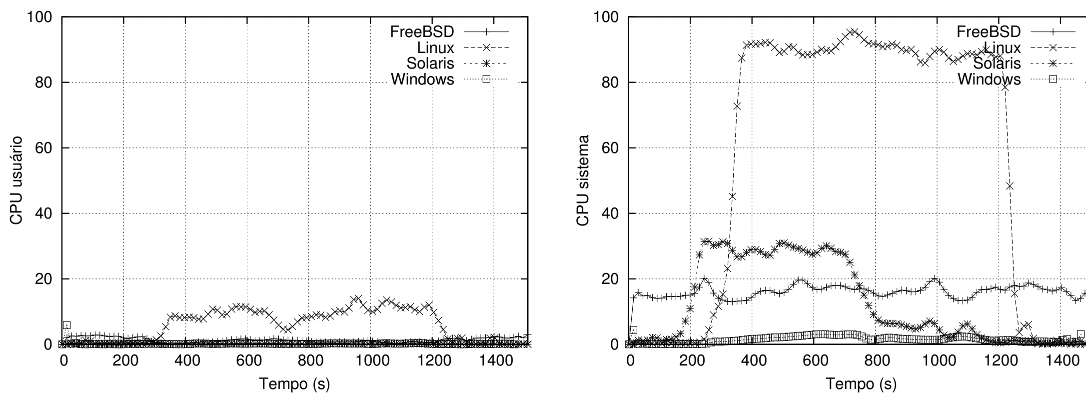

Figura 3. Consumo de CPU para 50.000 escritas/ciclo

O Windows apresenta pouco consumo de CPU em modo usuário, mesmo quando avaliado em 50.000 escritas/ciclo. Há um consumo aproximado de 6% apenas quando os processos então sendo ativados. Pode-se observar que o Windows é o sistema que apresenta menor consumo de CPU nos experimentos. Cabe observar que o Windows não conseguiu executar todos os processos dentro do tempo previsto t_p (Algoritmo 1).

O número de *page in/out* é bastante modesto quando avaliado em 1.000 escritas/ciclo. O Windows apresentou um maior valor nesta avaliação. Já quando avaliado em 50.000 escritas/ciclo, o número de *page in/out* é significativamente maior. O Linux é o sistema que apresenta a maior movimentação de páginas entre o disco e a memória, atingindo picos de quase 5.000 *page out* por segundo. Os demais sistemas não ultrapassaram

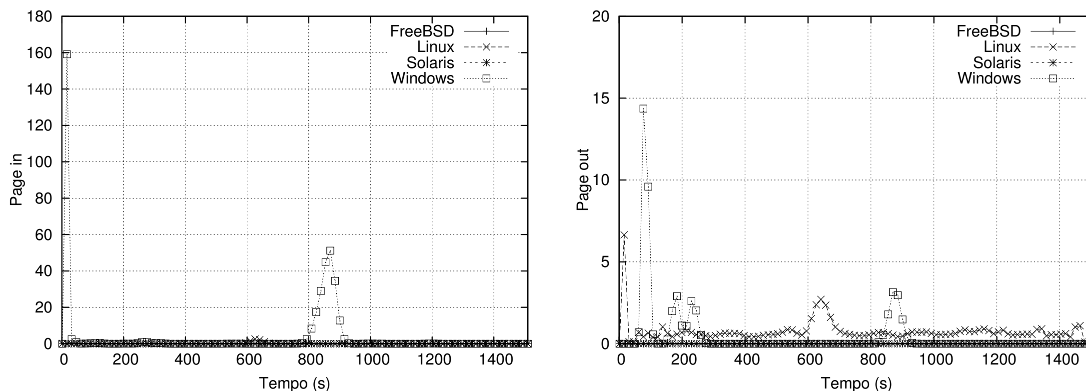

Figura 4. Operações de paginação para 1.000 escritas/ciclo

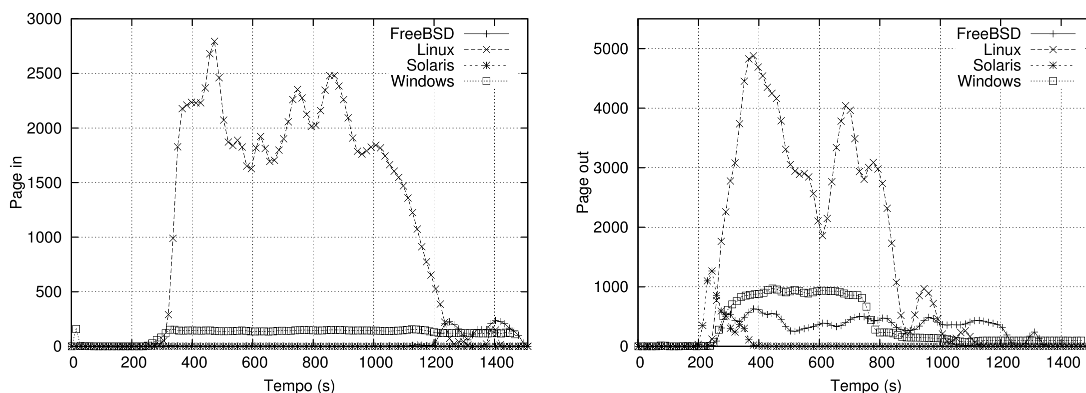

Figura 5. Operações de paginação para 50.000 escritas/ciclo

1.500 *page out* por segundo.

5.2. Influência do tempo de espera entre ciclos de escritas

As Figuras 6 e 7 mostram o consumo de CPU para $t_w = 100ms$ e $t_w = 1.000ms$, mantendo o número de escritas/ciclo $W = 10.000$ e o tempo de espera para ativação de cada processo $t_c = 30s$. As Figuras 8 e 9 apresentam o número de *page in/out* nesses experimentos.

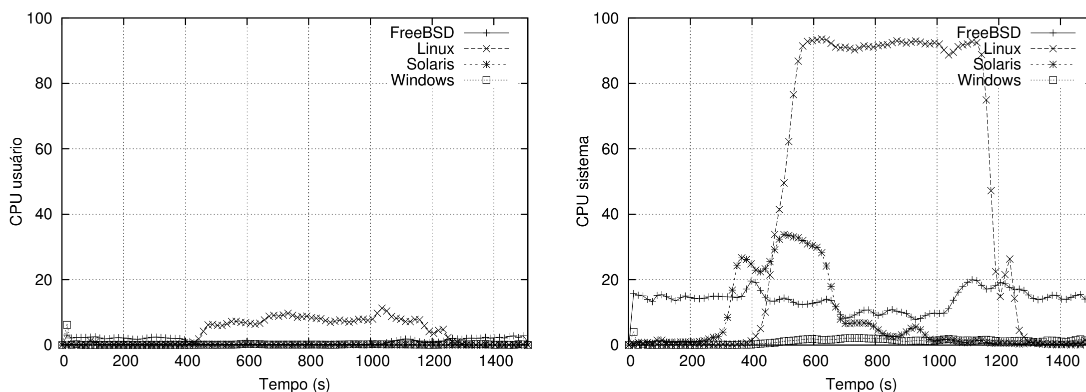

Figura 6. Consumo de CPU com tempo de espera de 100ms

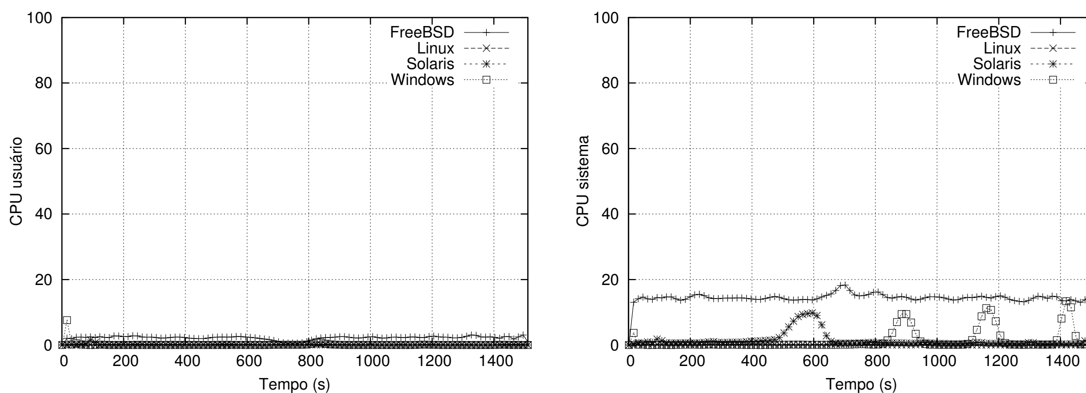

Figura 7. Consumo de CPU com tempo de espera de 1.000ms

Pode-se observar que quanto menor o tempo de espera entre ciclos de escritas, maior é o consumo de CPU. O Linux é o sistema com o maior consumo de CPU quando

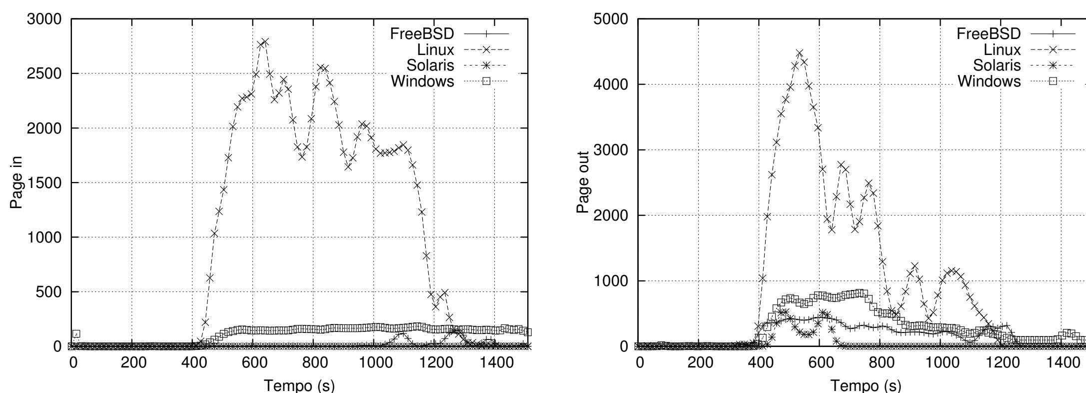

Figura 8. Operações de paginação com tempo de espera de 100ms

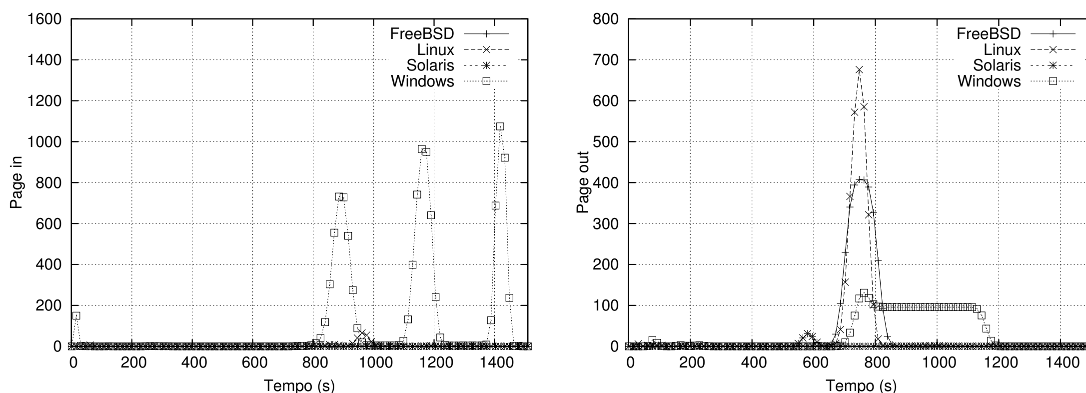

Figura 9. Operações de paginação com tempo de espera de 1.000ms

avaliado com tempo de espera de 100 ms. O FreeBSD apresenta um consumo médio de 15% de CPU em modo sistema, independentemente do tempo de espera entre ciclos de escrita. O Windows apresentou, em média, baixo consumo de CPU; no entanto, quando avaliado em 1.000ms apresentou picos de mais de 10% de uso de CPU em modo sistema. De forma geral, quanto maior o tempo de espera entre ciclos de escritas menor é o número de *page in/out*. O Linux apresenta o maior pico de *page in/out* por segundo, nos três primeiros experimentos. No entanto, o Windows apresentou maior taxa de *page in* quando avaliado em 1.000ms.

5.3. Influência do tempo entre duas ativações de processos

As Figuras 10 e 11 mostram o consumo de CPU para $t_c = 1s$ e $t_c = 30s$, mantendo o número de escritas por ciclo $W = 10.000$ e o tempo de espera entre ciclos $t_w = 100ms$. As Figuras 12 e 13 indicam o número de *page in/out* sob as mesmas condições.

O OpenSolaris apresentou o maior consumo de CPU em modo sistema quando avaliado com 1s de espera. Este pico se manteve próximo a 40% nos três experimentos. Já o Linux, apresentou maior consumo de CPU quando avaliado com t_c de 30 segundos, atingindo novamente picos de 100% de uso de CPU. Além disso, o Linux é o sistema que apresenta o maior número de *page in/out* nos três experimentos. O Linux atingiu mais de 4.500 *page out* por segundo. O Windows atingiu picos de aproximadamente 1.000 *page out* por segundo. O OpenSolaris, de modo geral, foi o sistema que apresentou o menor número de *page in/out* nos experimentos.

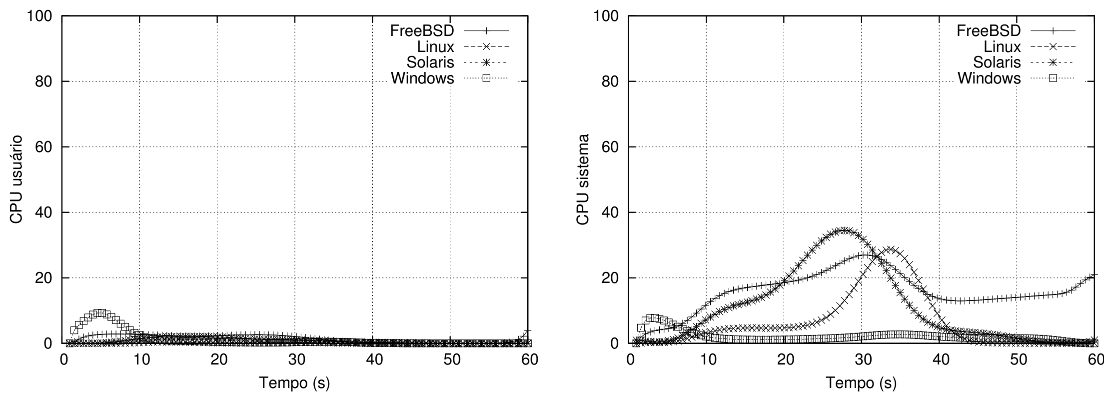

Figura 10. Consumo de CPU com 1s de espera entre dois processos

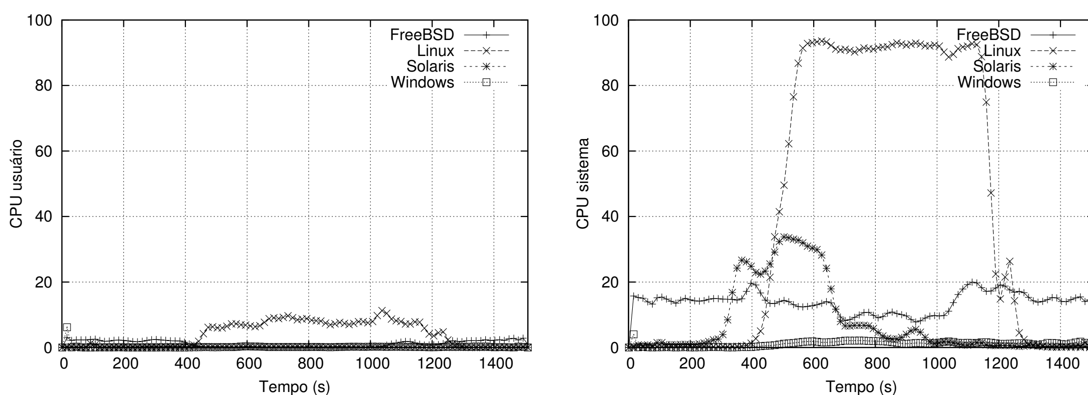

Figura 11. Consumo de CPU com 30s de espera entre dois processos

5.4. Estabilidade dos resultados

Para avaliar a estabilidade dos resultados, cada experimento foi repetido três vezes sob as mesmas condições. Tendo em vista o caráter dinâmico dos mecanismos de gerência de memória implementados pelos sistemas estudados, era de se esperar uma grande variação dos resultados obtidos durante os experimentos. Todavia, essa variação se mostrou modesta: a Figura 14 mostra os resultados obtidos no pior caso, um experimento sobre o sistema FreeBSD usando $W = 10.000$ escritas/ciclo, $t_w = 100ms$ e $t_c = 10s$. Todos os demais experimentos tiveram resultados mais estáveis.

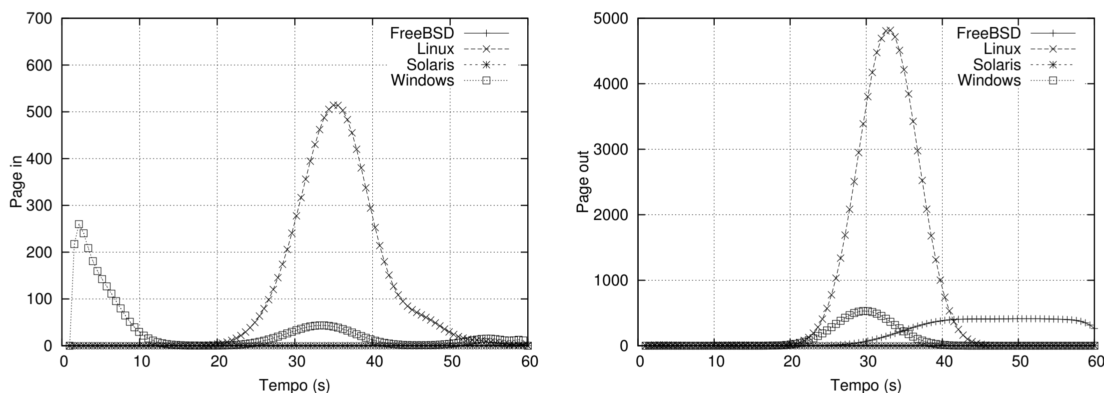

Figura 12. Operações de paginação com 1s de espera entre dois processos

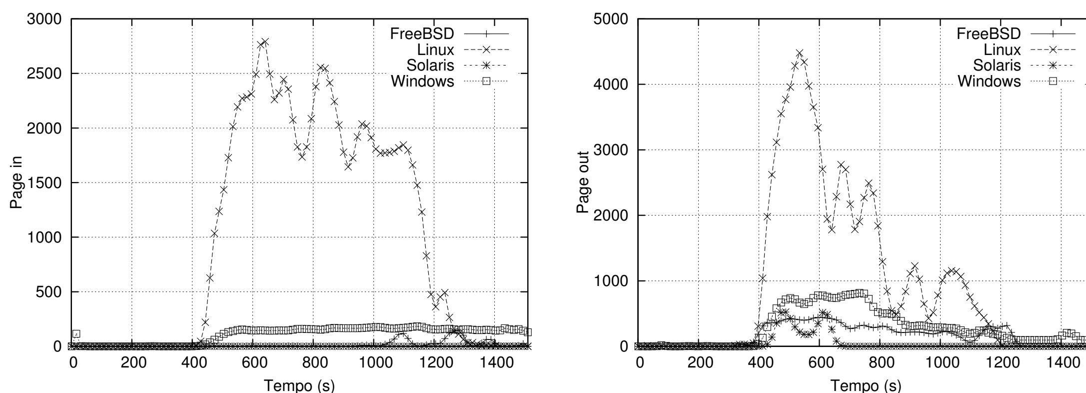

Figura 13. Operações de paginação com 30s de espera entre dois processos

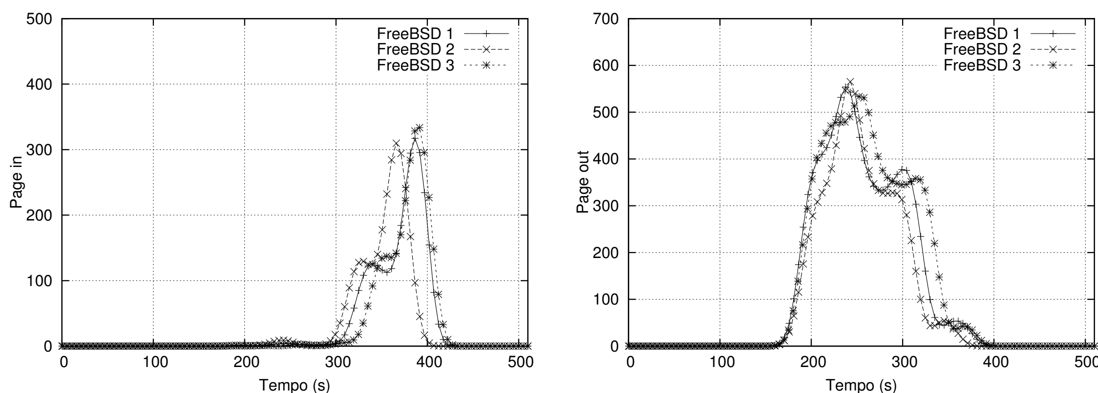

Figura 14. Variação do número de *page in/out* no FreeBSD

5.5. Avaliação dos resultados

De forma geral, os mecanismos de tratamento de *thrashing* dos sistemas avaliados não são suficientes para conter de forma eficiente esse fenômeno. Cada sistema se comportou de forma distinta nos experimentos realizados. Além do comportamento demonstrado nos gráficos apresentados, há que se considerar também a impressão subjetiva do usuário de um sistema sob *thrashing*. De forma geral, o FreeBSD é o sistema que apresentou a maior estabilidade de consumo de CPU, seja em modo usuário ou em modo sistema, durante os experimentos. Esse sistema, assim como o OpenSolaris, apresentou os maiores atrasos na interatividade do sistema, de acordo com o que foi percebido pela interação do usuário.

O OpenSolaris apresenta um crescimento inicial de consumo CPU, principalmente em modo de sistema, após a ativação dos processos. Esse consumo começa a decrescer de forma significativa à medida em que mais processos vão sendo ativados. O OpenSolaris foi o sistema que apresentou maior consumo de memória RAM antes dos experimentos, e também foi o que apresentou maior perda de interatividade, conforme percebido pela interação do usuário. O Windows apresentou, em geral, baixo consumo de CPU durante os experimentos. Ao contrário dos demais sistemas, o Windows atrasou a atividade de alguns processos, que não terminaram no prazo esperado. Além disso, foi o sistema que levou mais tempo para se recuperar do *thrashing*, sob a ótica da interatividade.

Ao contrário dos demais sistemas, em alguns momentos o Linux apresentou 100% de consumo de CPU, dois quais 90% referentes a consumo de CPU em modo de sistema. Além disso, o Linux apresentou os maiores picos de *page in/out*. Pode-se concluir que esse consumo elevado de CPU em modo sistema foi utilizado para o processamento de

page in/out, pois é nesse período que ocorrem os maiores picos. Pode-se observar que o mecanismo de *token* implementado no Linux consegue um desempenho melhor que os demais, justamente por ceder privilégios para um determinado processo em um dado instante. Assim, ao menos um processo consegue um avanço substancial em sua execução. Sob a ótica de usuário, o Linux foi o sistema que apresentou a menor degradação de interatividade e a mais rápida recuperação do *thrashing*. Essa percepção não é facilmente mensurável, mas pode ser facilmente percebida pela interação do usuário.

6. Conclusão

Este artigo apresentou a avaliação de alguns sistemas operacionais de mercado em situação de *thrashing* de memória. Foi proposta uma ferramenta portável que conduz cada sistema operacional a um *thrashing* controlado e posteriormente de volta à operação normal. Além disso, foram identificadas as informações de desempenho disponíveis em cada sistema e os mecanismos usados para coletá-las. Foram realizados experimentos que avaliam o comportamento dos sistemas sob estudo durante e após o *thrashing*. O impacto causado pelo *thrashing* parece ser ainda pouco estudado; em nossas pesquisas não conseguimos identificar outros estudos quantitativos desse fenômeno. O presente estudo pode ser estendido para abranger outros sistemas operacionais populares, como o *MacOS X*.

Referências

bandwidth (2008). *Bandwidth*. http://home.comcast.net/∼fbui/bandwidth.html.

Cowardin, J. (1997). A proc buffer for kernel instrumentation. Master's thesis, The College of William & Mary.

Denning, P. J. (1968). "Thrashing: its causes and prevention". In *AFIPS Fall Joint Computer Conference*.

Henning, J. L. (2006). *SPEC CPU2006 Benchmark Descriptions*. http://www.spec.org.

Jiang, S. and Zhang, X. (2005). Token-ordered LRU: an effective page replacement policy and its implementation in Linux systems. *Performance Evaluation*, 60(1-4):5 – 29.

Mauro, J. and McDougall, R. (2001). *Solaris Internals: Core Kernel Components*. Prentice Hall PTR.

McKusick, M. K. and Neville-Neil, G. V. (2004). *The Design and Implementation of the FreeBSD Operating System*. Pearson Education.

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

Mucci, P. J., London, K., and Thurman, J. (1998). *The CacheBench Report*. http://www.cs.utk.edu/∼mucci/DOD/cachebench.ps.

nbench (2008). *nbench*. http://www.tux.org/∼mayer/linux/bmark.html.

Russinovich, M. and Solomon, D. (2004). *Microsoft Windows Internals, Fourth Edition*. Microsoft Press.

Stream (2008). *STREAM bench*. http://www.streambench.org.