Sistemas Operacionais: Conceitos e Mecanismos

V - Gerência de Memória

Prof. Carlos Alberto Maziero
DInf UFPR
http://www.inf.ufpr.br/maziero

4 de agosto de 2017

http://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=so:so-cap05.pdf

Este texto foi produzido usando exclusivamente software livre: Sistema Operacional *GNU/Linux* (distribuições *Fedora* e *Ubuntu*), compilador de texto LaTeX 2ε, gerenciador de referências *BibTeX*, editor gráfico *Inkscape*, criadores de gráficos *GNUPlot* e *GraphViz* e processador PS/PDF *GhostScript*, entre outros.

Fragmentação

Ao longo da vida de um sistema, áreas de memória são liberadas por processos que concluem sua execução e outras áreas são alocadas por novos processos, de forma contínua. Com isso, podem surgir áreas livres (vazios ou *buracos* na memória) entre os processos, o que constitui um problema conhecido como *fragmentação externa*. Esse problema somente afeta as estratégias de alocação que trabalham com blocos de tamanho variável, como a alocação contígua e a alocação segmentada. Por outro lado, a alocação paginada sempre trabalha com blocos de mesmo tamanho (os quadros e páginas), sendo por isso imune à fragmentação externa.

A fragmentação externa é prejudicial porque limita a capacidade de alocação de memória no sistema. A Figura20apresenta um sistema com alocação contígua de memória no qual ocorre fragmentação externa. Nessa Figura, observa-se que existem 68 MBytes de memória livre em quatro áreas separadas (A_1...A_4), mas somente processos com até 28 MBytes podem ser alocados (usando a maior área livre, A_4). Além disso, quanto mais fragmentada estiver a memória livre, maior o esforço necessário para gerenciá-la: as áreas livres são mantidas em uma lista encadeada de área de memória, que é manipulada a cada pedido de alocação ou liberação de memória.

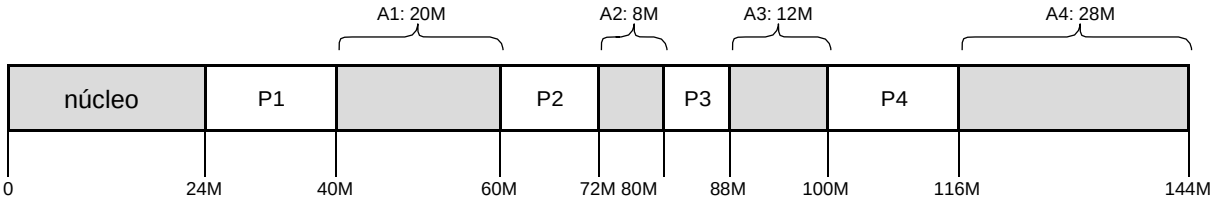


Figura 20: Memória com fragmentação externa.

Pode-se enfrentar o problema da fragmentação externa de duas formas: *minimizando* sua ocorrência, através de critérios de escolha das áreas a alocar, ou *desfragmentando* periodicamente a memória do sistema. Para minimizar a ocorrência de fragmentação externa, cada pedido de alocação deve ser analisado para encontrar a área de memória livre que melhor o atenda. Essa análise pode ser feita usando um dos seguintes critérios:

Melhor encaixe (*best-fit*) : consiste em escolher a menor área possível que possa atender à solicitação de alocação. Dessa forma, as áreas livres são usadas de forma otimizada, mas eventuais resíduos (sobras) podem ser pequenos demais para ter alguma utilidade.

Pior encaixe (*worst-fit*) : consiste em escolher sempre a maior área livre possível, de forma que os resíduos sejam grandes e possam ser usados em outras alocações.

Primeiro encaixe (*first-fit*) : consiste em escolher a primeira área livre que satisfaça o pedido de alocação; tem como vantagem a rapidez, sobretudo se a lista de áreas livres for muito longa.

Próximo encaixe (*next-fit*) : variante da anterior (*first-fit*) que consiste em percorrer a lista a partir da última área alocada ou liberada, para que o uso das áreas livres seja distribuído de forma mais homogênea no espaço de memória.

Diversas pesquisas [Johnstone and Wilson, 1999] demonstraram que as abordagens mais eficientes são a de melhor encaixe e a de primeiro encaixe, sendo esta última bem mais rápida. A Figura21ilustra essas estratégias.

Outra forma de tratar a fragmentação externa consiste em *desfragmentar* a memória periodicamente. Para tal, as áreas de memória usadas pelos processos devem ser movidas na memória de forma a concatenar as áreas livres e assim diminuir a fragmentação. Ao mover um processo na memória, suas informações de alocação (registrador base ou tabela de segmentos) devem ser ajustadas para refletir a nova posição do processo.

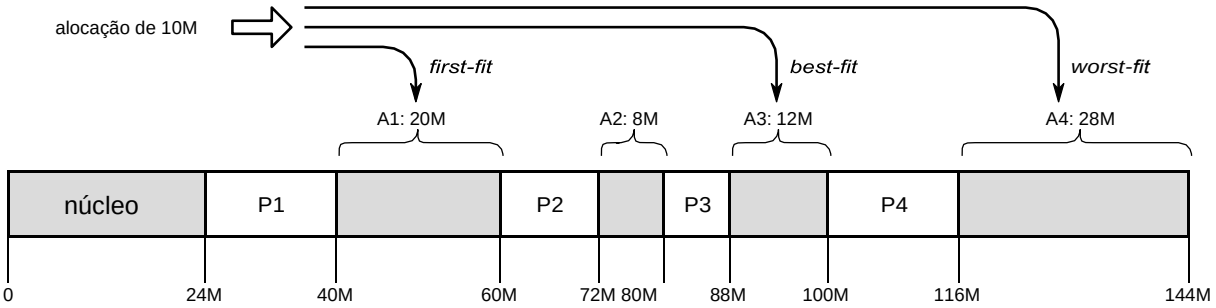
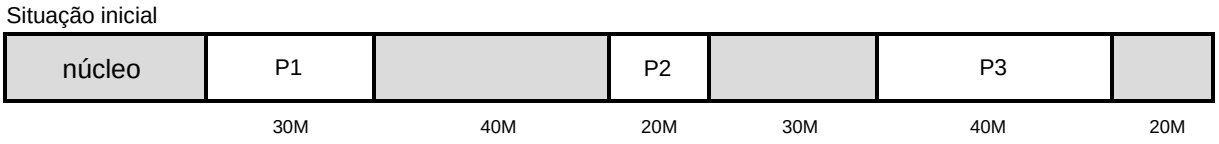


Figura 21: Estratégias para minimizar a fragmentação externa.

Obviamente, nenhum processo pode executar durante a desfragmentação. Portanto, é

importante que esse procedimento seja executado rapidamente e com pouca frequência, para não interferir nas atividades normais do sistema. Como as possibilidades de movimentação de processos podem ser muitas, a desfragmentação deve ser tratada como um problema de otimização combinatória, cuja solução ótima pode ser difícil de calcular. A Figura22ilustra três possibilidades de desfragmentação de uma determinada situação de memória; as três alternativas produzem o mesmo resultado, mas apresentam custos distintos.



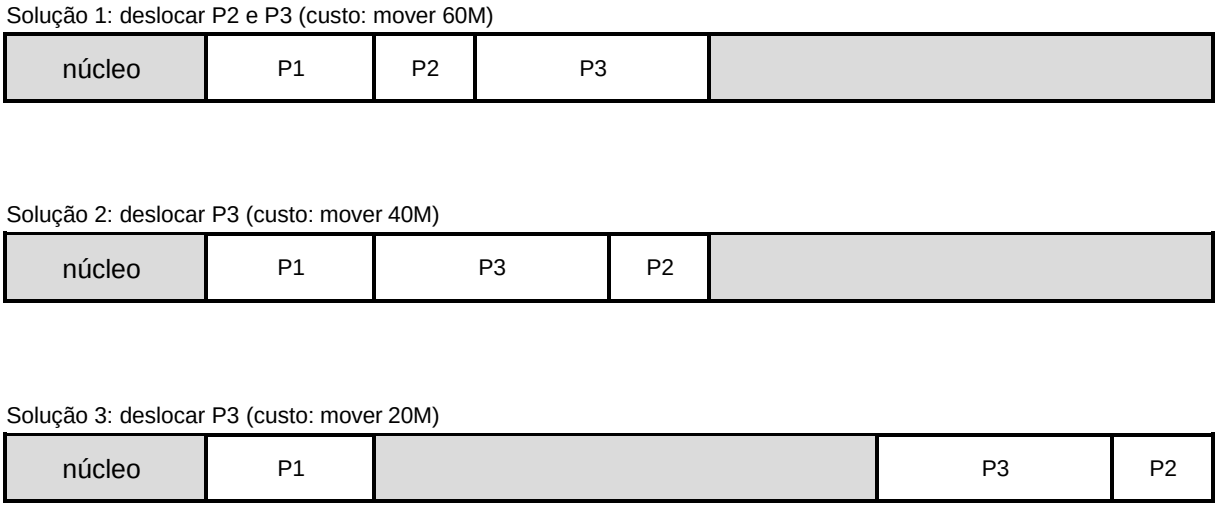
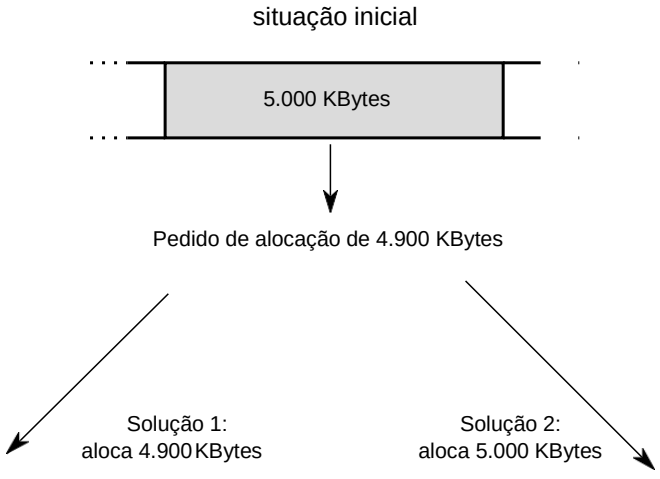


Figura 22: Possibilidades de desfragmentação.

Além da fragmentação externa, que afeta as áreas livres entre os processos, as estratégias de alocação de memória também podem apresentar a *fragmentação interna*, que pode ocorrer dentro das áreas alocadas aos processos. A Figura23apresenta uma situação onde ocorre esse problema: um novo processo requisita uma área de memória com 4.900 KBytes. Todavia, a área livre disponível tem 5.000 KBytes. Se for alocada

exatamente a área solicitada pelo processo (situação A), sobrará um fragmento residual com 100 KBytes, que é praticamente inútil para o sistema, pois é muito pequeno para acomodar novos processos. Além disso, essa área residual de 100 KBytes deve ser incluída na lista de áreas livres, o que representa um custo de gerência desnecessário. Outra possibilidade consiste em "arredondar" o tamanho da área solicitada pelo processo para 5.000 KBytes, ocupando totalmente aquela área livre (situação B). Assim, haverá uma pequena área de 100 KBytes no final da memória do processo, que provavelmente não será usada por ele.



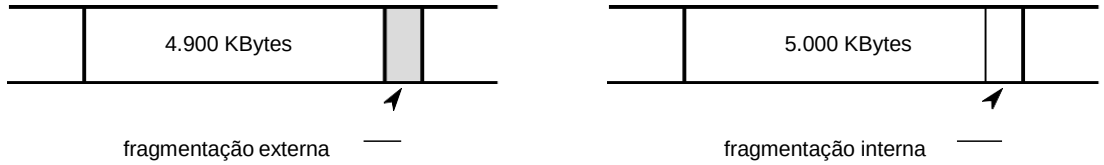


Figura 23: Fragmentação interna.

A fragmentação interna afeta todas as formas de alocação; as alocações contígua e segmentada sofrem menos com esse problema, pois o nível de arredondamento das alocações pode ser decidido caso a caso. No caso da alocação paginada, essa decisão não é possível, pois as alocações são feitas em páginas inteiras. Assim, em um sistema com páginas de 4 KBytes (4.096 bytes), um processo que solicite a alocação de 550.000 bytes (134,284 páginas) receberá 552.960 bytes (135 páginas), ou seja, 2.960 bytes a mais que o solicitado.

Em média, para cada processo haverá uma perda de 1/2 página de memória por fragmentação interna. Assim, uma forma de minimizar a perda por fragmentação interna seria usar páginas de menor tamanho (2K, 1K, 512 bytes ou ainda menos). Todavia, essa abordagem implica em ter mais páginas por processo, o que geraria tabelas de páginas maiores e com maior custo de gerência.

Referências

[Bansal and Modha, 2004] Bansal, S. and Modha, D. (2004). CAR: Clock with adaptive replacement. In *USENIX Conference on File and Storage Technologies*.

[Carr and Hennessy, 1981] Carr, R. and Hennessy, J. (1981). WSclock - a simple and effective algorithm for virtual memory management. In *ACM symposium on Operating systems principles*.

[Denning, 1980] Denning, P. (1980). Working sets past and present. *IEEE Transactions on Software Engineering*, 6(1):64–84.

[Denning, 2006] Denning, P. J. (2006). The locality principle. In Barria, J., editor, *Communication Networks and Computer Systems*, chapter 4, pages 43–67. Imperial College Press.

[Jiang and Zhang, 2002] Jiang, S. and Zhang, X. (2002). LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In *ACM SIGMETRICS Intl Conference on Measurement and Modeling of Computer Systems*, pages 31–42.

[Johnstone and Wilson, 1999] Johnstone, M. S. and Wilson, P. R. (1999). The memory fragmentation problem: solved? *ACM SIGPLAN Notices*, 34(3):26–36.

[Levine, 2000]Levine, J. (2000). *Linkers and Loaders*. Morgan Kaufmann.

[Navarro et al., 2002] Navarro, J., Iyer, S., Druschel, P., and Cox, A. (2002). Practical, transparent operating system support for superpages. In *5th USENIX Symposium on*

Operating Systems Design and Implementation, pages 89–104.

[Patterson and Henessy, 2005] Patterson, D. and Henessy, J. (2005). *Organização e Projeto de Computadores*. Campus.

[Silberschatz et al., 2001] Silberschatz, A., Galvin, P., and Gagne, G. (2001). *Sistemas Operacionais – Conceitos e Aplicações*. Campus.

[Tanenbaum, 2003] Tanenbaum, A. (2003). *Sistemas Operacionais Modernos, 2ª edição*. Pearson – Prentice-Hall.

[Uhlig and Mudge, 1997] Uhlig, R. and Mudge, T. (1997). Trace-driven memory simulation: a survey. *ACM Computing Surveys*, 29(2):128–170.