Referências Bibliográficas

[1] BRENNER, N. M.. **Fast fourier transform of externally stored data**. IEEE Transactions on Audio and Electroacoustics, AU-17(2):128–132, 1969.

[2] WILLIAMS, L.. **Pyramidal parametrics**. SIGGRAPH, 17:1–11, July 1983.

[3] FANT, K. M.. **A noaliasing, real-time spatial transform tecnique**. IEEE Computer Graphics and Applications, p. 71–80, – 1986.

[4] TANENBAUM, A.. **Operating Systems: Design and Implementation**. Prentice-Hall, 1987.

[5] FUNKHOUSER, T. A.; SEQUIN, C. H. ; TELLER, S. J.. **Management of large amounts of data in interactive building walkthroughs**. Computer Graphics 1992 Symposium on Interactive 3D Graphics, 25(2):11–20, 1992.

[6] FALBY, J. S.; ZYDA, M. J.; PRATT, D. R. ; MACKEY, R. L.. **NPSNET: Hierarchical data structures for real-time three-dimensional visual simulation**. Computers and Graphics, 17(1):65–69, – 1993.

[7] FUNKHOUSER, T. A.; SEQUIN, C. H.. **Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments**. Computer Graphics, 27(Annual Conference Series):247–254, 1993.

[8] HECKBERT, P. S.; GARLAND, M.. **Fast polygonal approximation of terrains and height fields**. Technical report, School of Computer Science, 1995.

[9] GOMES, J.; VELHO, L.. **Abstraction paradigms for computer graphics**. The Visual Computer, 11:227–239, – 1995.

[10] LINDSTROM, P.; KOLLER, D.; RIBARSKY, W.; HODGES, L. F. ; FAUST, N.. **Real-time, continuous level of detail rendering of height fields**. Proceeedings of ACM SIGGRAPH, p. 109–118, August 1996.

[11] HOPPE, H.. **Progressive meshes**. ACM SIGGRAPH, p. 99–108, August 1996.

[12] WALKER, CHRIS, N. C. J. B.; KESLIN, P.. **Image Vision Library 3.0 Programming Guide**. Silicon Graphics Computer Systems,, 1996.

[13] SCHMALSTIEG, D.; GERVAUTZ, M.. **Demand-driven geometry transmission for distributed virtual environments**. Technical report, Institute of Computer Graphics and Algorithms, Vienna University of Technology, January 1996.

[14] DUCHAINEAU, M.; WOLINSKY, M.; SIGELI, D. E.; MILLER, M. C.; ALDRICH, C. ; MINEEV-WEINSTEIN, M. B.. **Roaming terrain: Real-time optimally adapting meshes**. IEEE Visualization, p. 81–88, November 1997.

[15] HECKBERT, P. S.; GARLAND, M.. **Surface simplification using quadric error metrics**. Proceeedings of ACM SIGGRAPH, August 1997.

[16] HOPPE, H.. **View-dependent refinement of progressive meshes**. Computer Graphics, 31(Annual Conference Series):189–198, 1997.

[17] HECKBERT, P. S.; GARLAND, M.. **Survey of polygonal surface simplification algorithms**. Technical report, School of Computer Science, 1997.

[18] RICHTER, J.. **Advanced Windows**. Microsoft Press, third edition, 1997.

[19] GOMES, J.; VELHO, L.. **Image Processing for Computer Graphics**. Springer-Verlag, 1997.

[20] WOO, M.; NEIDER, J. ; DAVIS, T.. **OpenGL Programming Guide. The Official Guide to Learning OpenGL, version 1.1**. Addison-Wesley Logman, second edition, 1997.

[21] GOMES, J.; DARSA, L.; COSTA, B. ; VELHO, L.. **Warping and Morphing of Graphical Objects**. Morgan Kaufmann, 1998.

[22] MALLAT, S.. **A wavelet tour of signal processing**. Academic Press, 1998.

[23] PAJAROLA, R. B.. **Large scale terrain visualization using the restricted quadtree triangulation**. In: Ebert, D.; Hagen, H. ; Rushmeie, H., editors, IEEE VISUALIZATION '98, p. 19–26, 1998.

[24] HOPPE, H.. **Smooth view-dependent level-of-detail control and its application to terrain rendering**. In IEEE Visualization '98, p. 35–42, October 1998.

[25] MATOS, A.. **Visualização de panoramas virtuais**. Master's thesis, PUC-Rio, 1998.

[26] CHISTOPHER, T.; CHRISTOPHER, M. ; MICHAEL, J.. **The clipmap: A virtual mipmap**. SIGGRAF, p. 151–158, July 1998.

[27] HESINA, G.; SCHMALSTIEG, D.. **A network architecture for remote rendering**. Technical report, Institute of Computer Graphics and Algorithms, Vienna University of Technology, April 1998.

[28] MATOS, A.; GOMES, J. ; VELHO, L.. **Cache management for real time visualization of 2d data sets**. SIBGRAPI, p. 111–118, 1998.

[29] GARLAND, M.. **Multi-resolution modeling: Survey & future opportunities**. Eurographics'99, September 1999. State of the Art Report.

[30] COELHO, E. M.. **Visualização interativa de terrenos em um arquitetura cliente-servidor**. Master's thesis, PUC-Rio, 1999.

[31] HOFMANN, J. S.. **A survey of real-time algorithms**. Technical report, CSci 361 Fall 2000, 2000.

[32] BLOW, J.. **Terrain rendering at high levels of detail**. Proceedings of the 2000 Game Developers Conference, March 2000.

[33] VELHO, L.; GOMES, J.. **Variable resolution 4-k meshes: Concepts and applications**. Computer Graphics Forum, 2000.

[34] DÖLLNER, J.; BAUMAN, K. ; HINRIGHS, K.. **Texturing techniques for terrain visualization**. IEEE Visualization 2000, p. 227–234, october 2000.

[35] SILBERSCHATZ, A.; GALVIN, P.. **Sistemas Operacionais**. Prentice Hall, 2000.

[36] LINDSTROM, P.. **Out-of-core simplification of large polygonal models**. Proceeedings of ACM SIGGRAPH, p. 259–262, July 2000.

[37] CAPPS, M. V.. **The quick framework for task-specific asset prioritization in distributed virtual environments**. Proceedings of the IEEE Virtual Reality, 2000.

[38] VITTER, J. S.. **External memory algorithms and data structures dealing with massive data**. ACM Computing Surveys, (2), June 2001.

[39] YOUBING, Z.; JI, Z.; JIAOYING, S. ; ZHIGENG, P.. **A fast algorithm for large scale terrain walkthrough**. CAD/Graphics 2001, August 2001.

[40] LINDSTROM, P.; PASCUCCI, V.. **Visualization of large terrains made easy**. Proceeedings of IEEE Visualization, p. 363–370, October 2001.

[41] PARK, S.; LEE, D.; LIM, M. ; YU, C.. **Scalable data management using user-based caching and prefetching in distributed virtual environments**. ACM VRST 2001, 2001.

[42] LINDSTROM, P.; PASCUCCI, V.. **Terrain simplification: A general framework for view-dependent out-of-core visualization**. IEEE Transaction on Visualization and Computer Graphics, p. 239–254, May 2002.

[43] KEYHOLE. **Earthviewer**. http://www.earthviewer.com, 2002.

[44] MICROSOFT. **Terraserver**. http://terraserver-usa.com, 2003.

A
Sistema de Memória Virtual

A.1
Memória Virtual

Em geral, os sistemas computacionais são formados basicamente por uma memória primária, uma ou várias memórias secundárias e uma unidade de processamento de dados(CPU). A memória principal e os registradores são os únicos dispositivos de armazenamento que podem ser acessados diretamente pela CPU. A memória principal tem uma pequena capacidade de armazenamento e altas taxas de transferência de dados. A memória secundária é uma extensão da memória principal, este tipo de dispositivo tem o objetivo de armazenar permanentemente todos os dados contidos no sistema. A memória secundária tem grande capacidade de armazenamento e pequena taxas de transferência de dados.

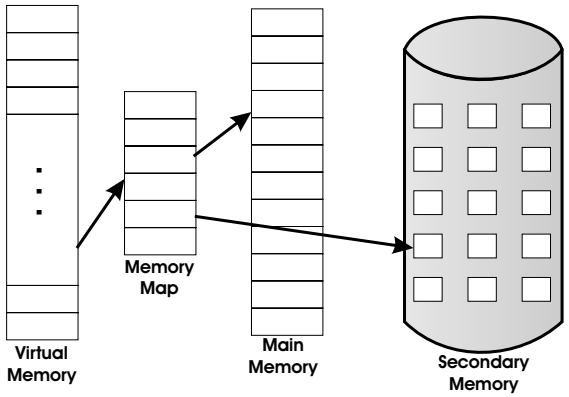

Figura A.1: Estrutura básica de um sistema de memória virtual.

Para serem processados, os dados devem ser armazenados na memória principal. Em geral, a quantidade de dados a ser processada supera a capacidade de armazenamento da memória principal. Desta forma, é necessário desenvolver um mecanismo de gerenciamento de memória que permita que uma grande quantidade de dados seja processada e apesar do

pequeno poder de armazenamento da memória principal. Estas técnicas de gerenciamento baseiam-se no princípio de que quantidade de dados que deve ser processada simultaneamente para obter um determinado resultado pode ser armazenada na memória principal. Assim, surgiram vários sistemas de gerenciamento de memória[35][4]. Neste trabalho, o sistema de interesse é o sistema de memória virtual.

O sistema de memória virtual foi desenvolvido para permitir que os sistemas operacionais pudessem executar vários processos sem a necessidade de carregá-los totalmente para a memória. Uma das principais características desse sistema é permitir que vários processos possam utilizar uma área maior de memória do que a disponível na memória física. Uma outra característica é que esse sistema de memória cria um modelo de abstração da memória que separa completamente o nível de memória lógica, que é o nível que o usuário lida, com o nível de memória física. A Figura A.1 mostra a estrutura de um sistema de memória virtual.

As duas principais características desse modelo de memória, isto é, o carregamento parcial dos programas e separação dos nívels de memórias, trousse vários benefícios:

- Como os programas podem utilizar um espaço de endereçamento virtual muito grande, suas áreas de memórias não ficam mais limitadas ao tamanho da memória disponível.
- Como o sistema não precisa carregar todo o programa para a memória, várias aplicações podem ser executadas simultaneamente.
- Devido a separação dos níveis de memórias, o programador não precisa mais se preocupar com o tamanho da memória física disponível, podendo concentrar-se totalmente no desenvolvimento da aplicação para resolver o problemas em questão.

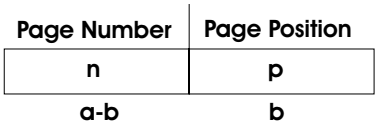

Figura A.2: Representação de um endereço lógico. O sistema de endereçamento lógico identifica unicamente cada unidade de memória.

No sistema de memória virtual, cada processo é associado a uma tabela de mapeamento ou mapa de memória. Este sistema utiliza o mecanismo de paginação para gerenciar o processo de troca de dados entre as memórias principal e secundária. No mecanismo de paginação, as memórias principal

e secundária são divididas em unidades de mesmo tamanho chamadas de *blocos*. A memória lógica é dividida em partes de mesmo tamanho dos blocos chamadas de *páginas* e o mapa de memória agora é chamado de *tabela de páginas*, como mostra a Figura A.3. Considerando esta forma de divisão da memória, um endereço lógico pode ser expresso por um número de página e uma posição nesta página. Supondo que o tamanho do espaço de endereçamento lógico é de 2^a, e o tamanho da página é de 2^b unidades de endereçamento (bytes ou palavras), então os $a - b$ bits mais á esquerda de um endereço lógico significam um número de página, e os b bits mais a esquerda significam a posição na página, veja a Figura A.2.

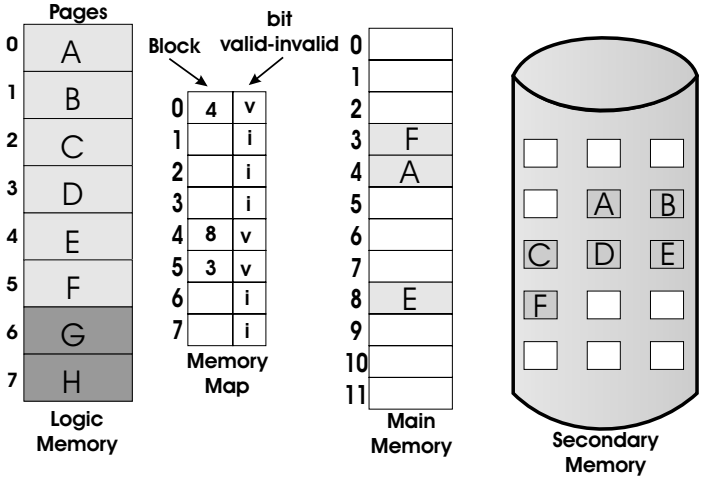

Figura A.3: Mecanismo de paginação sob demanda.

A tabela de páginas é representada por um vetor onde um elemento i contém informações relativas a uma página i do processo. Cada elemento deste vetor é formatado por dois campos. Um campo informa se uma página é válida ou inválida. Uma página válida significa que esta página está armazenada na memória principal e uma página inválida significa que esta página não está armazenada na memória principal ou que esta página não pertence ao espaço de endereçamento lógico do processo. O outro campo informa o bloco da memória principal que está associado com a página válida. A Figura A.3 mostra a estrutura da tabela de páginas.

A.2 Mecanismo de Paginação

Os sistemas operacionais implementam o mecanismo de paginação sob demanda, ou seja, uma página nunca é copiada para a memória principal a

menos que os dados armazenados nesta página sejam necessários. Quando um processo é executado o sistema operacional realiza duas tarefas. A primeira é criar uma tabela de páginas para o processo, existem várias formas de estruturar e armazenar uma tabela de páginas, veja [35] para mais detalhes. A segunda tarefa é procurar quais são as páginas que esse processo irá utilizar, assim apenas as páginas necessárias é que serão carregadas na memória principal.

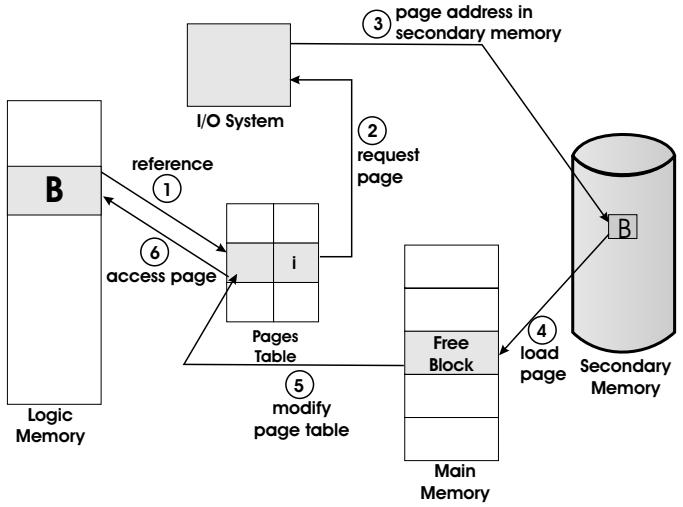

Figura A.4: Tarefas realizadas pelo sistema de gerenciamento de memória quando um processo acessa um página ausente.

O procedimento realizado pela técnica de paginação quando um processo faz acessos à memória virtual, pode ser dividido em seis passos, como mostra a Figura A.4. As tarefas realizadas nesses seis passos são descritas logo abaixo:

1. A tabela de páginas do processo é verificada para determinar se este processo está acessando uma página válida ou inválida. Caso a página referenciada seja válida, o sistema passa para o passo 6.

2. Se a referência for inválida, então se o processo esta acessando uma área de memória que não lhe pertence, este processo é terminado. Caso contrário, inicia-se o processo de transferência da página requerida que está armazenada na memória secundária para a memória principal.

3. Procura-se um bloco na memória principal que esteja livre.

4. Faz-se uma requisição para a realização de uma operação de leitura da página desejada na memória secundária para o bloco alocado no passo anterior.

5. Quando a operação de leitura termina, a tabela de páginas do processo é modificada para que esta, agora, indique que a página está válida, isto é, armazenada na memória principal.

6. O endereço lógico, referenciado pelo processo, é traduzido para um endereço físico. Após a tradução, o processo pode realizar as operações desejadas sobre os dados que estão armazenados neste endereço físico.

Se existem muitos processos sendo executados simultaneamente ou o se espaço memória principal é muito restrito, a tarefa realizada no passo 3 pode falhar, isto é, não encontrar nenhum bloco livre na memória principal. Quando isto ocorre, é necessário realizar uma operação de substituição de página antes de executar os outros passos. A operação de substituição de páginas consiste em selecionar e liberar um bloco da memória principal que não esteja sendo utilizado para dar lugar a página que se deseja carregar. Esta operação é realizada seguindo os seguintes passos:

1. Obter a posição, na memória secundária, da página que se deseja carregar.

2. Obter um bloco livre na memória principal:

   a Se já existe um bloco livre, ele é usado. Assim, passa-se diretamente para o passo 4.

   b Caso contrário, utiliza-se um algoritmo de substituição de páginas para selecionar um bloco cuja a página será substituída.

   c Se os dados contidos no bloco foram modificados, a página a ser substituída é gravada na memória secundária. Em seguida, as tabelas de páginas são utilizadas de acordo.

3. A página é transferida para o bloco que, agora, está livre. Em seguida, as tabelas de páginas são novamente atualizadas.

4. O processo continua sua execução.

A Figura A.5 mostra as fases de realizadas pelo processo de substituição de página.

A eficiência de execução dos processos está intimamente ligada com a eficiência do sistema de paginação sob demanda. Isto é, quanto mais tempo um processo levar para obter os dados desejados, mais tempo será gasto para se concluir uma operação que depende desses dados.

Para implementar um sistema de paginação sob demanda que tenha uma boa eficiência é necessário resolver basicamente dois problemas. O

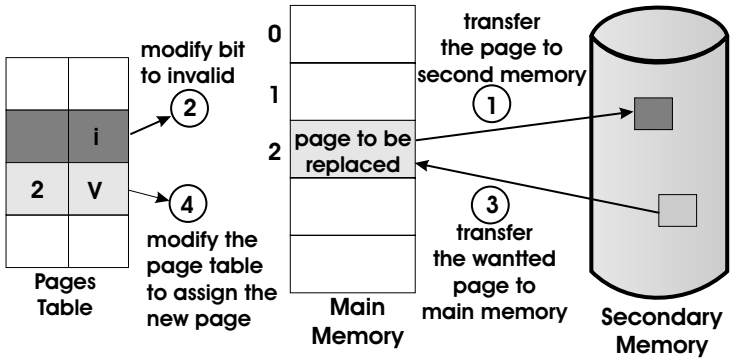

Figura A.5: Tarefas realizadas pelo sistema de gerenciamento de memória durante o processo de substituição de página.

primeiro é definir um algoritmo de alocação de blocos. Este algoritmo tem a função de determinar quantos blocos se deve alocar para cada processo no momento que eles são iniciados. O segundo problema é definir um algoritmo de substituição de páginas. Esses algoritmos são desenvolvidos como o objetivo de minimizar as operações de acesso a memória secundária, visto que a taxa de transferência nesse tipo de dispositivo é baixa. Assim, pequenas melhorias nesses algoritmos podem resultar em grandes ganhos para o desempenho do sistema. Existem vários algoritmos de alocação blocos e substituição de páginas, os mais clássicos são descritos em [35][4].