Sistemas Operacionais

Introdução a Arquitetura

Parte 04

“Gerência de Memória”

Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro
Prof. Edwar Saliba Júnior
Janeiro de 2026

Nota

- Com exceção das figuras que possuem referência bibliográfica, todo o restante do conteúdo destes *slides* foram extraídos do livro dos autores “Francis Berenger Machado” e “Luiz Paulo Maia”. Todos referenciados no último *slide* desta apresentação.

Gerência de Memória

- A memória principal sempre foi vista como um recurso escasso e caro. Então, uma das maiores preocupações dos projetistas foi desenvolver sistemas operacionais que não ocupassem muito espaço de memória e que otimizassem a utilização dos recursos computacionais.

- Mesmo atualmente, com a redução de custo e consequente aumento da capacidade da memória principal, seu gerenciamento é um dos fatores mais importantes no projeto de sistemas operacionais.

Gerência de Memória

- Sistemas **monoprogramáveis**:
  - a gerência da memória não é muito complexa;
- sistemas **multiprogramáveis**:
  - essa gerência se torna crítica, devido à necessidade de se maximizar o número de usuários e aplicações utilizando eficientemente o espaço da memória principal.

Gerência de Memória

- Em um ambiente de multiprogramação, o sistema operacional deve proteger as áreas de memória ocupadas por cada processo, além da área onde reside o próprio sistema.

- Caso um programa tente realizar algum acesso indevido à memória, o sistema operacional deve impedi-lo.

Alocação Contígua Simples

- Foi implementada nos primeiros sistemas operacionais, porém ainda está presente em alguns sistemas *monoprogramáveis*.

- Nesse tipo de organização, a memória principal é subdividida em duas áreas:
  - uma para o sistema operacional e
  - outra para o programa do usuário.

Alocação Contígua Simples

Técnica de *Overlay*

- Na alocação contígua simples, todos os programas estão limitados ao tamanho da área de memória principal disponível para o usuário.

- Para que caiba na memória, geralmente o programa é dividido em módulos. De forma que seja possível a execução independente de cada módulo, utilizando uma mesma área de memória. Técnica de *overlay* ou técnica de sobreposição.

- Considere um programa que tenha três módulos:
  - um principal,
  - um de cadastramento e
  - outro de impressão,

  sendo os módulos de cadastramento e de impressão independentes. A independência do código significa que quando um módulo estiver na memória para execução, o outro não precisa estar presente.

- O módulo principal é comum aos dois módulos. Logo, deve permanecer na memória durante todo o tempo da execução do programa.

Técnica de *Overlay*

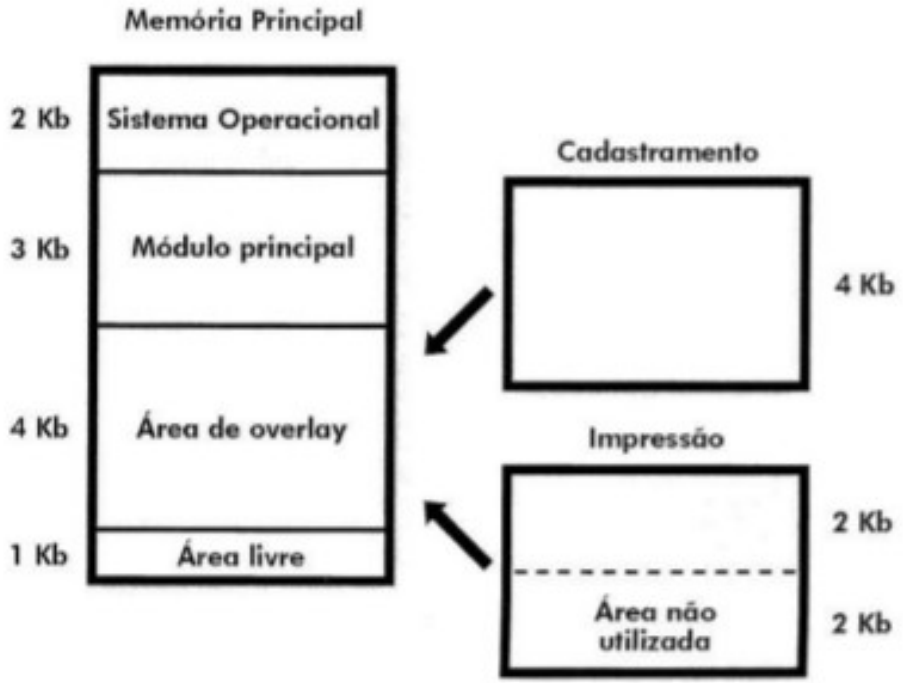

Figura 9.4: Técnica de overlay

Alocação Particionada Estática

- Nos primeiros sistemas multiprogramáveis, a memória era dividida em pedaços de tamanho fixo, chamados partições.
- O tamanho das partições, estabelecido na fase de inicialização do sistema, era definido em função do tamanho dos programas que executariam no ambiente e
- sempre que fosse necessária a alteração do tamanho de uma partição, o sistema deveria ser desativado e reinicializado com uma nova configuração.
- Problema: fragmentação interna.

Alocação Particionada Estática

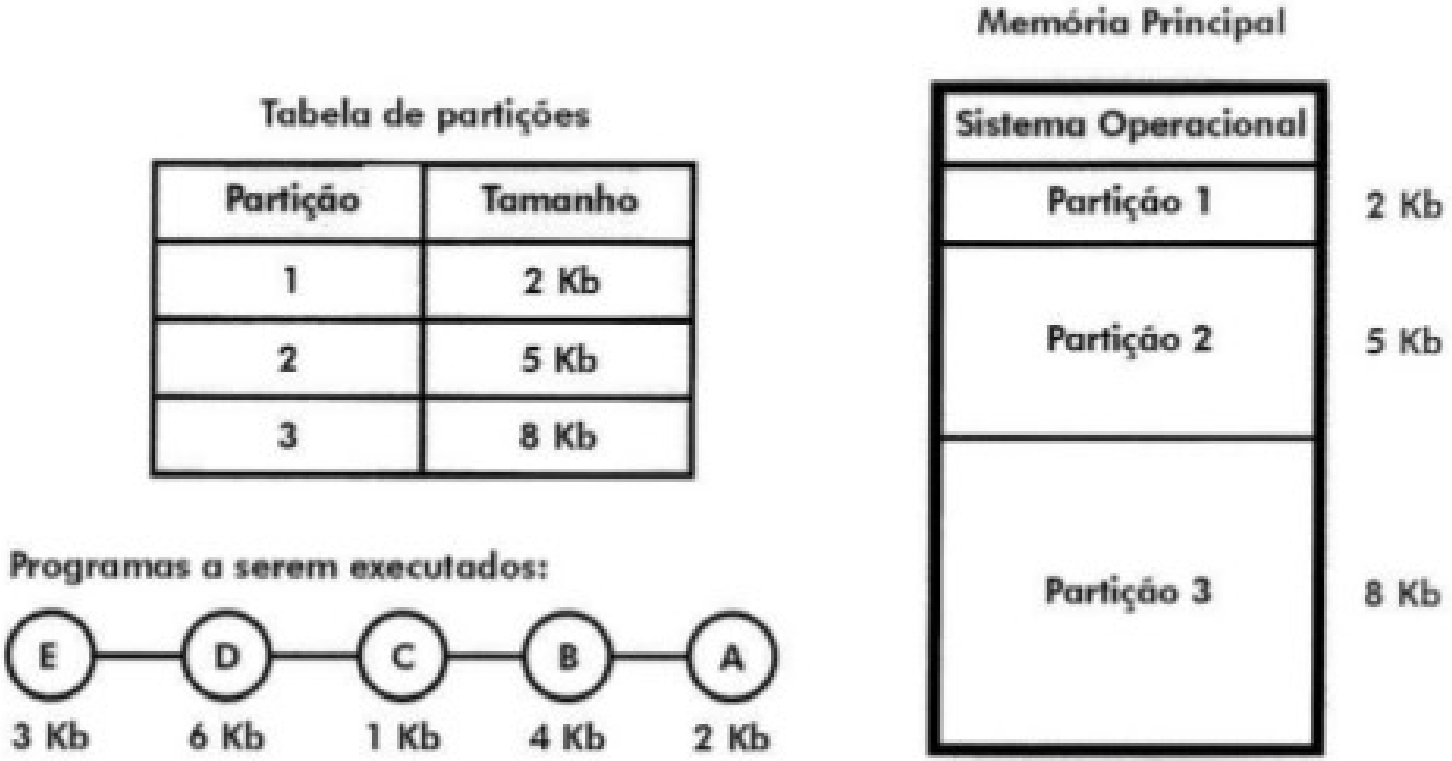

Figura 9.9: Alocação particionada estática

Alocação Particionada Estática

Problema:

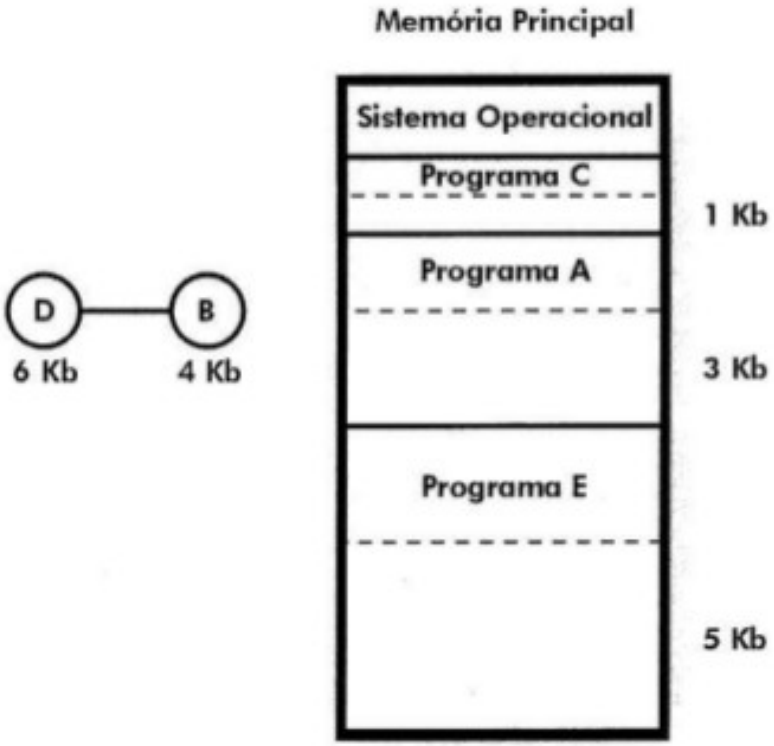

Figura 9.14: Fragmentação interna

Alocação Particionada Dinâmica

- Nesse esquema cada programa utiliza o espaço necessário, tornando essa área sua partição.

- Como os programas utilizam apenas o espaço de que necessitam, no esquema de alocação particionada dinâmica o problema da fragmentação interna não ocorre.

- Problema: fragmentação externa.

Alocação Particionada Dinâmica

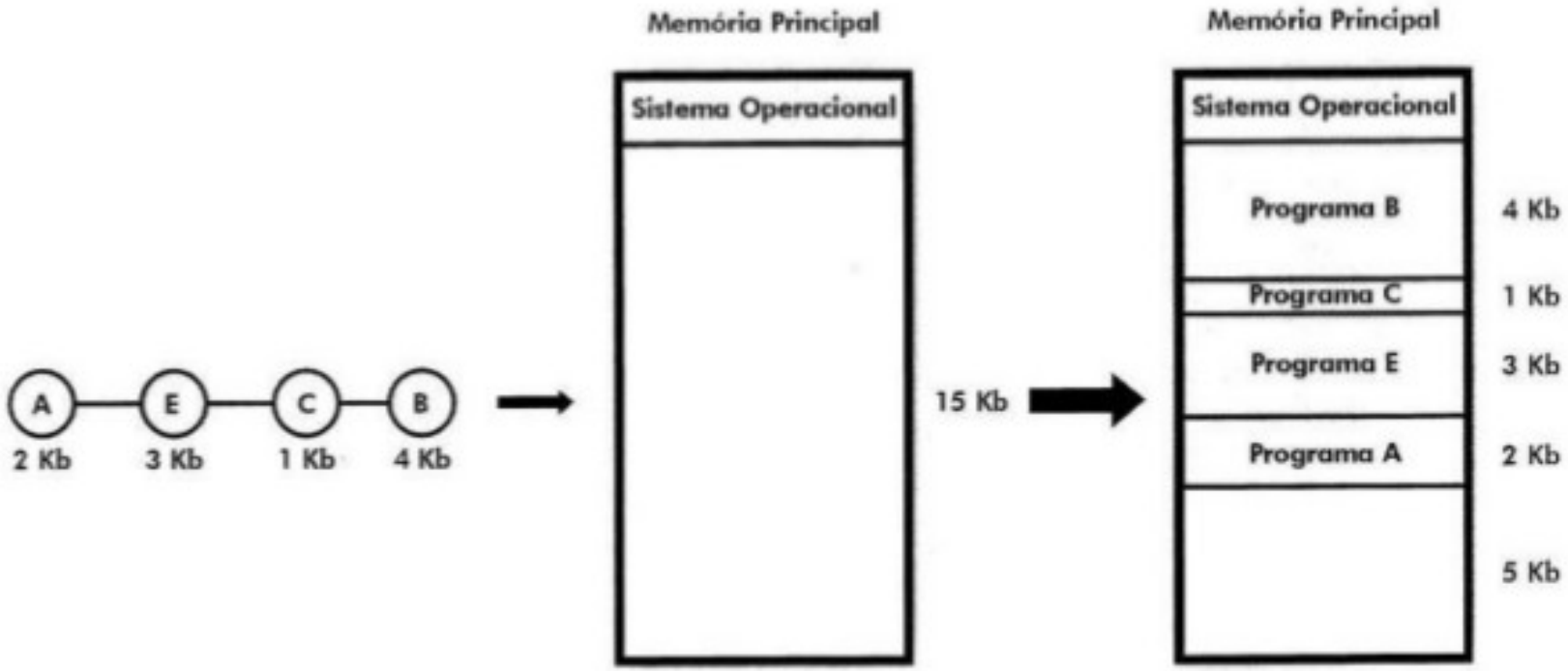

Figura 9.15: Alocação particionada dinâmica

Alocação Particionada Dinâmica

Problema:

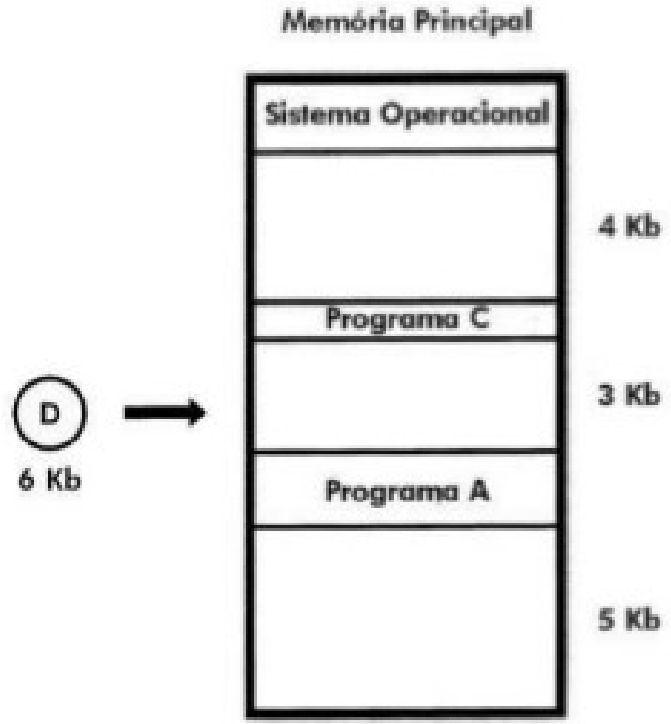

Figura 9.16: Fragmentação externa

Swapping

- Mesmo com o aumento da eficiência da multiprogramação e da gerência de memória, muitas vezes um programa não podia ser executado por falta de uma partição livre disponível.

- Então, a técnica de *swapping* foi introduzida para contornar este problema.

Swapping

- Nos esquemas apresentados anteriormente, um processo permanece na memória principal até o fim de sua execução, inclusive nos momentos em que espera por um evento, como uma operação de leitura ou gravação em disco.

- O *swapping* é uma técnica aplicada à gerência de memória para programas que esperam por memória livre para serem executados. O sistema escolhe um processo residente, que é transferido da memória principal para a memória secundária (*swap out*), geralmente disco e

- posteriormente, o processo é carregado de volta da memória secundária para a memória principal (*swap in*) e pode continuar sua execução como se nada tivesse ocorrido.

Swapping

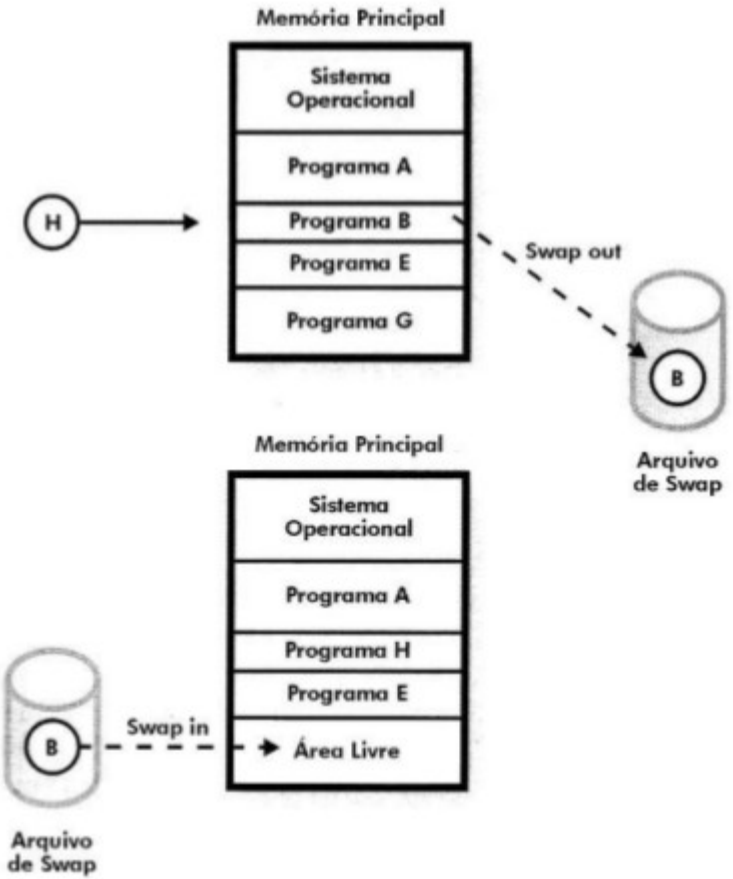

Figura 9.21: Swapping

Memória Virtual

- Técnica sofisticada e poderosa de gerência de memória, onde as memórias principal e secundária são combinadas. Esta combinação proporciona ao usuário a ilusão de existir uma memória principal muito maior do que a capacidade real existente na máquina.

- O conceito de memória virtual fundamenta-se em não vincular o endereçamento feito pelo programa aos endereços físicos da memória principal. Desta forma, programas e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível, pois podem possuir endereços associados à memória secundária.

Referências

- ERDEI, Jacob. **Basic Structure for a Pentium Microprocessor**. Disponível em: <https://www.pctechguide.com/cpu-architecture/basic-structure-of-a-pentium-microprocessor>. Acesso em: 13 jul. 2017.
- MACHADO, F. B.; MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 3. ed. Rio de Janeiro: LTC, 2010.
- PANATTA, B.; SANTI, D. de; MOERSCHBACHER, G.; LIMA, L. G. de. **Esquema Geral de Funcionamento do Processador**. Disponível em: <http://sca.unioeste-foz.br/~habib/x/trabalhos/grupoa4/public_html/processador.html>. Acesso em: 14 jul. 2017.
- UNIVERSIDADE FEDERAL FLUMINENSE. **Organização de Computadores II - processadores**. Disponível em: <http://orgcomp2.ic.uff.br/processadores.php>. Acesso em: 14 jul. 2017.