CURSO SUPERIOR EM TECNOLOGIA EM REDES DE COMPUTADORES

SISTEMAS OPERACIONAIS

Aula 06

Gerencia de Memória

PROFESSOR ANTÔNIO ROGÉRIO MACHADO RAMOS

Avaliação 2 e recuperação da avaliação 1
na aula 10

GERÊNCIA DE MEMÓRIA

A memória pode ser vista como um vetor de elementos, onde cada elemento representa a menor unidade de informação que pode ser lida ou gravada.

Geralmente esse elemento é o Byte e ele é referenciado por um endereço que pode ser acessado diretamente, ou seja, não é necessário passar por endereços intermediários para se chegar ao endereço desejado.

end	valor
0	200
1	50
2	150
3	300
4	30
5	500
6	100
...	
n	

TIPOS DE MEMÓRIA

MEMÓRIA FÍSICA
Memória de hardware, representada geralmente pela RAM. Também conhecida como memória principal.

MEMÓRIA VIRTUAL
Representada pela memória persistente, sendo que geralmente se usa o HDD. também conhecida como memória secundária.

MEMÓRIA LÓGICA
Acessada pelos processos. É uma abstração da memória física com a memória virtual.
Esta memória é representada pelas páginas e segmentos (vistos adiante) e seu tamanho é determinado pela capacidade do S.O. e pelas limitações do hardware.

ROTINAS

Controle do espaço de memória alocado para determinar o que está ocupado e o que está livre.

Alocação de memória de acordo com a demanda dos processos, considerando que existe a área ocupada pelo processo pesado (instruções) e a área dos dados utilizados para o processamento.

Liberação da memória ao findar o processamento.

Transferência do processo, ou parte dele, entre a memória principal e a memória secundária.

MÁQUINA PURA

O controle da memória é total e direto.

O S.O. não gerencia a memória.

O desenvolvimento é mais flexível, porque o acesso da memória é direto.

A falta de gerência por parte do S.O. torna o desenvolvimento mais complexo, exigindo um conhecimento maior do desenvolvedor.

Empregado no desenvolvimento para computadores mais simples e dedicados, que não possuem S.O.

Os programas para estas máquinas geralmente são desenvolvidos em computadores em uma linguagem fonte conhecida, como o C, e submetidos a um cross compiler (compilador cruzado) para gerar o código executável para a máquina dedicada.

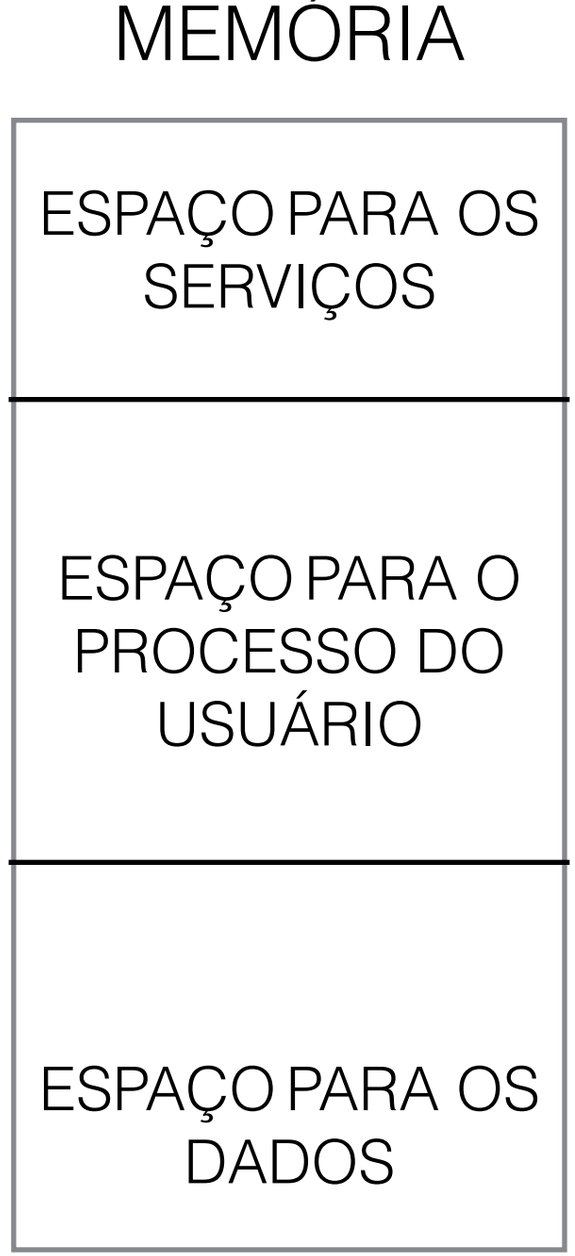
MONOPROGRAMADOS

Espaço de memória dividido entre o processo, os serviços e a área para dados utilizados pelos serviços e pelo processo.

Fácil de implementar, uma vez que o usuário não precisa conhecer o hardware, apenas deve usar o serviço correspondente ao dispositivo desejado.

Não existe overhead porque o S.O. é apenas um conjunto de serviços utilizados pelo processo.

Não existe controle do S.O. sobre o processo. Se o processo fizer um erro, este pode travar todo o sistema. O máximo que pode ser gerado é uma trap que pode desviar o controle do processo para o monitor residente.



ALOCAÇÃO ESTÁTICA

O código executável é gerado pelo compilador. Este código inicia em um endereço e tem um tamanho expresso em bytes.

Tal código é carregado pelo serviço do S.O. chamado loader (carregador) na memória na posição de memória especificada no código.

Se o código for carregado em outro endereço, as instruções de salto, presentes no código, vão indicar endereços inválidos do código, gerando erro de execução.

PROCESSO - endereço inicial(10) - tamanho(6)

```
10 mov ax,0     move 0 para o reg. ax
11 add ax,1     adiciona 1 ao reg. ax
12 cmp ax,9     retorna V se ax==9
13 jne 11       se for F, salta para 11
14 mov bx,10    seleciona o serviço 10
15 int 21       aciona o vídeo
```

Se o processo acima não for carregado na memória a partir do endereço 10, o comando do endereço 13 vai realizar um salto para uma posição inválida do código.

Esse tipo de processo não pode ser realocado na memória.

RELOCAÇÃO ESTÁTICA

Nos antigos mainframes, o compilador utilizava o valor fornecido pelo monitor residente do endereço inicial onde o processo deve ser carregado.

Com esta informação, o compilador recalculava os endereços de salto na geração do código executável e gerava um código para ocupar a partir da posição de memória inicial fornecida pelo monitor.

Desta forma, o programa executável poderia ser alocado em uma posição de memória diferente.

PROCESSO - endereço inicial(10) - tamanho(6)

```
10 mov ax,0     move 0 para o reg. ax
11 add ax,1     adiciona 1 ao reg. ax
12 cmp ax,9     retorna V se ax==9
13 jne 11       se for F, salta para 11
14 mov bx,10    seleciona o serviço 10
15 int 21       aciona o vídeo
```

O endereço inicial, neste caso, foi fornecido pelo S.O.

A cada nova criação do processo, ele pode ser carregado em uma região diferente da memória, tornando a alocação mais flexível.

RELOCAÇÃO DINÂMICA

O programa executável é construído considerando um endereço relativo de memória iniciando em 0.

O loader recebe o endereço inicial da memória referente a uma posição disponível.

Ao carregar na memória, os endereços de alto são recalculados conforme o endereço inicial informado.

O processo pode ocupar qualquer posição disponível na memória com tamanho suficiente.

PROCESSO COM ENDEREÇO INICIANDO EM 0

```
00  mov ax,0      move 0 para o reg. ax
01  add ax,1      adiciona 1 ao reg. ax
02  cmp ax,9      retorna V se ax==9
03  jne 01        se for F, salta para 01
04  mov bx,10     seleciona o serviço 10
05  int 21        aciona o vídeo
```

LOADER (endereço inicial(10), tamanho(6))

```
10  mov ax,0
11  add ax,1
12  cmp ax,9
13  jne 11
14  mov bx,10
15  int 21
```

Endereço de salto recalculado de 01 para 11. É adicionado o endereço inicial 10.

RELOCAÇÃO DINÂMICA

Esta modalidade de relocação é semelhante a anterior.

O diferencial é que os endereços de salto não são calculados no momento da carga, mas durante a execução do processo.

Com isso, o processo pode ser relocado na memória durante a execução, deslocando-se pela memória para otimizar o espaço.

Apesar de gerar um overhead, o algoritmo está bem otimizado e o processo é bem realizado.

PROCESSO COM ENDEREÇO INICIANDO EM 0

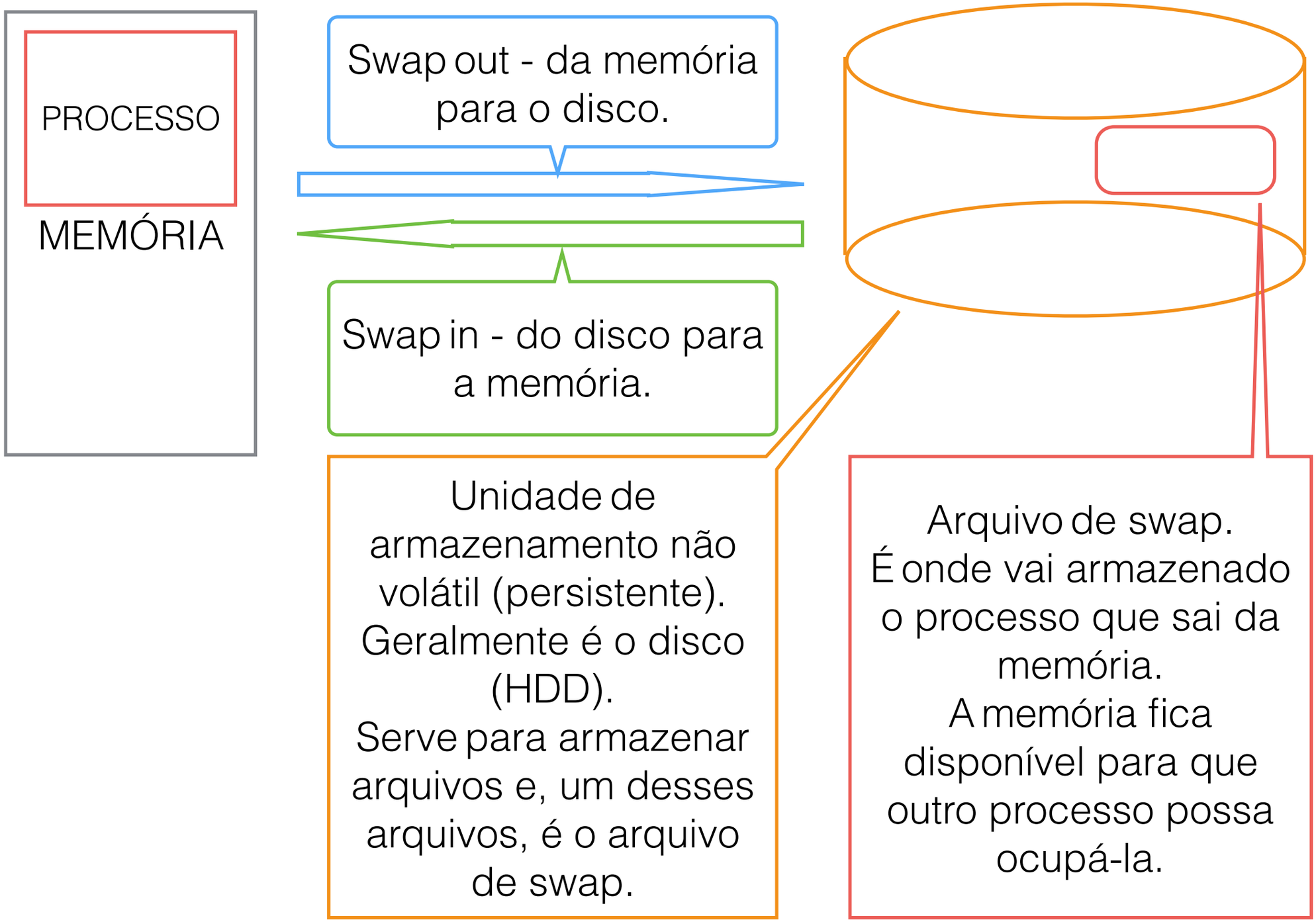
```
00  mov ax,0      move 0 para o reg. ax
01  add ax,1      adiciona 1 ao reg. ax
02  cmp ax,9      retorna V se ax==9
03  jne 01        se for F, salta para 01
04  mov bx,10     seleciona o serviço 10
05  int 21        aciona o vídeo
```

Endereço inicial dinâmico(10), tamanho(6)

```
10  mov ax,0
11  add ax,1
12  cmp ax,9
13  jne 11
14  mov bx,10
15  int 21
```

Endereço de salto recalculado de 01 para 11 em tempo de execução.

SWAPPING


PROCESSO
MEMÓRIA
Swap out - da memória para o disco.
Swap in - do disco para a memória.
Unidade de armazenamento não volátil (persistente). Geralmente é o disco (HDD).
Serve para armazenar arquivos e, um desses arquivos, é o arquivo de swap.
Arquivo de swap.
É onde vai armazenado o processo que sai da memória.
A memória fica disponível para que outro processo possa ocupá-la.

PARTIÇÕES MÚLTIPLAS

A memória RAM aumentou sua capacidade nos computadores a ponto de permitir que vários processos possam ocupá-la. Mesmo assim, os processos ficaram maiores a ponto de serem divididos em partes que podem ficar na memória e também no disco (HDD, etc.).

Os computadores também aumentaram sua capacidade de processamento e os discos passaram a ser usados não só para swapping mas também para armazenamento de dados. O arquivo de swap não armazena mais processos inteiros, mas também partes de processos maiores que vão para memória quando são necessários.

O ideal é que o computador tenha a memória RAM com tamanho suficiente para comportar todos os processos em execução. Se o tamanho for insuficiente, será feito mais swap, gerando um overhead que compromete o desempenho da máquina.

PARTIÇÃO FIXA

São partições divididas em grupos de tamanhos diferentes.
Esses tamanhos são definidos na carga do S.O. e não são modificados.
Os processos são alocados nas partições de acordo com os seus tamanhos.

100KB	100KB	100KB	200KB	200KB	200KB

ALOCAÇÃO LOCAL
Cada grupo de partições tem uma fila de processos com tamanho igual ou menor que a partição.

100KB	100KB	100KB	FILA DE PROCESSOS COM TAM<=100KB

200KB	200KB	200KB	FILA DE PROC TAM<=200KB

ALOCAÇÃO GLOBAL
Uma única fila de processos para todas as partições. Se um processo é maior do que a partição disponível, ele preempta (cede a vez) para o próximo processo tentar a alocação.

100KB	100KB	100KB

200KB	200KB	200KB

FILA ÚNICA DE PROCESSOS PARA TODAS AS PARTIÇÕES

FRAGMENTAÇÃO NA PARTIÇÃO FIXA

O processo ocupa uma partição de tamanho maior ou igual ao tamanho dele. Quando o tamanho é maior, é gerada uma fragmentação interna.

O S.O. possui um registro da posição na memória e o status de cada partição fixa.

FRAGMENTAÇÃO INTERNA - É o espaço que sobra dentro da partição e que não pode ser ocupado por nenhum outro processo.

Existem ainda as partições que não são ocupadas porque não existe nenhum processo do mesmo tamanho ou menor do que ele.

O processo também pode ocupar uma partição de tamanho igual, não gerando fragmentação interna. Geralmente, as partições são dimensionadas considerando antecipadamente o tamanho dos processos que são executados no S.O. Em um S.O. moderno, as partições fixas são utilizadas para acomodar páginas de processos.

PARTIÇÃO VARIÁVEL

São partições dimensionadas de acordo com o processo que vai ocupar a memória. Desta forma, é possível que existam partições com tamanhos distintos espalhadas pela memória.

100KB	60KB	140KB	200KB	400KB

Neste tipo de partição não existe fragmentação interna porque todas as partições são definidas do mesmo tamanho do processo, no momento da carga do processo.

100KB	100KB	100KB	ESPAÇO DE MEMÓRIA DISPONÍVEL: 500KB
100KB	DISP	100KB	200KB	300KB

Quando um processo termina, sua partição é desfeita e a memória tem o espaço liberado.

100KB	200KB	100KB	300KB	50KB	200KB
100KB	200KB	DISP	300KB	50KB	200KB
100KB	200KB	DISP		50KB	200KB

PARTIÇÃO VARIÁVEL

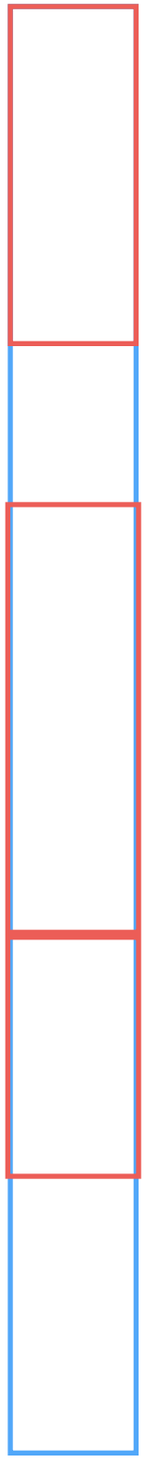
Quando os espaços liberados são adjacentes, eles naturalmente se agrupam formando um espaço maior. Caso não sejam adjacentes, se faz necessário um processamento para juntá-los, deslocando os processos na memória.

100KB	60KB	140KB	200KB	400KB
100KB	DISP	140KB	200KB	400KB
100KB	DISP	140KB	DISP	400KB
100KB	DISP	DISP	140KB	400KB
100KB	DISP		140KB	400KB

Essa desfragmentação de memória gera um overhead. Por isso, geralmente ela é feita por um serviço que opera apenas no modo idle (quando a máquina está ociosa).

Apenas processos que possuem relocação dinâmica (o endereço de salto é calculado em tempo de execução) podem ser deslocados pela memória. Processos com relocação estática inviabilizam a desfragmentação.

FRAGMENTAÇÃO NA PARTIÇÃO VARIÁVEL



Os processos vão ocupando os espaços de memória disponíveis de acordo com os métodos de alocação.

Quando o espaço de memória disponível é pequeno demais para ser ocupado por um processo, ele se torna um fragmento. Considerando que ele está fora de uma partição, ele é um fragmento externo.

Cada partição possui um registro de seu endereço inicial e o tamanho que ocupa na memória.
O processo fica associado à partição e os valores de endereço e tamanho ficam nos registradores para confinar o processo dentro da partição e impedir que outros processos invadam a sua partição.

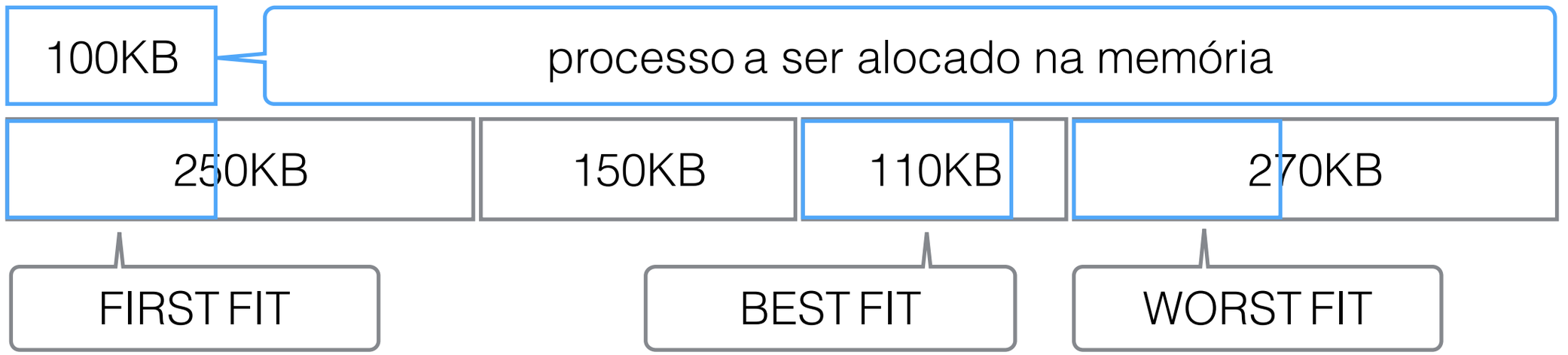
Considerando que a fragmentação externa é composta de fragmentos externos que nada mais são do que espaços disponíveis de memória, esses espaços podem ser agrupados para formarem um espaço que comporte um processo maior. O método de alocação da partição na memória pode reduzir a fragmentação externa.

ALOCAÇÃO NA PARTIÇÃO VARIÁVEL

ALOCAÇÃO FIRST FIT
Aloca o primeiro espaço de memória disponível que seja de tamanho maior ou igual ao tamanho do processo. É um método simples com menos overhead, mas é pouco inteligente e gera mais fragmentação externa.

ALOCAÇÃO BEST FIT
Aloca o menor espaço de memória disponível que pode comportar o processo. Gera mais overhead porque exige uma pesquisa das áreas de memória disponíveis, porém gera menos fragmentação externa.

ALOCAÇÃO WORST FIT
Apesar do nome, esse método simplesmente aloca o maior espaço de memória disponível para o processo. Esse método gera espaços de memória maiores para comportar outros processos, revelando-se tão bom quanto o BEST FIT.

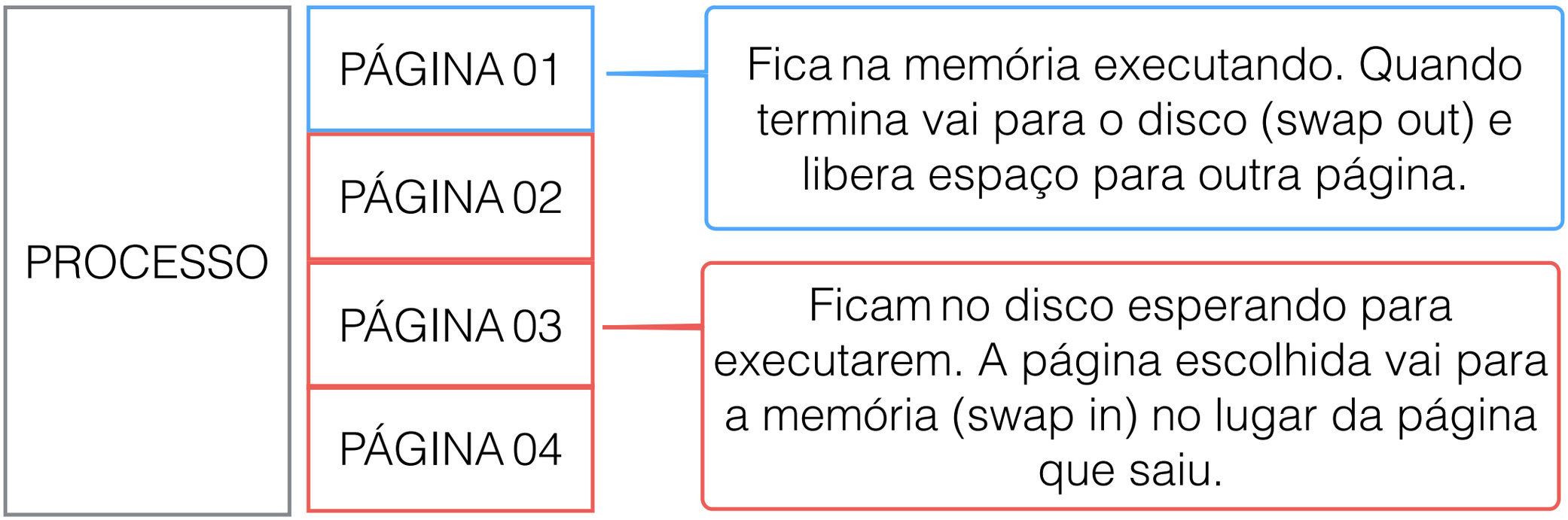


PÁGINAS DE MEMÓRIA

Com o aumento da memória do computador e da capacidade e multiprogramação, mais processos foram para a memória e menos swap de processos foram ocorrendo.

O problema é que os processos também ficaram maiores e foram divididos para que todas as partes de cada processo em execução ficassem na memória enquanto que as demais partes ficam no disco esperando para executarem.

As partes de um processo chamam-se páginas e cada página nada mais é do que uma partição fixa.



PÁGINAS LÓGICAS E FÍSICAS

As páginas físicas ficam na RAM. Quanto mais RAM, mais páginas :)

As páginas que ficam no disco estão no arquivo de trocas (swap). Quanto maior é este arquivo, mais páginas no disco podemos ter.

As páginas lógicas são todas as páginas gerenciáveis pelo S.O. Muitas vezes não se tem espaço na RAM e no disco para representar todas essas páginas. Nestes casos, o número de páginas lógicas está limitado à capacidade do disco + a capacidade da RAM.

A tabela de páginas representa o conjunto de páginas lógicas. Cada página tem um registrador onde cada bit representa um atributo da página.

Se o invalid_bit=0 então
    a página correspondente está na RAM
senão
    a página correspondente está no disco.

O processo é executado por páginas, quando a página termina, o S.O. encaminha o PC para a próxima página lógica. Dependendo do invalid_bit o processo pode continuar executando ou ficará bloqueado, esperando o swap in de sua página.

SWAP IN

Quando a página termina sua execução, e o processo necessita de outra para continuar, a próxima página lógica é selecionada. Quando invalid_bit=1, se diz que houve page fault e é necessário buscar a página no disco.

Se não existe espaço na RAM para fazer swap in, uma página deve ser escolhida como vítima para dar lugar á nova página carregada.

Existem vários métodos para escolher a página vítima. O que vai ser estudado aqui é o algoritmo de segunda chance.

O reference_bit, presente no registrador de cada página na RAM, é setado para 1 toda vez que a página é acessada para execução.

Quando há page fault, o algoritmo de segunda chance passa pelas páginas na memória fazendo reference_bit=0 nas páginas com reference_bit=1. Se na passagem existir alguma página com reference_bit=0 significa que ela não foi usada deste o último page fault. Ela então é escolhida como vítima e será substituída.

Se o lock_bit, presente no registrador de cada página na RAM, estiver setado em 1, significa que a página não pode ser escolhida como vítima, independente do valor do reference_bit. As páginas com lock_bit=1 são dos serviços do S.O. e esses serviços devem ter todas as suas páginas na RAM.

SWAP OUT

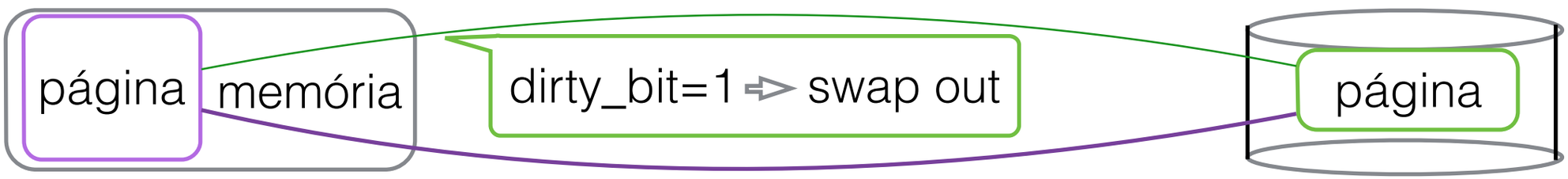
O dirty_bit, presente no registrador de cada página na RAM, quando igual a 1, significa que a página foi modificada na memória. Se ela for escolhida como vítima, se faz necessário o swap out.

O swap out grava a página a ser substituída no arquivo de swap. Quando isso acontece, existe um overhead adicional que triplica o tempo de swap (in e out).

O processo de gravação é duas vezes maior que o tempo de leitura (grava, lê verificando e confirma), na melhor da hipóteses (se confirmação der negativo, refaz o processo novamente).

Se o dirty_bit=0, a página vítima não foi alterada e é exatamente igual a sua imagem no disco (se estiver lá). Desta forma, basta ler a nova página do disco e gravar sobre a página vítima.

O read_only_bit, presente no registrador das páginas na RAM, quando igual a 1, determina que esta página não pode ser alterada na memória, ficando sempre igual a sua imagem no disco.



TRATAMENTO DE PAGE FAULT

Se vários processos entrarem em page fault, um grande overhead vai ser gerado, fazendo com que o disco tenha que atender os processos que estão todos bloqueados esperando por suas páginas. O LED do HDD vai ficar aceso que nem o LED do power.

Esse overhead ocorre porque o HDD é muito lento se comparado com a CPU. Tem também o escalonamento do HDD para atender aos vários processos esperando pelo swap de suas páginas. O HDD pode atender a uma requisição por vez e os processos ficam na fila esperando...

Quando tranca tudo (cpu em 0% e HDD em 100%) diz-se que o sistema deu trashing. Por isso que digo: Compre o máximo de memória para o seu computador que o dinheiro pode pagar!

Se você tiver muita RAM, os processo ficam com mais páginas nela.

Para prevenir altas taxas de page fault, que seguramente resultam em trash, o S.O. possui um algoritmo de tratamento para deixar o justo número de páginas na RAM para os processos. Quanto maior o número de page faults, mais o processo ganha páginas na RAM.

ALGORITMO DO TRATAMENTO DE PAGE FAULT

```
VARIÁVEIS
    INT PID // identificador do processo
    INT TP // tempo médio de swap (nano segundos).
    INT TM // tempo médio de acesso à memória (nano segundos).
    REAL TMAX // tempo máx. de page faults admitida (nano segundos).
    REAL TMIN // tempo mín. de page faults admitida (nano segundos).
    REAL TPF[PID] // taxa page fault do processo [0-1].
    REAL TA[PID] // tempo de acesso à página (nano segundos).
INÍCIO
    obter TMAX, TMIN, TP, TM
    percorrer processos
        obter PID, TPF[PID]
        TA[PID]=TP*TPF[PID]+TM*(1-TPF[PID])
        se TA[PID]>TMAX     então AUMENTE PÁGINAS PARA PID
        se TA[PID]<TMIN     então REDUZA   PÁGINAS PARA PID
    fim percorrer
FIM
```

Os processos de um modo geral estão com TPF com escopo (0-1). Os processo que tendem a 1 devem ser bonificados com páginas na memória e os que tendem a 0, devem ceder páginas da memória.

SEGMENTAÇÃO

Os segmentos são construídos sobre as partições variáveis.

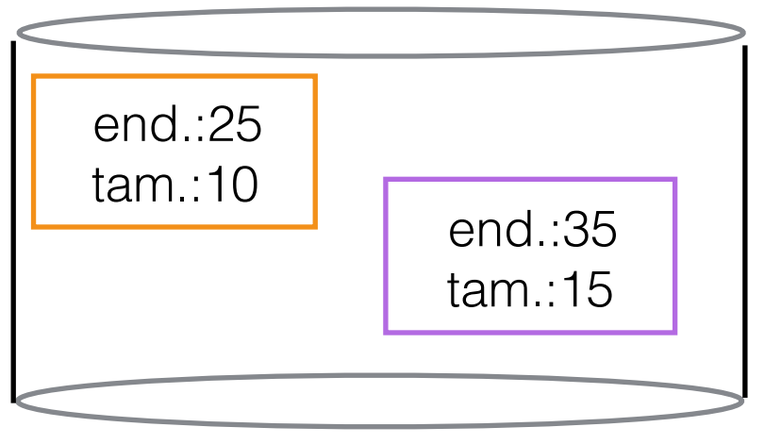
Diferente das páginas, os segmentos são definidos pelo programa executável, que tem os parâmetros de quantos processos ele vai ser desmembrado, cada um equivalendo a um segmento de memória.

Programa	Processos	Segmentos	Memória
Análise de Perfil de vendas Dados Menu CRUD Estatísticas Relatórios	Dados	end.:10 tam.:05	end.:10 tam.:05
	Menu	end.:20 tam.:05	end.:15 tam.:05
	CRUD	end.:15 tam.:05	end.:20 tam.:05
	Estatísticas	end.:25 tam.:10	
	Relatórios	end.:35 tam.:15	

end.: endereço inicial na memória lógica.

tam.: tamanho do segmento na memória

HDD

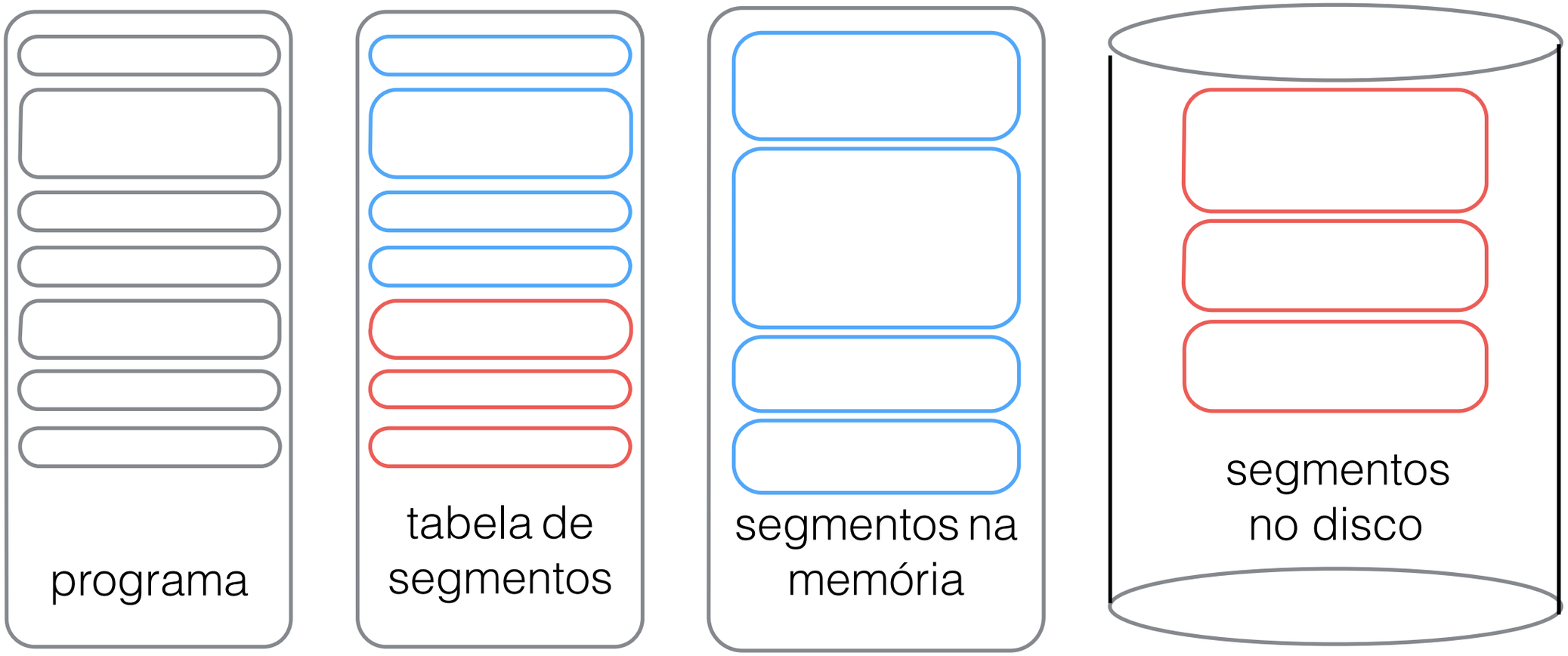


SEGMENTAÇÃO

O sistema de organização da tabela de segmentos é similar ao da tabela de páginas.

Existe uma tabela lógica de segmentos que faz referência com os segmentos na memória e no disco (para o caso de não haver espaço na RAM).

A maior diferença deste modelo para o das páginas é que as páginas são pré-definidas pelo S.O. e os segmentos são criados por demanda, a medida que os processos são carregados para execução.



SEGMENTAÇÃO

VANTAGENS

Os segmentos gerados podem ser compartilhados com vários programas, reduzindo o espaço em memória. Por exemplo, a estrutura de dados utilizada em um processo pode ser compartilhada por outros que acessem o mesmo segmento.

Os segmentos são filhotes da partição variável. Os processos tem segmentos de mesmo tamanho, não gerando fragmentação interna (não existe desperdício de espaço no segmento).

DESVANTAGENS

A gerência de memória para os segmentos é mais complexa, justamente porque eles são criados por demanda e possuem tamanhos variáveis, de acordo com o processo, gerando um overhead maior.

Cada segmento deve ter um registrador que define o endereço inicial na memória lógica e o seu tamanho e, além deste controle, é necessário saber quais áreas de memória estão desocupadas para fazer a alocação de outros segmentos (mais overhead gerado).

É gerada uma fragmentação externa, causada pelas áreas de memória disponíveis pequenas demais para conter um segmento. Se faz necessária a relocação dos segmentos para aumentar as áreas livres.

SEGMENTAÇÃO PAGINADA

Nos sistemas operacionais modernos, existe uma gerência de memória, com o driver correspondente para fazer o endereçamento da memória física no vetor contínuo de memória apresentado no início deste material.

Na camada acima, existe a gerência de páginas com o controle das páginas armazenadas nas memórias física e virtual.

E, na camada acima das páginas, existe a camada de segmentos, gerenciada pelos serviços correspondentes, de maneira semelhante ao gerenciamento das páginas.