Sumário: C5 Sistemas Operacionais

A memória principal é um componente fundamental em qualquer sistema de computação porque constitui o *"espaço de trabalho"†* do sistema. O hardware de memória envolve diversas estruturas, como caches, unidade de gerência etc., o que exige um esforço de *gerênca** significativo por parte do sistema operacional.

Estruturas de memória

Há vários *tipos de memória**, embora tenham o mesmo objetivo: armazenar dados. As características de cada permite definir uma **hierarquia de memória**: memórias mais rápidas são menores, mais caras e consomem mais energia; além de serem voláteis.

Outras características importantes:

- tempo de acesso (latência) ⇒ relativo ao ínicio da transferência
- taxa de transferência ⇒ relativa a velocidade de leitura/escrita

Endereços, variáveis e funções

Ao programar em linguagens de alto nível, não há a necessidade do programador definir ou manipular endereços de memória explicitamente. Todavia, o processador acessa endereços de memória para buscar as instruções a executar e seus operandos; acessa também outros endereços de memória para escrever os resultados do processamento das instruções.

Dessa forma, os endereços das variáves e trechos de código usados por um programa devem ser definidos em algum momento entre a escrita do código e sua execução pelo procesador, que podem ser:

- durante a edição
- durange a compilação
- durante a ligação

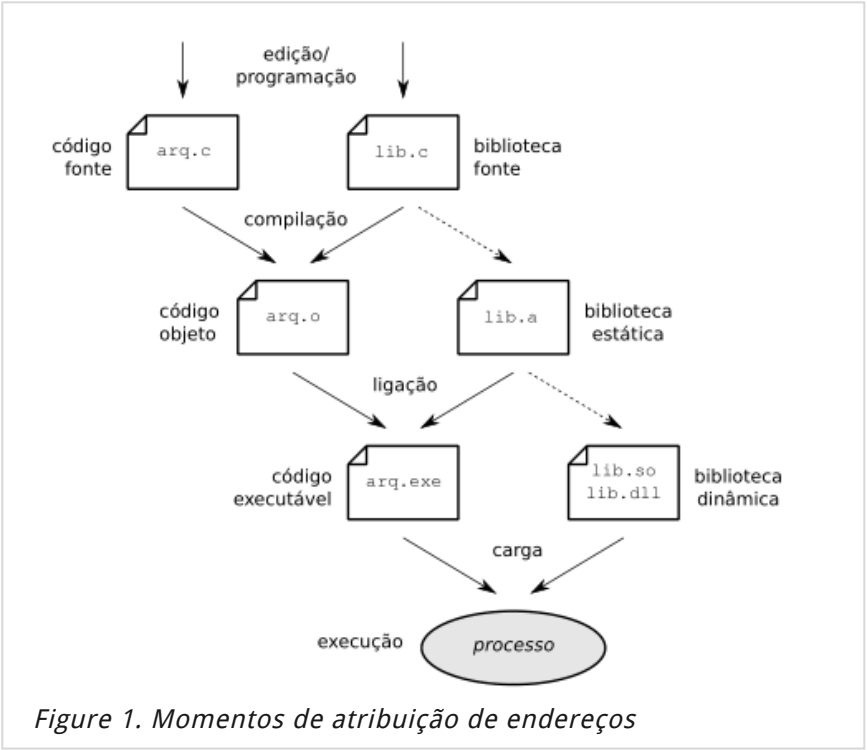

Figure 1. Momentos de atribuição de endereços

- durante a carga
- durante a execução

Endereços lógicos e físicos

Ao executar uma sequência de instruções, o processador escreve endereços no *barramento de endereço*. **Tais endereços são chamados de *endereços lógicos***, que logo em seguida são interceptados pela *MMU**. Caso o acesso a um determinado endereço solicitado pelo processador não seja possível, a MMU gera uma interrupção de hardware para notificar o processador sobre a tentativa de acesso indevido.

A proteção de memória entre processos é essencial para a segurança e estabilidade dos sistemas mais complexos, nos quais centenas ou milhares de processos podem estar na memória simultaneamente.

A forma de conversão da MMU pode ser alterada para cada processo.

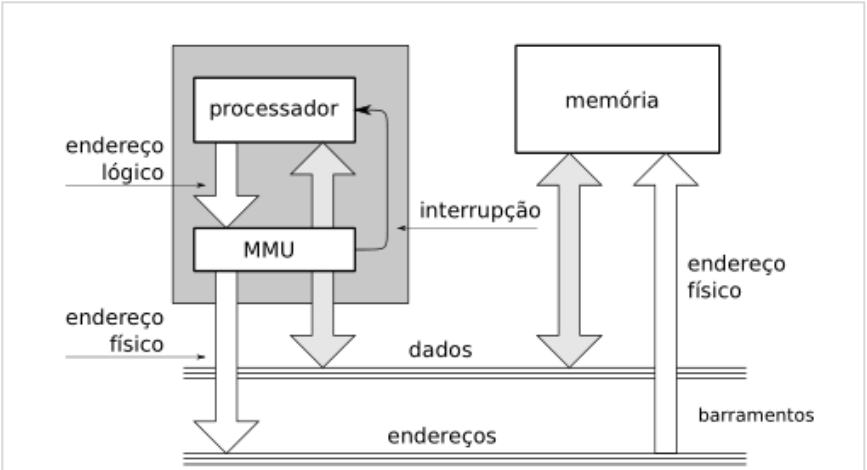

Figure 2. Funcionamento básico de uma MMU

Modelo de memória dos processos

Cada processo é visto pelo sistema operacional como uma cápsula isolada, ou seja, uma área da memória exclusiva que só ele e o núcleo do sistema podem acessar.

Essa área contém as seguintes seções:

- TEXT
- DATA
- HEAP
- STACK

As duas áreas de tamanho variável (*stack* e *heap*) estão dispostas em posições opostas e vizinhas à memória livre (não alocada). Dessa forma, a memória livre disponível ao processo pode ser aproveitada da melhor forma possível, tanto pelo *heap* quanto pelo *stack*, ou por ambos.

Estratégias de alocação

Em um sistema monoprocesso basta reservar uma área de memória para o núcleo do sistema operacional e alocar o processo na memória restante, respetaindo a disposição de suas áreas internas. Como a maioria das arquiteturas de hardware define o vetor de interrupções nos endereços inicias

da

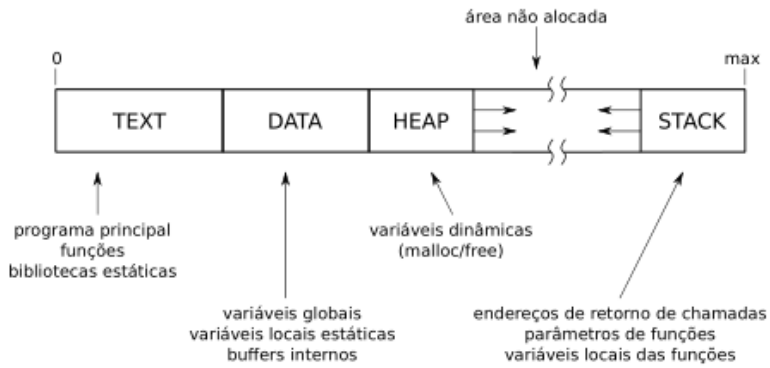

Figure 3. Organização da memória de um processo

memória, geralmente o núcleo também é colocado na parte inicial da memória. O restante de memória disponível é destinada aos processos no nível do usuário.

Nos **sistemas multiprocessos**, o espaço de memória destinado aos processos deve ser dividido enre eles usando uma estratégia que permita eficiência e flexibilidade. Os principais tipos são:

- Particões fixas
- Alocações contígua
- Alocação por segmentos
- Alocação paginada
- Alocação segmentada paginada

Partições fixas

A forma mais simples: consiste em dividir a memória destinada aos processos em N partições fixas, de tamanhos iguais ou distintos.

Características

- Em cada partição pode ser carregado um processo.
- MMU é um registrador de relocação: end. lógico + *val. do reg.* ⇒ end. físico

- O *valor* está guardado no registrador associado à partição *i* ativa no momento.
- Deve-se atualizar o registrador a cada troca de processo ativo.
- Endereços maiores que tam. da partição são rejeitados.

Desvantagens

- Processos com tamanhos distintos das partições ⇒ áreas de memória sem uso no final de cada partição.
- # máximo de processos na memória = # de partições.
- Proceso com tam. maior do que a partição não podem ser carregados.

Alocação contígua

Complemento: o tamanho da partição agora pode ser ajustado para se adequar a cada processo.

Características da MMU

- Dois registradores:
  - *base* ⇒ define o endereço inicial da partição ativa.
  - *limite* ⇒ define o tamanho em bytes dessa partição.
  - Seus valores...
    - devem ser ajustados pelo despachante (*dispatcher*) a cada troca de contexto.
    - são armazenados no respectivo TCB.
    - são 0 e ∞ caso o processo seja do núcleo.
- Cada end. lógico gerado pelo processo em execução é comparado ao valor do registrador limite, e se
  - **>=**, então uma interrupção é gerada pela MMU de volta para o processador--**endereço inválido**.
  - **<**, então é somado ao valor do registrador base—obtém o end. físico correspondente.
- Propicia a proteção de memória entre os processos.

- Tentativa de acesso fora do [base, base+limite-1] conduz a uma IRQ—end. inválido.
    - Então o processador... interrompe a execução, retorna ao núcleo, ativa rotina de tratamento.

Vantagens

- Depende apenas de dois registradores e de uma lógica simples para MMU
- Pode ser implementada em hardware de baixo custo, ou incorporada a processadores mais simples.

Desvantagens

- Pouco flexível
- Muito sujeita à fragmentação externa

Alocação por segmentos

Complemento: o espaço de memória de um processo é fracionado em áreas, ou *segmentos*, que podem ser alocados separadamente na memória física.

Além das áreas funcionais básicas, também podem ser definidos segmentos para itens específicos, como

- bibliotecas compartilhadas,
- vetores,
- matrizes,
- pilhas de *threads*,
- buffers de entrada/saída,
- etc.

O espaço de memória de cada processo é, agora, visto como uma coleção de segmentos de tamanhos diversos e políticas de acesso distintas.

Os endereços gerados pelos processos são *bidimensionais**, ou seja, indicam as posições de memória e os segmentos onde elas se encontram: [*segmento:offset*].

Cabe ao compilador colocar os diversos trechos de código fonte de cada programa em segmentos separados.

A implementação da tabela de segmentos varia conforme a arquitetura de hardware considerada.

Para cada endereço de memória acessado pelo processo em execução, é necessário acessar a tabela de segmentos para obter os valores de base e limite correspondentes ao endereço lógico acessado. Todavia, como as tabelas de segmentos normalmente se encontram na memória principal, esses acessos têm um custo significativo.

Para contornar esse problema, os processadores definem alguns *registradores de segmentos*, que permitem armazenar os valores de base e limite dos segmentos mais usados pelo processo ativo. O conteúdo desses registradores é preservado no TCB de cada processo a cada troca de contexto, tornando o acesso à memória bastante eficiente caso poucos segmentos sejam usados simultaneamente.

Características da MMU

- Similiar à alocação contígua, todavia cada segmento terá seus próprios valores de base e limite.
    - Logo, precisamos de uma *tabela de segmentos**.

A alocação por segmentos exige o uso de endereços bidimensionais, o que é pouco intuitivo para o programador e torna mais complexa a construção de compiladores, além de ser bastante suscetível à fragmentação externa.

Alocação paginada

Aqui, o espaço de endereçamento **lógico** dos processos é mantido linear e unidimensional.

- Internamente, tal espaço é dividido em pequenos blocos de mesmo tamanho, denominados *páginas*.
- Na memória física, o espaço é dividido em blocos de mesmo tamanho que as páginas, denominados *quadros*.

A alocação é feita indicando em que quadro se encontra cada página de cada processo.

- As páginas não usadas pelo processo não precisam estar mapeadas.

O mapeamento entre as páginas de um processo e os quadros correspondentes na memória física é feita através de uma *tabela de páginas*. Cada entrada: página $x \Rightarrow$ **# quadro**.

Cada processo possui sua própria tabela de páginas; a tabela de páginas ativa, que corresponde ao processo em execução no momento, é referênciada por um registrador do processador denominado PTBR--*Page Table Base Register*. A cada troca de contexto, esse registrador deve ser atualizado.

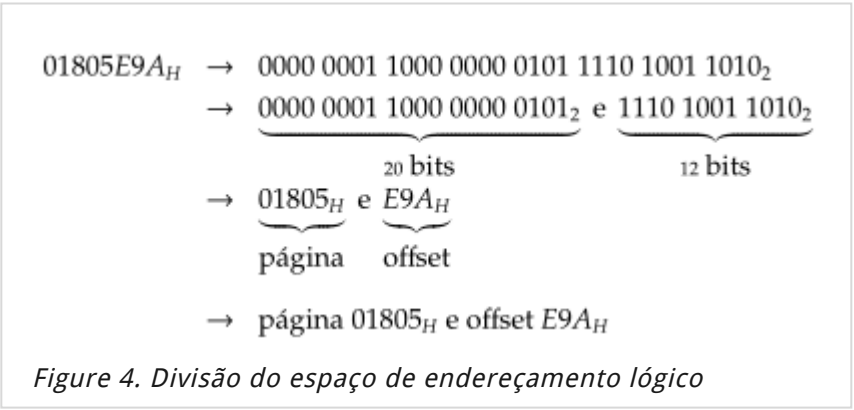

Figure 4. Divisão do espaço de endereçamento lógico

Funcionamento da MMU

1. Decompor o endereço lógico em número de página e *offset*.
2. Obter o número do quadro onde se encontra a página desejada.
3. Construir o endereço físico,
    a. compondo o número do quadro com o *offset*.
4. Caso a página solicitada não esteja mapeada em um quadro da memória física, a MMU deve gerar uma interrupção de *falta de página* para o processador.
5. Essa interrupção provoca o desvio da execução para o núcleo do sistema operacional, que deve então tratar a falta da página.

Flags de controle

Dado que o espaço de endereçamento lógico de um processo pode ser extremamente grande, uma parte significativa das páginas de um processo ode não estar mapeada em quadros de memória física.

- A tabela de páginas de cada processo indica as áreas não mapeadas com um flag adequado (*válido/inválido*).
- Outras flags (bits) de controle: **presença**, **proteção**, **referência**, e **modificação**; entre outros.

Tabelas multi-níveis

Em uma arquitetura de 32 bits com páginas de 4 KBytes, cada entrada na tabela de páginas ocupada cerca de 32 bits: 20 ⇒ # quadro, 12 ⇒ flags. Com 2^{20} páginas, cada tabela tem 4 MBytes caso seja armazenada de forma linear. Com muitos processos pequenos e muitas páginas não mapeadas, tal tabela ocupará mais espaço que o próprio processo.

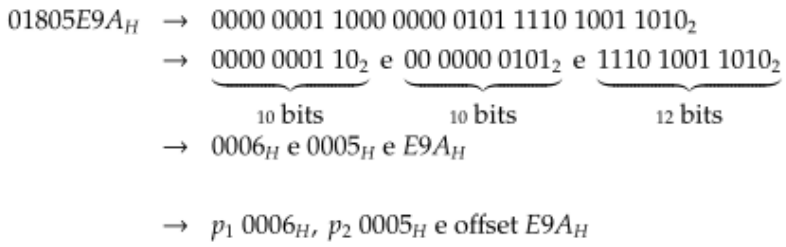

Figure 5. Divisão do espaço de endereçamento lógico usando tabelas multi-níveis

A tradução de EL para EF é similar. Haverá uma decomposição a mais, apenas.

No pior caso onde um processo ocupa toda a memória possível, seriam necessárias uma tabela de primeiro nível e 1.024 tabelas de segundo nível. Isso representa 0,098% a mais para armazenar que se a tabela de páginas fosse estruturada em um só nível. É $4 \times (2^{10} \times 2^{10} + 2^{10})$ bytes vs. 4×2^{20} bytes.

Cache da tabela de páginas

Por outro lado, há um efeito colateral no uso de tabelas multi-níveis: **tempo de acesso à memória**.

- Cada acesso a um endereço de memória implica em mais acessos para percorrer a árvore de tabelas e encontra o número de quadro desejado.
- Por isso, consultas recentes à tabela de páginas são armazenadas em um *cache* dentro da própria MMU.

O cache da tabela de páginas na MMU, denominado TLB (*Translation Lookaside Buffer*), armazena pares [*página, quadro*] obtidos em consultas recentes às tabelas de páginas do processo ativo.

- Funciona como uma tabela de *hash*: $p \Rightarrow q$ (ou *cache miss*).
- TLBs de processadores típicos têm entre 16 a 256 entradas.

A tradução de EL para EF é mais rápida, mas também mais complexa.

- TLB é um hardware: MMU primeiro o consulta; caso não tenha o nṁero do quadro, é realizada uma busca normal (completa) na tabela de páginas.
- O quadro obtido nessa consulta/busca é usado para compor o EF e também é adicionado ao TLB para agilizar as consultas futuras.
- Quanto maior a taxa de acertos (*cache hit ratio*), melhor é o desempenho dos acessos à memória física.
- Quantidade de entradas no TLB, política de substituição das entradas do TLB, forma como cada processo acessa a memória, etc. também influenciam na sua taxa de acerto.
- A cada troca de contexto, a tabela de páginas é substituída e portanto o cache TLB deve ser esvaziado.

Alocação segmentada paginada

Cada uma das principais formas de alocação tem suas vantagens:

- **Alocação contígua**: simplicidade e rapidez;

- **Alocação por segmentos**: múltiplos espaços de endereçamento para cada processo, oferencendo flexibilidade ao programador;
- **Alocação por páginas**: grande espaço de endereçamento linear, enquanto elimina a fragmentação externa.

Alguns processadores oferecem mais de uma forma de alocação, como por exmeplo alocação *com segmentos com a alocação por páginas*, visando oferecer a flexibilidade de uma e a baixa fragmentação da outra.

Localidade de referências

A forma como os processos acessam a memória tem um impacto direto na eficiência dos mecanismos de gerência de memória, sobretudo o cache de páginas (TLB) e o mecanismo de memória virtual.

Uso eficiente dos mecanismos é concentrar o acesso em poucas páginas por vez, que determina o conceito de *localidade de referências*.

- Acessos a muitas páginas distintas em curto período irão gerar frequentes erros de cache (TLB) e faltas de páginas.

Formas de localidade de referências:

- Temporal: uso tende a gerar mais uso
- Espacial: uso de um tende a provocar uso de vizinhos
- Sequencial: *part. espacial*, onde predomina acesso sequencial de recursos

Considere um programa para o preenchimento de uma matriz, onde **cada linha está alocada** em uma **página distinta**.

- Uma implementação na qual o percurso é linha por linha usa de forma eficiente o cache da tabela de páginas.
    - Um erro de cache por nova linha acessada.
- Uma implementação com percurso por colunas gera um erro de cache TLB a cada célula acessada.
    - O cache TLB **pode não ter** tamanho suficiente para armazenar as entradas referentes às páginas usadas pela matriz.

A localidade de referẽncia de uma implementação depende de um conjunto de fatores:

- Estruturas de dados ⇒ algumas têm seus elementos alocados de forma contíngua, outras não.
- Algoritmos ⇒ define comportamento do programa no acesso à memória.
- Compilador ⇒ analisa quais vars e trechos de código são usados com frequẽncia para poder colocá-los nas mesmas páginas.

Fragmentação

É o surgimento de áreas livres entre os processos, comumente chamada de fragmentação *externa*.

Esse problema somente afeta as estratégias de alocação que trabalham com blocos de tamanho variável, como alocação contígua e alocação segmentada.

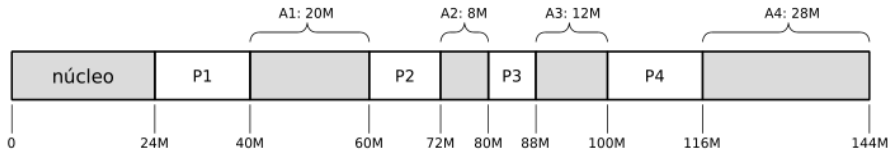

Figure 6. Memória com fragmentação externa: somente processo com até 28MB podem ser alocados

Quanto mais fragmentada estiver a memória livre, maior o esforço necessário para gerenciá-la.

- As áreas livres são mantidas em uma lista encadeada, que é manipulada a cada pedido de alocação ou liberação de memória.

Pode-se enfrentar tal problema ao

- *minimizar* sua ocorrẽncia, através de critérios de escolha das áreas a alocar

- *desfragmentar* periodicamente a memória do sistema.

No primeiro caso, temos os critérios

- **Melhor encaixe** (*best-fit*) ⇒ menor área possível (gera resíduos pequenos e logo inúteis)
- Pior encaixe (*worst-fit*) ⇒ maior área possível (gera resíduos grandes e possivelmente úteis)
- **Primeiro encaixe** (*first-fit*) ⇒ primeira área livre (rápido)
- Próximo encaixe (*next-fit*) ⇒ percorre lista a partir da última área alocada/liberada (homogeniza)

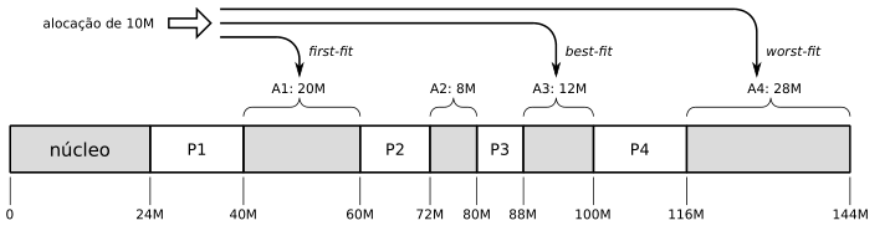

Figure 7. Estratégias para minimizar a fragmentação externa

No segundo caso (desfragmentar), as áreas de memória usadas pelos processos devem ser movidas na memória de forma a concatenar as áreas livres.

- As informações de alocação (reg. base ou tab. de segmentos) devem ser ajustadas para refletir a nova posição do processo.
- Nenhum processo pode executar durante a desfragmentação.
- Muitas possibilidades de movimentação ⇒ otimização combinatória.

Alem da fragmentação externa, as estratégias de alocação de memória também podem aprensetar a *fragmentação interna*, que pode ocorrer dentro das áreas alocadas aos processos.

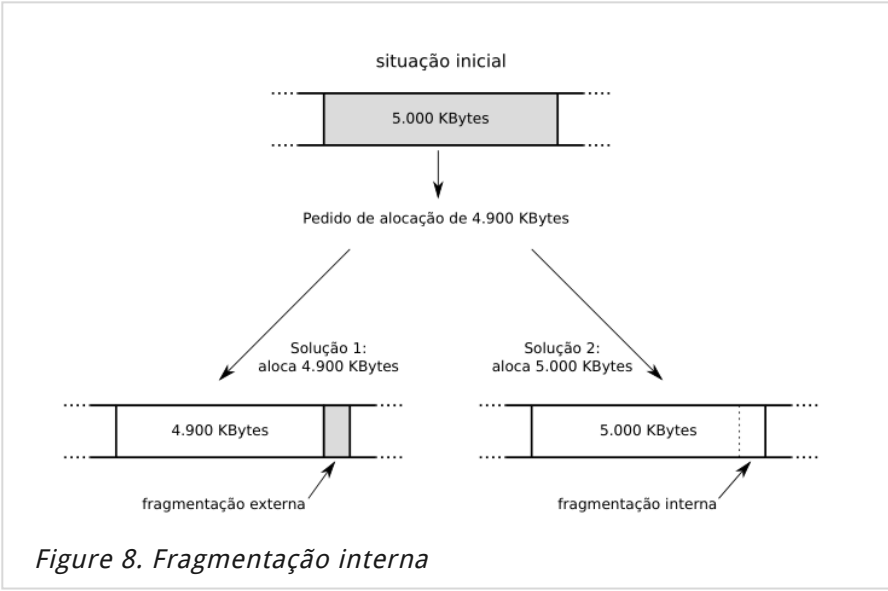

Figure 8. Fragmentação interna

- Afeta todas as formas de alocação.
- As alocações contíngua e segmentada sofrem menos, pois o nível de arrendodamento 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.
- Pode-se minimizar a perda por usar páginas menores; todavia isso implica em ter mais páginas por processo, o que geraria tabelas de páginas maiores e com maior custo de gerência.

Há um custo x benefício entre perder memória livre vs ganhar performance (acesso, gerência, etc).

Compartilhamento de memória

Nos sistemas atuais, é comum ter várias instâncias do mesmo programa em execução, como várias instâncias de editores de texto, de navegadores, etc. Assim, centenas ou milhares de cópias do mesmo código executável poderão coexistir na memória do sistema.

Normalmente, a seção **CODE** não precisa ter seu conteúdo modificado durante a execução (*read-only*). Assim, seria possível *compartilhar* essa área entre todos os processos que executam o mesmo código.

- Tal compartilhamento pode ser implementado através dos mecanismos de tradução de endereços da MMU, como segmentação e paginação.
- Em princípio, toda área protegida contra escrita pode ser compartilhada.

No compartilhamento **por segmentação** os segmentos de código dos processos apontariam para o mesmo segmento da memória física.

No compartilhamento **por paginação** as entradas das tabelas de páginas dos processos envolvidos são ajustadas para referenciar os mesmos quadros de memória física.

- As páginas compartilhadas podem ter endereços lógicos distintos, apesar de referenciarem os mesmos endereços físicos.

Uma forma mais agressiva de compartilhamento de memória é proporcionada pelo mecansmo denominado *copiar-ao-escrever* (COW, *Copy-On-Write*).

- Todas as áreas de processo (segmentos ou páginas) são passíveis de compartilhamento por outros processos, à condição que ele ainda não tenha modificado seu conteúdo.
    a. **Ao carregar um novo processo em memória**, o núcleo protege todas as áreas de memória do processo contra escrita, usando os flags da tabela de páginas (ou de segmentos).
    b. **Quando o processo tenta escrever na memória**, a MMU gera uma interrupção (negação da escrita).
    c. **O sistema operacional ajusta então os flags daquela área** para permitir a escrita e devolve a execução ao processo.
    d. Processos subsequentes idênticos ao primeiro serão **mapeados sobre as mesmas áreas de memória** física do primeiro processo que ainda não foram modificadas por ele.
    e. Se um dos processos envolvidos tentar escrever em uma dessas áreas, **a MMU gera uma interrupção**.
    f. **O núcleo então faz uma cópia separada daquela área física** para o processo que deseja escrever nela e desfaz seu

compartilhamento, ajustando as tabelas do processo que provocou a interrupção.

        - Os demais processos **continuam compartilhando** a área inicial.

Todo esse procedimento é feito de forma transparente para os processos envolvidos, visando compartilhar ao máximo as áreas de memória dos processos e assim otimizar o uso da RAM.

Áreas de memória compartilhada também podem ser usadas para permitir a comunicação entre processos.

- Dois ou mais processos solicitam ao núcleo o mapeamento de uma área de memória comum, sobre a qual podem ler e escrever.
- Como os endereços lógicos acessados nessa área serão mapeados sobre a mesma área de memória física, o que cada processo escrever nessa área poderá ser lido pelos demais, imediatamente.

Memória Virtual

Um **problema** constante nos computadores é a disponibilidade de memória física. Observando o comportamento de um sistema operacional, constata-se que nem todos os processos estão constantemente ativos, e que nem todas as áreas de memória estão sendo usadas.

- Por isso, as áreas de memória pouco acessadas poderiam **ser transferidas** para um meio de armazenamento mais barato e abundante, como um disco rígido.
- Quando um processo precisar acessá-la, ela deve ser transferida de volta para memória RAM.

O uso de um armazenamento externo como **extensão da memória RAM** se chama *memória virtual*.

Mecanismo básico

Nos primeiros sistemas a implementar estratégias de memória virtual, processos inteiros eram transferidos da memória para o disco rígido e vice-versa--*swapping*.

Nos sistemas atuais as transferências são feitas por páginas ou grupos de páginas--*paging*.

- As páginas têm um tamanho único e fixo, o que permite
    - simplificar os algoritmos de escolha de páginas a remover,
    - os mecanismos de transferência para o disco
    - e também a formatação da área de troca no disco.

A **ideia central** é retirar da memória principal as páginas menos usadas, salvando-as em um área do disco rígido reservada para esse fim.

- Essa operação é feita de modo reativo ou proativo.
- **Reativo** é quando a quantidade de memória física disponível cai abaixo de um certo limite.
- **Reativo** é quando se aproveita os perídos de baixo uso do sistema para retirar da memória as páginas pouco usadas.
- As páginas a retirar são escolhidas de acordo com algoritmos de substituição.
- As entradas das tabelas de páginas relativas às páginas transferidas para o disco devem então ser ajustadas de forma a referencar os conteúdos correspondentes no disco rígido.

No caso de um disco exclusivo ou partição de disco, essa área de troca é geralmente formatada usando uma estrutura de sistema de arquivos otimizada para o armazenamento e recuperação rápida das páginas.

Quando um **processo tenta acessar** uma página a MMU verifica se a mesma está mapeada na memória RAM.

- Em caso **positivo**, faz o acesso ao endereço físico correspondente.
- Em caso **negativo**, uma interrupção de falta de página é gerada pela MMU.

No caso da **falta de página**, o sistema deve verifivar se a página solicitada não existe ou se foi transferida para o disco, usando os flags de controle da respectiva entrada da tabela de páginas.

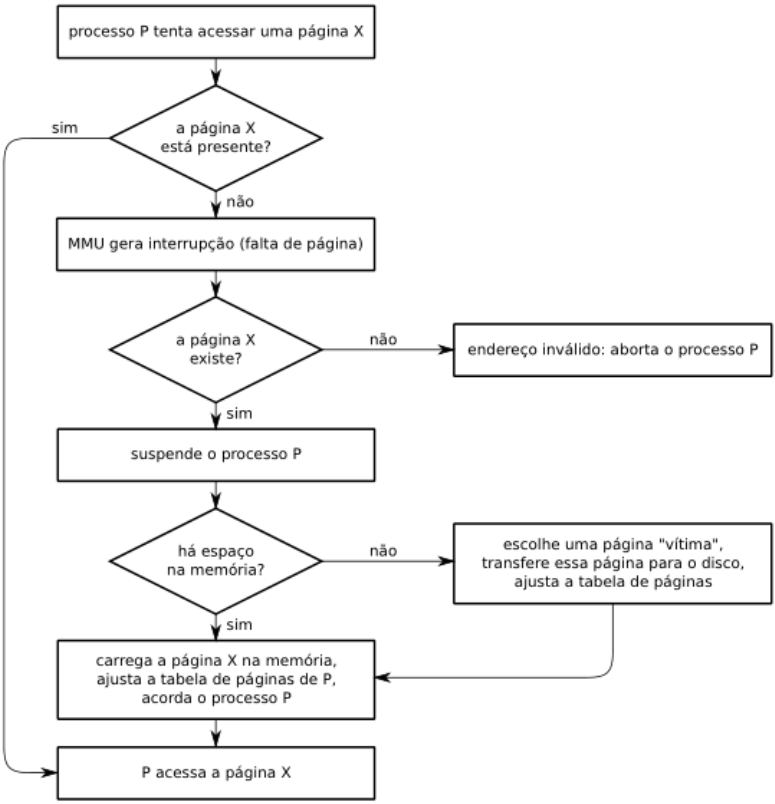

Figure 9. Ações do mecanismo de memória virtual

- Caso **não exista**, o processo tentou acessar um endereço inválido e deve ser abortado.
- Por outro lado, o processo **deve ser suspenso** enquanto o sistema operacional transfere a página de volta para a memória RAM e faz os ajustes necessários na tabela de páginas.

Questões importantes

- Caso a memória principal já esteja cheia, uma ou mais páginas deverão ser removidas para o disco antes de trazer de volta para a página faltante.

- Retomar a execução do processo que gerou a falta de página pode ser uma tarefa complexa
    - A instrução não completada deve ser reexecutada.
    - No caso de instruções que envolvam várias ações e vários endereços de memória, deve-se descobrir
        - qual dos endereços gerou a falta de página,
        - que ações da instrução foram executadas e então executar somente o que estiver faltando.

Eficiência de uso

O mecanismo de memória virtual seria a solução ideal para as limitações da memória principal, se não houvesse um problema importante: **o tempo de acesso dos discos utilizados**.

- Um disco rígido típico tem um tempo de acesso cerca de 100.000 vezes maior que a memória RAM.
- Cada falta de página provocada por um processo implica em...
    - um acesso ao disco para buscar a página faltante;
    - ou dois acessos, caso a memória RAM esteja cheia e outra página tenha de ser removida antes.

A frequência de falta de página depende de vários fatores, como

- **O tamanho da memória RAM**, em relação à demanda dos processos em execução ⇒ *memória insuficiente ou muito carregados podem gerar muitas faltas de página*.
- **O comportamento dos processos em relação ao uso da memória** ⇒ *acessos respeitando a localidade de referências geram menos faltas de página.*
- **A escolha das páginas a remover da memória** ⇒ *remoção de páginas muito acessadas geram mais faltas de página.*

Glossário e complementos

- **Espaço de trabalho**: local onde são mantidos os processos, threads, bibliotecas compartilhadas e canais de comunicação, além do próprio

núcleo do sistema operacional.

- Uma **gerência** adequada da memória é essencial para o bom desempenho de um computador.
- **Tipos de memória**: registradores, cache interno do processador (L1), cache externo da placa mãe (L2) e a memória principal (RAM); discos rígidos e unidades de armazenamento externas, também.
- Os **barramentos de endereço** servem para buscar instruções e operandos, mas também para ler e escrever valores em posições de memória e portas de entrada/saída.
- **Endereços lógicos** são assim chamados porque correspondem à lógica do programa, mas não são necessariamente iguas aos endereços reais das instruções e variáveis na memória do computador.
- **Endereços físicos** são os endereços reais das instruções e variáveis na memória do computador.
- **MMU**, ou Unidade de Gerência de Memória, faz parte do próprio processador e é responsável pela análise dos endereços lógicos e determina os endereços físicos correspondentes.
- Os **endereços lógicos bidimensionais** fazem parte da estratégica de alocação chamada *alocação por segmentos*, onde são representados pelo número do *segmento* e a posição de memória deste (*offset*).
- A **tabela de segmentos** guarda os valores das bases e limites para cada segmento usado pelo processo, além de *flags* com informações sobre cada segmento, como permissões de acesso etc.