AULA 14 - Memória Virtual

A idéia básica da memória virtual é permitir que programas muito maiores que a memória disponível possam ser executados. Para isso, em 1961 *Fotheringham* criou o método conhecido como memória virtual. Com o uso dessa memória o SO mantém as partes ativas do programa na memória e o restante em disco, sendo carregadas ou removidas da memória de acordo com as necessidades.

Memória virtual é uma técnica que se utiliza da memória secundária para produzir o efeito prático de aumentar significativamente o espaço de endereçamento disponível aos programas. Essa técnica não depende do tamanho da memória principal – que continua sendo limitado – para ser implementada. A memória secundária (normalmente um disco rígido) passa a servir como uma espécie de extensão da memória principal, armazenando a maior parte dos programas e dados carregados para execução. À memória principal são transferidas por vez apenas algumas partes destes programas e dados, essenciais ao momento pontual da execução.

Deste modo, cria-se dois espaços de endereçamento de memória: o espaço de endereçamento virtual e o espaço de endereçamento real. Endereços virtuais são os endereços gerados pelos compiladores e *linkers* na implementação dos programas. Eles representam o espaço completo que os programas necessitam para serem carregados, isto é, o tamanho de memória que efetivamente ocupam. No entanto, a memória real – ou memória principal (RAM – *Random Access Memory*) – pode ser bastante limitada, possuindo um espaço de endereçamento real às vezes menor até mesmo que um único programa. Um endereço real corresponde a uma célula física de endereçamento presente na memória principal da máquina.

Obviamente, apenas parte das aplicações carregadas no espaço de endereçamento virtual poderá ocupar a memória real num certo instante. Por outro lado, somente os dados carregados na memória principal são processados pela CPU. Como várias aplicações podem estar sob execução, concorrendo pela utilização do processador, uma solução encontrada foi dividir o espaço total de endereçamento dos programas em pequenos blocos de tamanho fixo, as chamadas páginas de memória. A implementação dessa estratégia constitui o mecanismo de **paginação**. Outra alternativa é dividir o código de uma aplicação com base em critérios lógicos, ou seja, modularizando o programa de acordo com a função dos diversos trechos de código identificados. **Segmentação** é o nome que se dá a esta segunda estratégia de particionamento de código, em que

os segmentos gerados possuem tamanhos independentes e variáveis. Ambos os mecanismos são importantes, sobretudo em ambientes multiusuários e multiprogramáveis. Conforme o momento de execução, parte dos dados (*working set*) são transferidos temporariamente para a memória principal a fim de serem processados, retornando à memória secundária quando solicitados pelo sistema operacional, o qual é responsável pelo gerenciamento da memória.

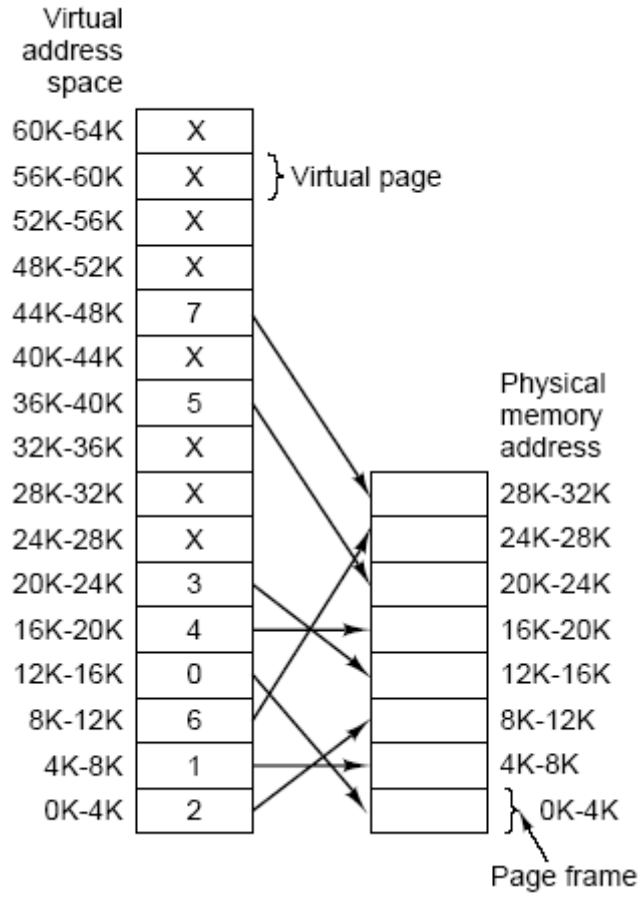


Figura 1. Endereços virtuais x físicos. A tradução é feita por uma tab. de páginas.

Paginação

A técnica denominada paginação é utilizada na maioria dos sistemas com memória virtual. Como sabemos, os endereços gerados pelos programas são denominados *endereços virtuais* e constituem o espaço de endereçamento virtual. Em computadores que não tem memória virtual, os *endereços virtuais* são idênticos aos endereços físicos. Por outro lado, quando usamos memória virtual estes endereços diferem. Portanto, o endereço gerado pelo programa não pode ser colocado diretamente no barramento do sistema para acessar uma posição qualquer de memória. Assim, a MMU mapeia os end. virtuais em físicos.

A *memória física* é dividida em blocos de tamanho fixos denominados *molduras de páginas* (*page frames*). Já a *memória lógica* é dividida em blocos de tamanho fixos denominados *páginas* (*pages*). As páginas e as molduras de páginas são sempre do mesmo tamanho.

Quando um programa tenta usar uma página virtual que não está mapeada é gerada uma interrupção da CPU para o sistema operacional a fim de buscar esta página na memória. Esta interrupção (trap) é denominada *falta de página (page fault)*. As ações desencadeadas são: o sistema operacional escolhe uma moldura de página (*page frame*) pouco usada e a salva em disco. Em seguida, carrega a página virtual referenciada pela instrução na moldura de página que foi liberada. Feito isso o sistema operacional pode reinicializar a instrução causadora da interrupção.

Tabela de Páginas

A tabela de páginas é montada durante o instante em que o processo é carregado e servirá para informar qual é a página fisica correspondente a cada página lógica. Para que um programa possa ser executado corretamente é necessário que o endereço lógico especificado em cada instrução seja traduzido no endereço fisico que representa as posições reais de memória.

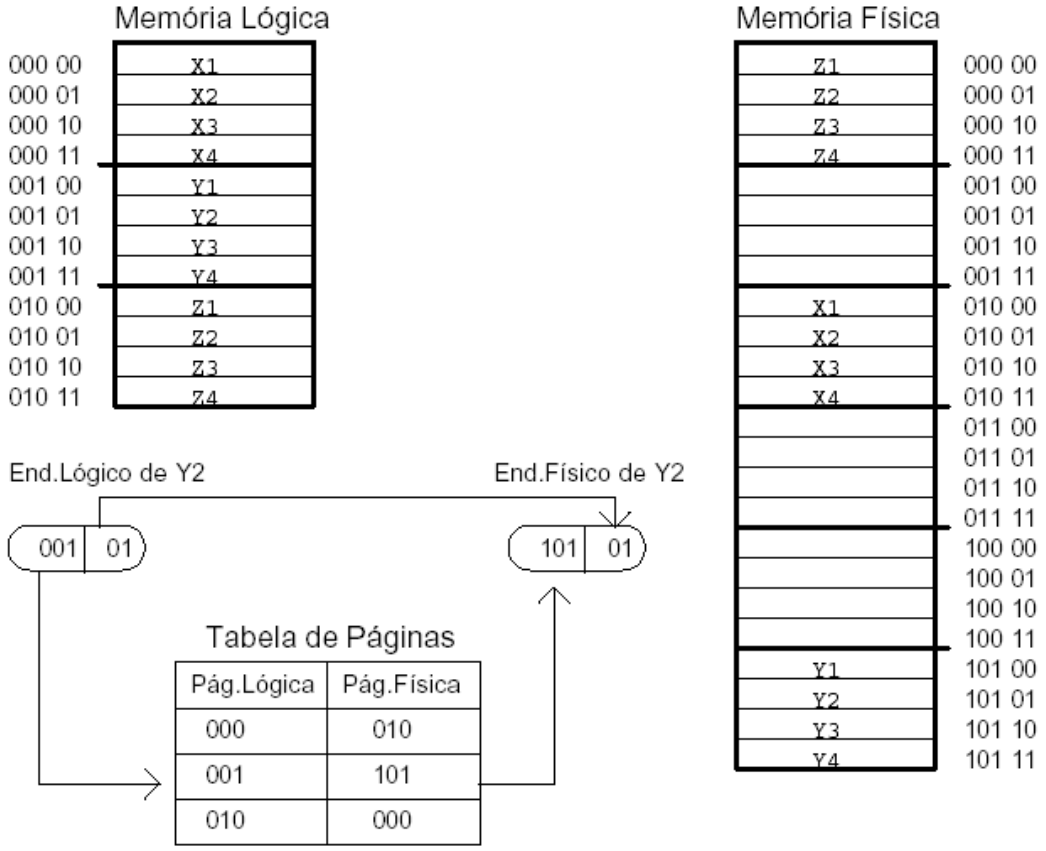


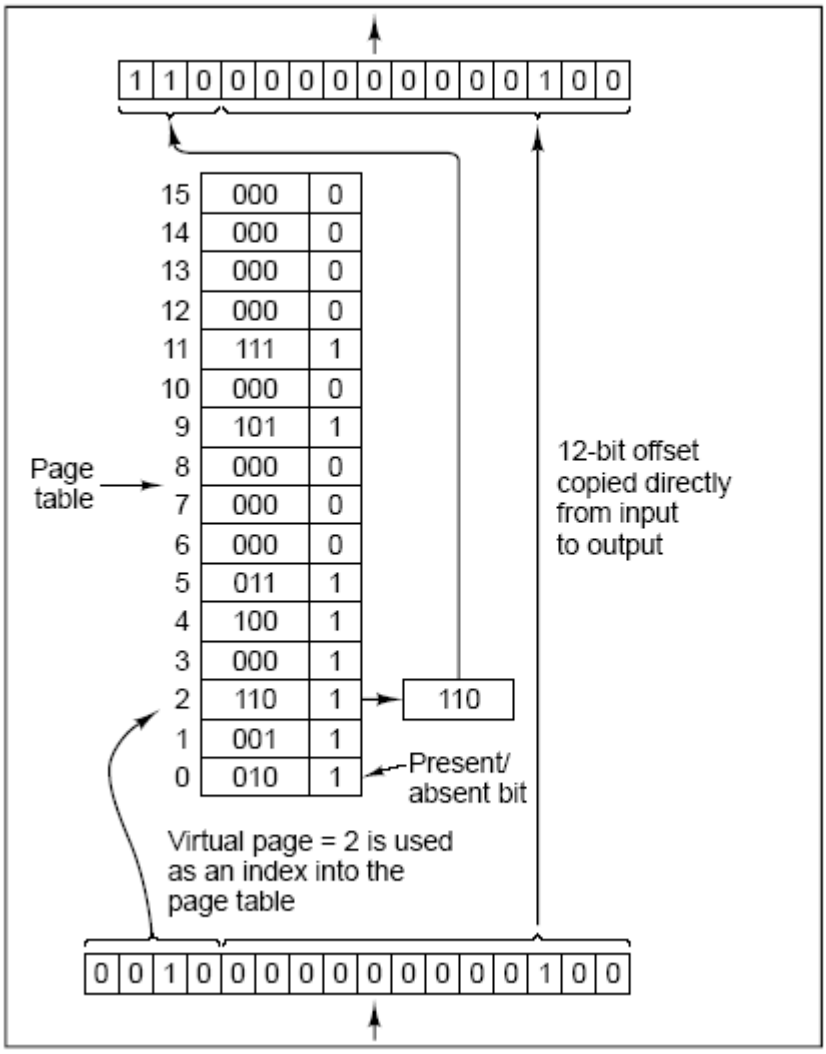
Figura 2. Esquema de paginação.

Conforme podemos observar na Figura 2 o endereço lógico é constituído de 2 partes: a primeira (bits de mais alta ordem) corresponde ao número da página lógica e a segunda representando o deslocamento dentro desta página (bits de mais baixa ordem). O número da página lógica é usado para encontrar a página física correspondente através da tabela de páginas. Os bits de mais baixa ordem são mantidos e ajudam a constituir o endereço físico. Para isso é só utilizar os bits encontrados na tabela de páginas com os bits de deslocamento. A Figura 2 mostra como isso é feito considerando o endereço lógico do byte Y2. O endereço lógico 00101 é dividido em número de página lógica 001 e deslocamento 01. A entrada 001 da tabela de páginas indica que essa página lógica foi carregada na página física 101. Então, unimos as duas partes e formamos o endereço físico 10101.

Geralmente, os tamanhos de página variam entre 512 bytes e 64 Kbytes, sendo que um valor muito utilizado é o de 4 Kbytes. Os espaços de endereçamento lógico podem variar de 64 Kbytes (sistemas antigos) até muitos Gbytes (máquinas atuais). Já os espaços de endereçamento físico podem ser na ordem de Gbytes.

Durante a paginação, qualquer página física disponível pode ser utilizada para carregar uma página lógica. Por isso, não existe fragmentação externa. No entanto, pelo fato da unidade de alocação ser uma página, qualquer processo sempre ocupará um número inteiro de páginas físicas (podendo não ocupar todo o espaço da página) gerando com isso uma fragmentação interna. Por exemplo, imagine que um sistema utilize páginas de 4 Kbytes, e um programa P1 precise de 201 Kbytes para ser carregado e poder executar. Nesse sistema, 51 páginas serão alocadas, totalizando 204 Kbytes. Portanto, 3 Kbytes serão desperdiçados (fragmentação interna).

Podemos dizer que o tamanho da página definido pode trazer algumas vantagens e desvantagens ao sistema computacional. Portanto, é importante saber definir um tamanho de página coerente para que o sistema possa realizar a paginação. Utilizar páginas de tamanhos maiores significa que um processo ocupará menos páginas e, portanto, a tabela de páginas será menor e a leitura do disco será mais eficiente. Normalmente, as páginas maiores resultam em melhor desempenho do mecanismo de gerência de memória. Contudo, devemos saber que páginas maiores geram uma fragmentação interna maior. O tamanho das páginas normalmente é fixado pelo *hardware* que suporta a gerência de memória, ou seja, pela *MMU*.



Implementação da Tabela de Páginas

A forma como a tabela de páginas é implementada é um fator importante e que deve ser pensado, uma vez que ela deve ser consultada a cada acesso à memória. Tabelas de páginas pequenas podem ser colocadas em registradores de acesso rápido, melhorando bastante o desempenho de pesquisas pelo endereço físico. Suponha que um PC tenha uma memória lógica para os processos de apenas 64 Kbytes e que cada página ocupe 8 Kbytes, então existirá apenas 8 entradas nas tabelas de páginas.

O que acontece quando um processo é escalonado e outro assume o processador? A tabela de páginas do processo que será executado deverá ser copiada do descritor de processo (DP) para os registradores.

Tabelas de páginas muito grandes não podem ser mantidas em registradores, portanto, outra solução deve ser pensada. Uma outra estratégia é manter a tabela de páginas na própria memória do computador. Nesta abordagem, a MMU possui dois registradores para localizar a tabela de páginas na memória. O primeiro é chamado de *registrador de base da tabela de páginas* (*Page Table Base Register*, **PTBR**) que indica o endereço físico de memória onde a tabela está colocada. O segundo denominado registrador de limite da tabela de páginas (*Page Table Limit Register*, **PTLR**) indica o número de entradas da tabela.

Qual o problema dessa solução? Agora cada acesso que um processo faz à memória lógica transforma-se em dois acessos à memória física. No primeiro acesso, a tabela de páginas é consultada, e o endereço lógico é transformado em endereço físico. No segundo acesso, a memória do processo é lida ou escrita. Quando processos são escalonados, os valores do PTBR e do PTLR para a tabela de páginas do processo que recebe o processador devem ser copiados do DP para os registradores na MMU. Além disso, a memória *cache* deve ser esvaziada (*flushed*), pois ela ainda contém entradas da tabela de página do processo que executou antes e agora perdeu o processador.

Qual uma estratégia para melhorar o desempenho desta solução? Adicionar uma memória cache especial pode ser uma forma de reduzir o tempo de acesso à memória no esquema apresentado. Esta cache irá manter as entradas da tabela de páginas mais recentemente utilizadas. Essa memória *cache* interna à MMU é chamada de *Translation Lookaside Buffer* (**TLB**). Portanto, quando a entrada requerida da tabela de páginas está na TLB, o acesso à memória lógica do processo é feito com um único acesso à memória física, como quando registradores são utilizados. Nessa situação diz que houve um acerto (***hit***). Por outro lado, quando a entrada da tabela de páginas associada com a página lógica não está na TLB diz que ocorreu uma falta (*miss*). Neste caso, é necessário um duplo acesso à memória física, como no esquema sem o cache. O procedimento adotado é semelhante ao que acontece na cache convencional, ou seja, a entrada referente a página lógica acessada é incluída na TLB. Os próximos acessos já encontrarão essa entrada da tabela de páginas na TLB e o acesso será rápido.

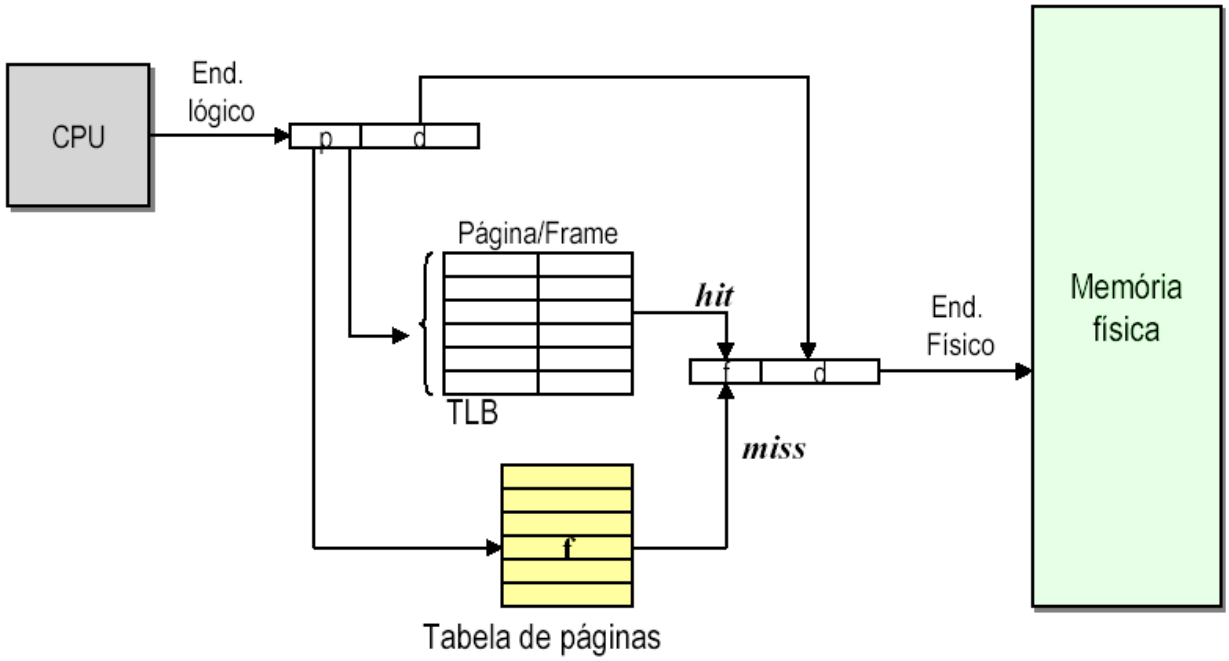


Figura 3. Implementação da tabela de páginas com TLB.

As entradas na tabela de páginas são compostas por alguns campos que permitem localizar o endereço efetivo de cada página na memória principal (quando carregada) ou na memória secundária (área de swap). As seguintes informações são mantidas [CAS03]:

1. *Número da página virtual (NPV)*: identifica uma única página virtual correspondente à entrada; de acordo com o tamanho de página utilizado, o endereço na memória secundária pode ser calculado.
2. *Bit de presença*: indica se a página está ou não carregada na memória principal (a página está carregada se o bit possui valor 1).
3. *Endereço do bloco*: se a página estiver carregada na memória principal, este campo informa o endereço físico em que está alocada; um bloco, ou janela (frame), é um conjunto de células da memória principal que comporta dados de uma página.
4. *Bits de proteção*: definem as permissões de acesso à página, normalmente através de 2 ou 3 bits (leitura, gravação e eventualmente execução).
5. *Bit de modificação*: indica se a página carregada na memória teve ou não seu conteúdo alterado; se teve, o conteúdo deverá ser atualizado na memória secundária antes de a página deixar a memória principal.

Outro conceito associado a paginação é o *working set* (conjunto de trabalho). O *working set* de um processo em execução é definido como o conjunto das páginas requeridas para o seu processamento durante um determinado intervalo de tempo. Por este motivo, o conjunto de trabalho representa a concretização do princípio de localidade.

Quando uma informação que será processada não está disponível em uma página da memória principal, dizemos que ocorreu uma falta de página. Neste momento, é preciso que esta página seja alocada na memória para que a execução do processo possa continuar. No entanto, se a memória já estiver cheia, uma das páginas carregadas deverá ser sobreposta pela página que está sendo trazida da memória secundária. Portanto, deverá acontecer uma substituição de página e, conseqüentemente, será necessário escolher qual página deixará a memória. Para isso, o sistema operacional implementa um algoritmo de substituição de página e escolhe aquela menos importante à execução do processo e a retira da memória.

Segmentação

O esquema de paginação foi criado para facilitar o gerenciamento da memória. Contudo, os programadores e compiladores não são capazes de ver a memória lógica dividida em páginas, mas sim em segmentos. Dependendo do sistema a granularidade desses segmentos pode ser diferente. Por exemplo, comumente verificamos que um programa pode ser segmentado em: código, dados alocados dinamicamente, dados alocados estaticamente e pilha de execução. No entanto, alguns sistemas enxergam cada objeto ou módulo como se fosse um segmento.

Quando o sistema utiliza o esquema de segmentação, o endereço lógico é também dividido em: número do segmento e deslocamento dentro do segmento (semelhante ao esquema de paginação). Os segmentos não necessariamente precisam ter o mesmo tamanho. No entanto, existe um tamanho máximo definido para os segmentos.

Os endereços lógicos são traduzidos para um endereço físico através de uma tabela de segmentos, que basicamente contém as mesmas informações da tabela de páginas vista anteriormente. Essa tabela informa o local onde o segmento foi colocado em memória e qual o seu tamanho.

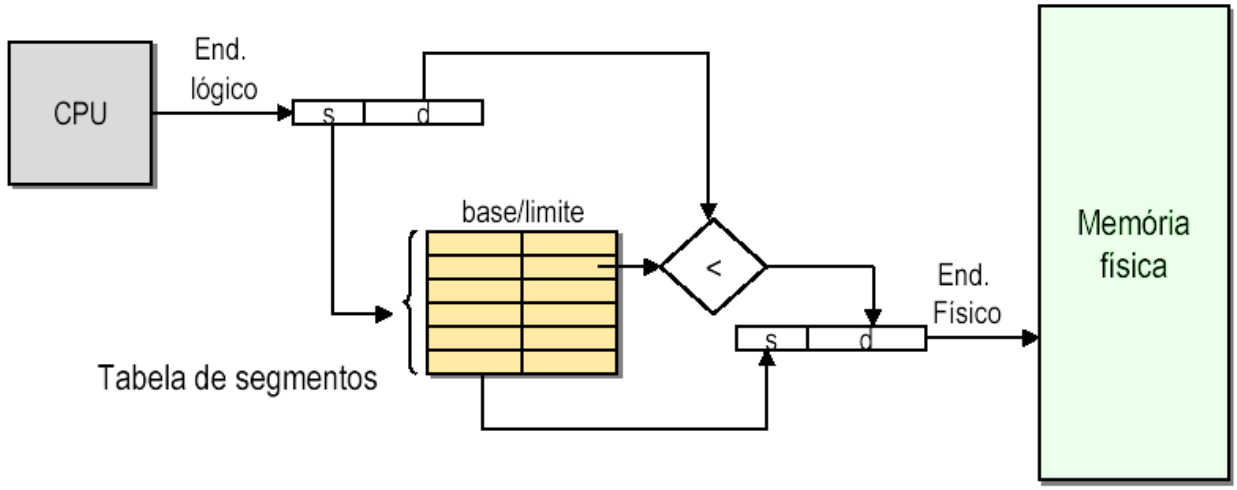
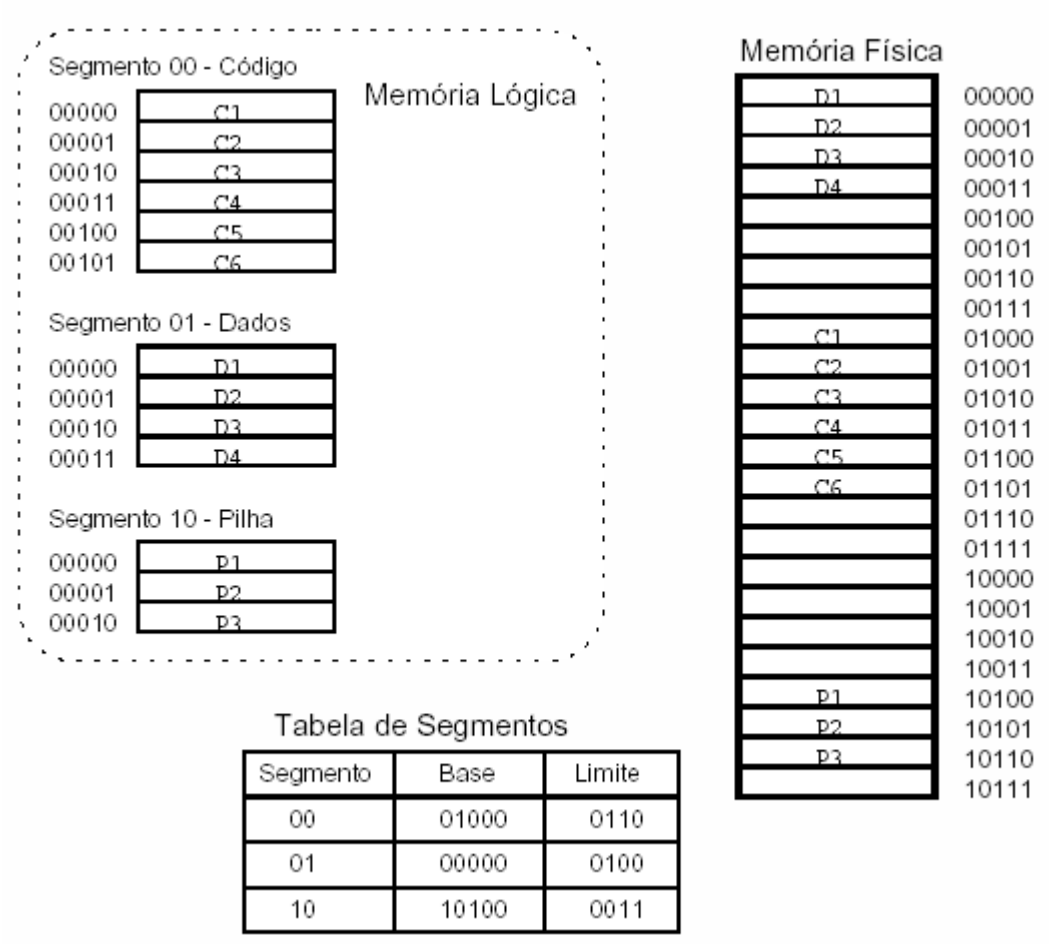


Figura 4. Esquema de tradução de endereços lógicos em físico.

Quando o deslocamento é maior que o limite especificado na tabela de segmentos significa que o processo que está executando está tentando endereçar além do final do segmento, ou seja, fora da sua área de memória lógica. Nesse instante será gerada uma interrupção de proteção.



Segmento	Base	Limite
00	01000	0110
01	00000	0100
10	10100	0011

Figura 5. Exemplo de tradução de end. lógico para físico.

Questionamento: como o end. lógico de D3 é traduzido para o end. físico?

A tabela de segmentos também pode ser implementada como a tabela de páginas. Devemos ressaltar que a segmentação não apresenta fragmentação interna, uma vez que cada segmento recebe a quantidade exata de memória que ele precisa. Contudo, a fragmentação externa se faz presente, uma vez que pequenos segmentos podem ser liberados e dificilmente serão reutilizados. É interessante observar que segmentos podem ser compartilhados, como por exemplo, um segmento que represente uma rotina específica dentro de um programa. Um segmento compartilhado só é removido da memória se ele não estiver mais sendo utilizado por nenhum processo. Este compartilhamento pode ajudar bastante em termos economia de espaço de memória.

Exercícios

01. Explique o que é uma página de memória.
02. Por que a falta de páginas deve ser evitada na execução dos programas?
03. Por que os sistemas operacionais usam um esquema chamado *paging on demand*?
04. Descreva passo a passo o que acontece após uma falta de página.
05. Qual a diferença entre paginação e segmentação?
06. Uma máquina tem um endereçamento virtual de 28 bits e um endereçamento físico de 27 bits. As páginas são de 4 KB e as palavras são de 4 Bytes. Quantas entradas/linhas são necessárias para a tabela de páginas armazenar todas as traduções dos endereços virtuais?
07. Quais as fragmentações apresentadas na gerência de memória que utilizam partições fixas e as que usam partições variáveis?
08. Considere um sistema operacional que trabalha com paginação simples. As páginas são de 1 Kbyte e as palavras são de 1 Byte. O end. lógico é formado por 29 bits. O end. físico é formado por 27 bits. Calcule:
    a. O tamanho (em Bytes) do espaço de endereçamento virtual.
    b. O tamanho (em Bytes) do espaço de endereçamento físico.
    c. O tamanho (em bits) das entradas/linhas da tabela de páginas, só traduzindo p em f.
    d. O tamanho (em bits e bytes) da tabela de páginas. Considere que para cada página virtual tenhamos uma entrada/linha na tabela.

09. O sistema operacional XYZ utiliza paginação como mecanismo de gerência de memória. São utilizadas páginas de 8 Kbytes e palavras de 2 bytes. Um endereço lógico ocupa 30 bits e o endereço físico ocupa 28 bits. Cada entrada na tabela de páginas contém, além do número da página física, um bit de válido/inválido e um bit que indica apenas leitura (*read-only*). Calcule:
    a. O tamanho (em Bytes) do espaço de endereçamento virtual.
    b. O tamanho (em Bytes) do espaço de endereçamento físico.
    c. O tamanho (em bits) das entradas/linhas da tabela de páginas, só traduzindo p em f.
    d. O tamanho (em bits e bytes) da tabela de páginas. Considere que para cada página virtual tenhamos uma entrada/linha na tabela.

Bibliografia

OLIVEIRA, R. S.; CARISSIMI, A. S. e TOSCANI, S. S. ***Sistemas Operacionais***. Sagra Luzzatto, Porto Alegre, 2001. (Cap. 5)

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 2ª Edição. Prentice Hall, 2003.

Referência Bibliográfica

[CAS03] CASSETTARI, H. H. Análise da Localidade de Programas e Desenvolvimento de Algoritmos Adaptativos para Substituição de Páginas. Qualificação de Mestrado. Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais. 2003.