


Resolução da Prova SEFAZ/2007 Tecnologia da Informação



Prof. Jaime Correia Neto
jcn25@yahoo.com

37- O gerenciamento eficiente da memória é uma tarefa crítica do sistema operacional. A respeito de tal contexto, assinale a opção correta.

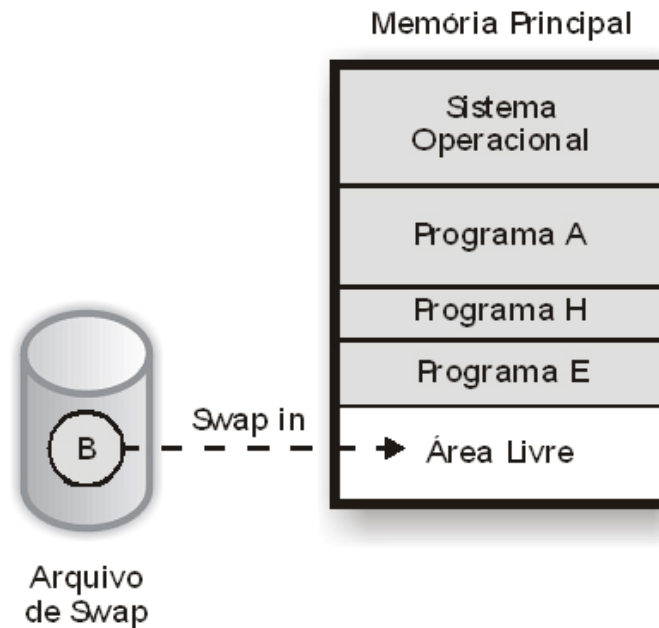
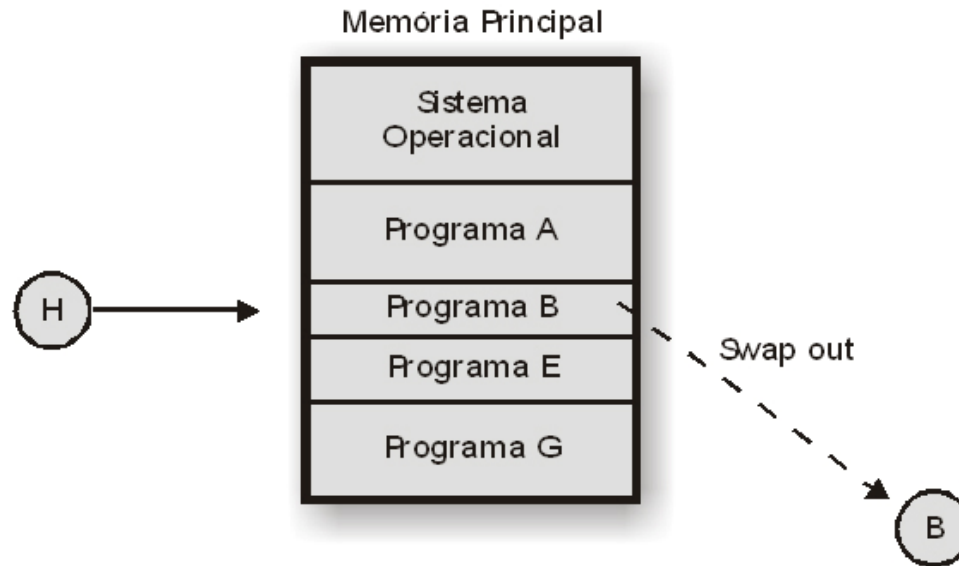
- a) A técnica denominada troca de processos (*swapping*) é usada pelo sistema operacional para mudar a localização dos processos na memória, agrupando todas as áreas de memória livre em um único bloco, consumindo parte do tempo útil do processador para essa execução.
- b) Na técnica de paginação hierárquica, cada página de um processo é transferida para a memória, apenas quando é necessária. Para melhorar o desempenho, são usados discos redundantes e replicação de páginas operados de forma paralela.
- c) Um endereço físico é aquele gerado pela CPU (*Central Processing Unit*) enquanto o endereço lógico é aquele tratado diretamente pela unidade de memória, carregado especificamente no registrador de endereço de memória.
- d) Cada entrada em uma tabela de segmentos possui uma base de segmento que contém o endereço físico inicial no qual o segmento está localizado e o limite de segmento, que indica a extensão do segmento.
- e) Em máquinas Intel 80x86, o sistema operacional Linux usa apenas cinco segmentos: segmento para o código do *kernel*, segmento para os dados do *kernel*, segmento para o código do usuário, segmento para os dados do usuário e o segmento de estado da tarefa (TSS – *Task State Segment*).



“Dois métodos gerais para o gerenciamento de memória podem ser usados, dependendo (em parte) dos recursos de hardware disponíveis. A estratégia mais simples, denominada **troca de processos** (*swapping*), consiste em trazer totalmente cada processo para a memória, executá-lo durante um certo tempo e então devolvê-lo ao disco. A outra estratégia, denominada **memória virtual**, permite que programas possam ser executados mesmo que estejam apenas parcialmente carregados na memória principal.”

TANENBAUM

O item A está falso uma vez que afirma que o objetivo do swapping é mudar a localização dos processos na memória. Seu objetivo é retirar o processo para o disco, liberando, assim, espaço para outro processo.



Processo de Swapping

Um novo processo H inicia e precisa de espaço na memória. O processo B é removido para o disco para liberar espaço.

"Hierarchical Paging

Most modern computer systems support a large logical address space (2^{32} to 2^{64}). In such an environment, the page table itself becomes excessively large. For example, consider a system with a 32-bit logical address space. If the page size in such a system is 4KB (2^{12}), then a page table may consist of up to 1 million entries ($2^{32}/2^{12}$). Assuming that each entry consists of 4 bytes, **each process may need up to 4 MB of physical address space for the page table alone**. Clearly, we would not want to allocate the page table contiguously in main memory. One simple solution to this problem is to divide the page table into smaller pieces. We can accomplish this division in several ways. One way is to use a two-level paging algorithm, in which the page table itself is also page."


OPERATING SYSTEMS CONCEPTS – SILBERSCHATZ

"A fim de minimizar o problema de continuamente armazenar tabelas de páginas muito grandes na memória, diversos computadores utilizam tabelas de páginas em múltiplos níveis. Um exemplo simples desse método é mostrado na Figura 4.12. Na Figura vê-se um endereço virtual de 32 bits dividido em um campo *PT1* de 10 bits, um campo *PT2* de 10 bits e um campo *Deslocamento* de 12 bits. Como o campo *Deslocamento* tem 12 bits, as páginas são de tamanho 4 KB. Os outros dois campos têm conjuntamente 20 bits, o que possibilita um total de 2^{20} páginas virtuais."

“O segredo para o método multinível de tabelas de páginas é evitar que todas elas sejam mantidas na memória o tempo todo, especialmente aquelas que não são necessárias. Suponha, por exemplo, que um processo necessite de 12 megabytes: os 4 megabytes da base da memória para o código do programa, outros 4 megabytes para os dados do programa e quatro megabytes do topo da memória para a pilha. Sobra, entre o topo dos dados e a base da pilha, um gigantesco espaço não usado.”

TANENBAUM

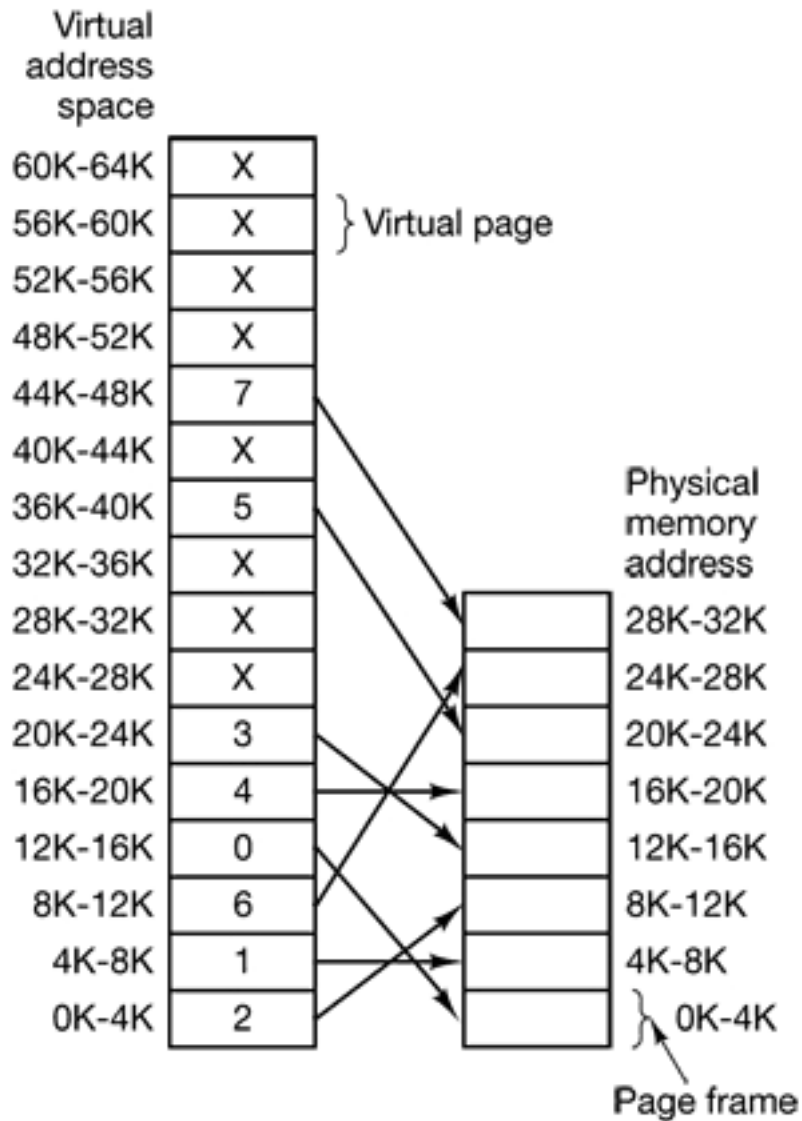
“Na maioria dos sistemas, existe uma tabela de páginas para cada processo. Cada processo pode ocupar uma grande área de memória virtual. Por exemplo, na arquitetura VAX, cada processo pode ter até $2^{31} = 2$ Gbytes de memória virtual. Usando páginas de $2^9 = 512$ bytes, a tabela de páginas de *cada processo* pode ter até 2^{22} entradas. Claramente, a quantidade de memória requerida para tabelas de páginas seria inaceitável. **Para superar esse problema, na maioria dos sistemas de memória virtual, as tabelas de páginas são também endereçadas na memória virtual, e não na memória real.** Isso significa que as tabelas de páginas também estão sujeitas à paginação. Quando um processo está em execução, parte da sua tabela de páginas tem de estar na memória principal, incluindo a entrada da tabela que corresponde à página que está sendo executada. Alguns processadores usam um esquema de dois níveis para organizar tabelas de páginas muito grandes. ”



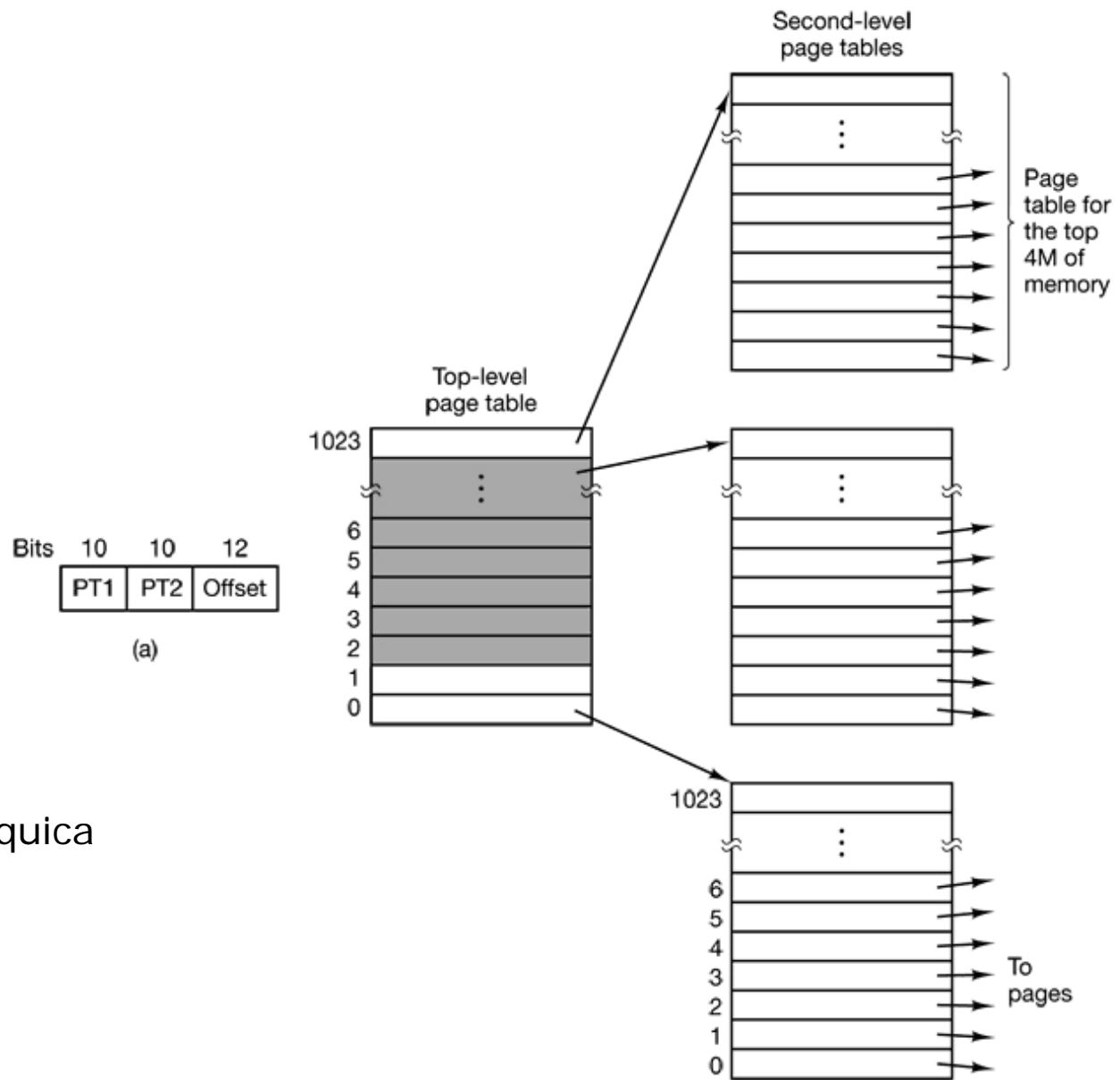
“Nesse esquema, existe um diretório de páginas, no qual cada entrada aponta para uma tabela de páginas. Dessa maneira, se o tamanho do diretório de páginas é X e o tamanho máximo de uma tabela de páginas é Y , o processo pode ter até $X \times Y$ páginas. Tipicamente, o tamanho máximo de uma tabela de páginas é restrito ao tamanho de uma página.”


STALLINGS

O item B está falso, pois o objetivo da paginação hierárquica, como visto é de otimizar o uso da memória, fazendo com que a tabela de páginas quando muito grande também sofra paginação (multinível). Isso permite economia da memória física.



Paginação Hierárquica ou Multinível






Esses endereços não são fixos, podendo mudar toda vez que o processo for trazido para a memória principal. Para isso, **é feita uma distinção entre endereços lógicos e endereços físicos. Um endereço lógico corresponde a uma posição relativa ao início do programa. As instruções do programa contêm apenas endereços lógicos. Um endereço físico designa uma posição na memória principal. Quando o processador executa um processo, ele automaticamente converte um endereço lógico para um endereço físico, somando ao endereço lógico a posição inicial da área de memória do processo, conhecida como seu *endereço-base*.** Isso é mais um exemplo de característica do hardware do processador, projetada para atender a uma necessidade do sistema operacional. A natureza exata dessa característica de hardware depende da estratégia de gerenciamento de memória em uso.


STALLINGS

O item C é falso uma vez que o registrador de endereço de memória (MAR) da CPU lida apenas com endereços físicos. Pois comunica-se diretamente com a memória.



*“Memória virtual por segmentação é a técnica de gerência de memória onde o espaço de endereçamento virtual é dividido em blocos de tamanhos diferentes chamados *segmentos*. Na técnica de segmentação, um programa é dividido logicamente em sub-rotinas e estruturas de dados, que são alocadas em segmentos na memória principal.*


Enquanto na técnica de paginação o programa é dividido em páginas de tamanho fixo, sem qualquer ligação com sua estrutura, na segmentação existe uma relação entre a lógica do programa e sua alocação na memória principal. Normalmente, a definição dos segmentos é realizada pelo compilador, a partir do código fonte do programa, e cada segmento pode representar um procedimento, função, vetor ou pilha.”



“O espaço de endereçamento virtual de um processo possui um número máximo de segmentos que podem existir, onde cada segmento pode variar de tamanho dentro de um limite. O tamanho do segmento pode ser alterado durante a execução do programa, facilitando a implementação de estruturas de dados dinâmicas. Espaços de endereçamento independentes permitem que uma sub-rotina seja alterada sem a necessidade de o programa principal e todas as suas sub-rotinas serem recompiladas e religadas. Em sistemas que implementam paginação, a alteração de uma sub-rotina do programa implica recompilar e religar a aplicação por completo.

O mecanismo de mapeamento é muito semelhante ao de paginação. Os segmentos são mapeados através de *tabelas de mapeamento de segmentos* (TMS), e os endereços são compostos pelo *número do segmento virtual* (NSV) e por um *deslocamento*. O NSV identifica unicamente o segmento virtual que contém o endereço, funcionando como um índice na TMS. O deslocamento indica a posição do endereço virtual em relação ao início do segmento no qual se encontra. O endereço físico é obtido, então, combinando-se o endereço do segmento, localizado na TMS, com o deslocamento, contido no endereço virtual.”

Arquitetura de Sistemas Operacionais - LTC

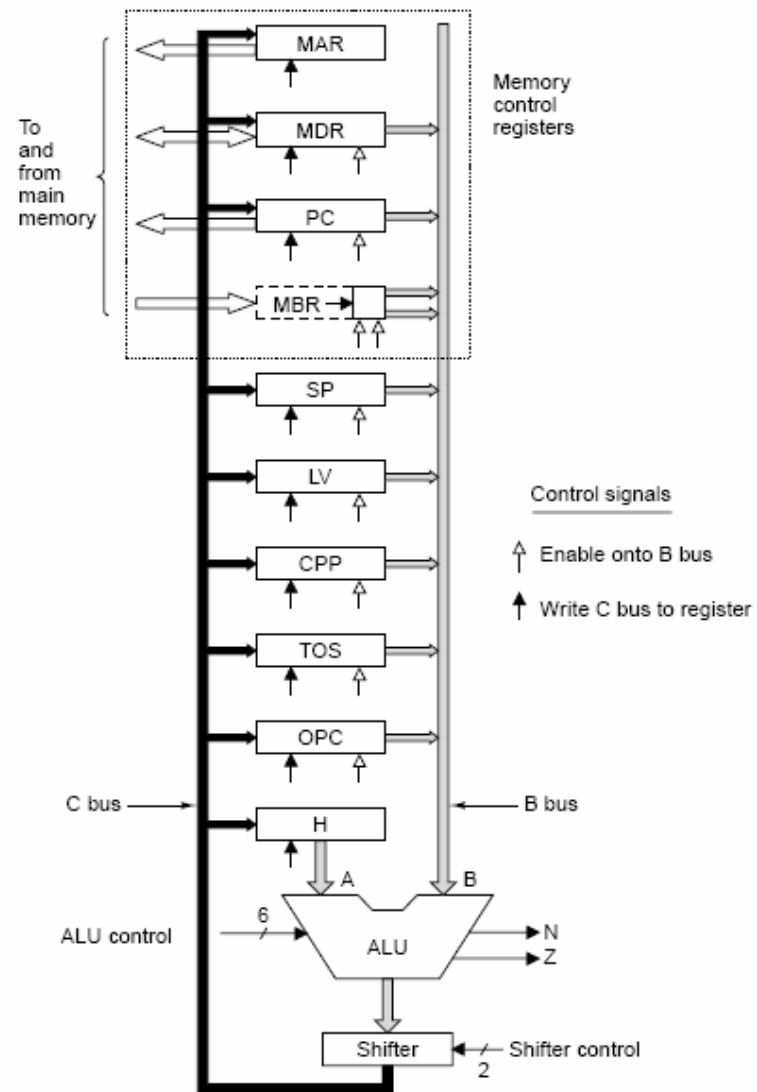


“Although the user can now refer to objects in the program by a two-dimensional address, the actual physical memory is still, of course, a one-dimensional sequence of bytes. Thus, we must define an implementation to map two-dimensional user-defined addresses into one-dimensional physical addresses. This mapping is effected by a segment table. **Each entry in the segment table has a *segment base* and a *segment limit*. The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.**”

OPERATING SYSTEMS CONCEPTS – SILBERSCHATZ

O item D está correto foi extraído de forma idêntica ao trecho do livro Operating Systems Concepts citado acima.

Note que o MAR (Memory Address Register comunica-se diretamente com a memória.)



"The 2.6 version of Linux uses segmentation only when required by the 80 x 86 architecture.

All Linux processes running in User Mode use the same pair of segments to address instructions and data. These segments are called user code segment and user data segment , respectively. Similarly, all Linux processes running in Kernel Mode use the same pair of segments to address instructions and data: they are called kernel code segment and kernel data segment , respectively. Table 2-3 shows the values of the Segment Descriptor fields for these four crucial segments.

Besides the four segments just described, Linux makes use of a few other specialized segments. We'll introduce them in the next section while describing the Linux GDT - Global Descriptor Table ."

Understanding the Linux Kernel, 3rd Edition
By Daniel P. Bovet, Marco Cesati

Segment	Base	G	Limit	S	Type	DPL	D/B	P
user code	0x00000000	1	0xffffffff	1	10	3	1	1
user data	0x00000000	1	0xffffffff	1	2	3	1	1
kernel code	0x00000000	1	0xffffffff	1	10	0	1	1
kernel data	0x00000000	1	0xffffffff	1	2	0	1	1

O item E é falso, pois como vimos o Linux quando necessário a utilização da arquitetura 80x86 usa 4 segmentos e não cinco.

Contudo, o GDT do linux trabalha com 18 segmentos, os 4 vistos anteriormente e mais 14 segmentos, onde está incluso o TSS.

<i>Linux's GDT</i>	<i>Segment Selectors</i>	<i>Linux's GDT</i>	<i>Segment Selectors</i>
null	0x0	TSS	0x80
reserved		LDT	0x88
reserved		PNPBIOS 32-bit code	0x90
reserved		PNPBIOS 16-bit code	0x98
not used		PNPBIOS 16-bit data	0xa0
not used		PNPBIOS 16-bit data	0xa8
TLS #1	0x33	PNPBIOS 16-bit data	0xb0
TLS #2	0x3b	APMBIOS 32-bit code	0xb8
TLS #3	0x43	APMBIOS 16-bit code	0xc0
reserved		APMBIOS data	0xc8
reserved		not used	
reserved		not used	
kernel code	0x60 (<code>__KERNEL_CS</code>)	not used	
kernel data	0x68 (<code>__KERNEL_DS</code>)	not used	
user code	0x73 (<code>__USER_CS</code>)	not used	
user data	0x7b (<code>__USER_DS</code>)	double fault TSS	0xf8