Hierarquia de Memória - Parte 4 - Memória Virtual

Arquitetura de Computadores

Emilio Francesquini
e.francesquini@ufabc.edu.br

2020.Q1

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

Disclaimer

UFABC

- Estes slides foram preparados para o curso de Arquitetura de Computadores na UFABC.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- O conteúdo destes slides foi baseado no conteúdo do livro *Computer Organization And Design: The Hardware/Software Interface*, 5th Edition.
- Algumas ilustrações foram retiradas do livro *Operating System Concepts* de Silberschatz, Galvin e Gagne.

Fundamentos

UFABC

- O código precisa estar na memória para ser executado, mas o programa completo raramente é usado
  - Código de erro, rotinas incomuns, grandes estruturas de dados
- Todo o código do programa não é necessário ao mesmo tempo

- Considere a capacidade de executar programas parcialmente carregados
  - Programa não seriam mais limitados pelos por limites físicos da memória RAM
  - Cada programa precisa de menos memória durante a execução: assim mais programas podem executar ao mesmo tempo
    - Maior utilização da CPU e taxa de transferência sem aumento no tempo de resposta ou no tempo de retorno
  - Menos E/S necessário para carregar ou trocar programas em memória: logo cada programa do usuário é executado mais rapidamente

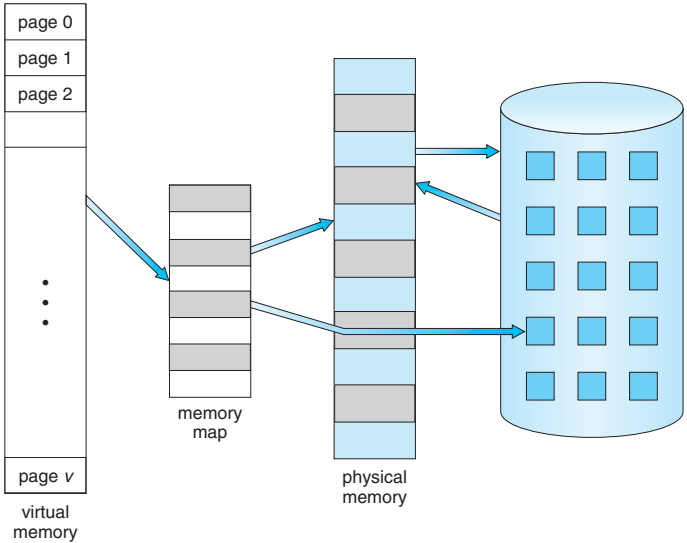
Memória virtual

UFABC

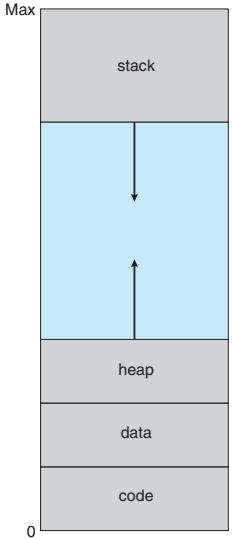
Memória virtual - separação da memória lógica do usuário da memória física

- Apenas parte do programa precisa estar na memória para execução
- O espaço de endereçamento lógico pode, portanto, ser muito maior do que espaço de endereçamento físico
- Permite que os espaços de endereço sejam compartilhados por vários processos
- Permite uma criação de processo mais eficiente
- Mais programas em execução simultaneamente
- Menos E/S necessário para carregar ou trocar processos
- Memória virtual pode ser implementada via:
  - Paginação por demanda
  - Segmentação por demanda

Memória virtual que é maior que a memória física



Espaço de endereço virtual



Espaço de endereço virtual

UFABC

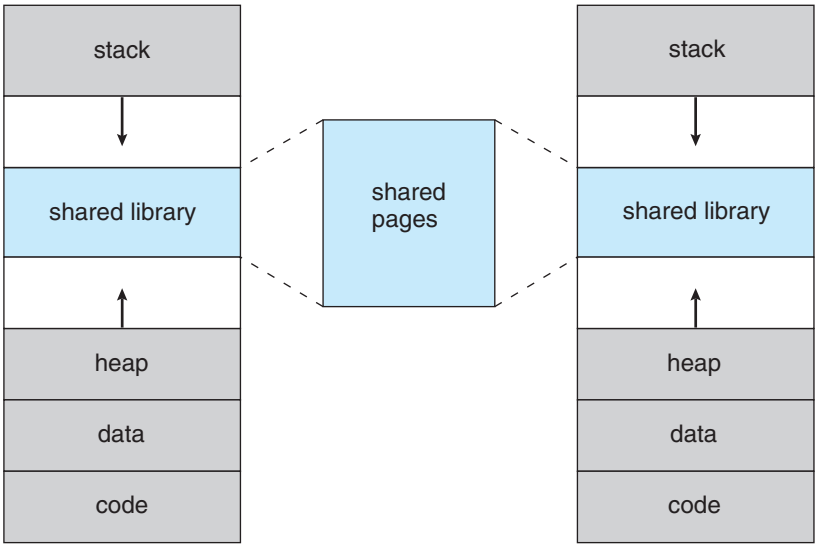
- Geralmente projeta-se o espaço de endereço lógico da pilha para começar no endereço lógico máximo e crescer ”para baixo”enquanto heap cresce ”para cima”
  - Maximiza o uso do espaço de endereçamento
  - Espaço de endereço não utilizado entre os dois é buraco
    - Nenhuma memória física necessária até a pilha ou heap crescerem até uma nova página

Espaço de endereço virtual

UFABC

- Habilita espaços de endereços esparsos com buracos deixados para crescimento, bibliotecas ligadas dinamicamente, etc.
- Bibliotecas compartilhadas via mapeamento no espaço de endereçamento virtual
- Memória compartilhada com páginas mapeadas em modo de leitura-escrita no espaço de endereço virtual
- As páginas podem ser compartilhadas durante o `fork()`, acelerando a criação de processos

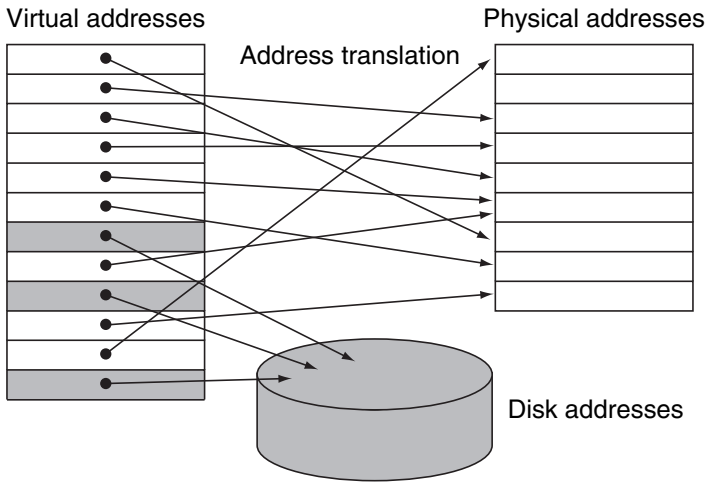
Biblioteca compartilhada usando memória virtual

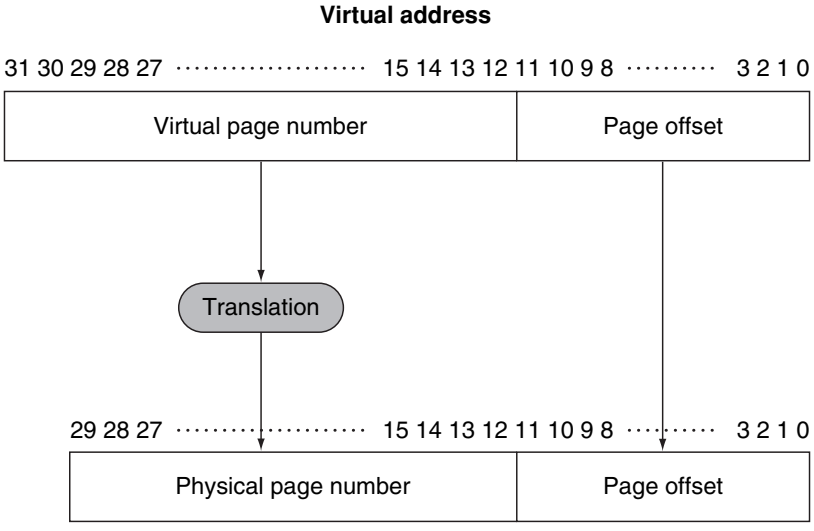


Memória-Virtual - Objetivos

- Usar a memória principal como ”cache”para o armazenamento secundário (disco, SSD,...)
  - Sua manutenção é um trabalho conjunto do SO e da CPU
- Programas compartilham a memória principal
  - Cada programa recebe um espaço de endereçamento virtual que armazena seus dados e código que são utilizados com mais frequência
  - Isola um programa dos demais
- A CPU em conjunto com o SO traduz o endereço virtual para os endereços físicos.
  - Cada ”bloco”da memória virtual é chamada de página
  - Quando ocorre um ”miss”de uma página, chamamos de falha de página (*page fault*)

- Para tamanhos de página fixos (ex. 4KB)




Virtual address
31 30 29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Virtual page number
Page offset
Translation
29 28 27 ····· 15 14 13 12 11 10 9 8 ····· 3 2 1 0
Physical page number
Page offset
Physical address

Exemplo - Intel Core i7-7500U

UFABC

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
Address sizes:         39 bits physical, 48 bits virtual
...
Model name:            Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
...
Virtualization:        VT-x
L1d cache:             64 KiB
L1i cache:             64 KiB
L2 cache:              512 KiB
L3 cache:              4 MiB

$ getconf PAGESIZE
4096
```

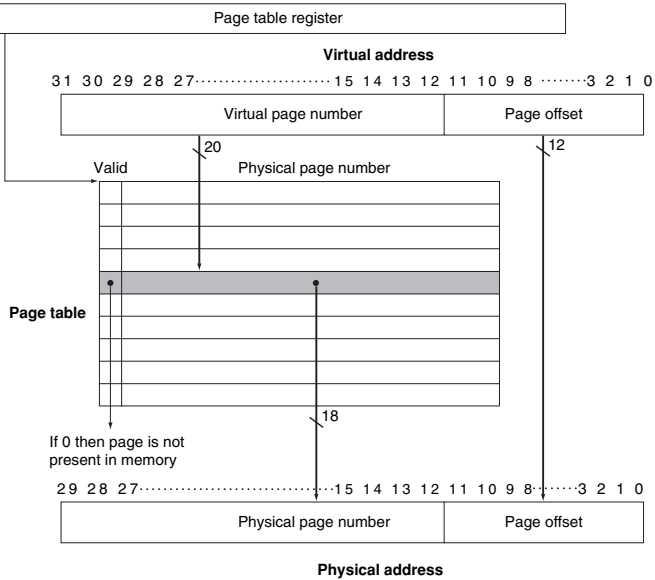
- Quando ocorre uma falha de página a página deve ser carregada do disco
  - Isso leva **milhões** de ciclos do processador
  - Tratamento é feito pelo SO
- Então, a estratégia é minimizar ao máximo a taxa de falhas de página (*page fault rate*)
  - Localização totalmente associativa (*Fully associative placement*)
  - Algoritmos de substituição de páginas inteligentes

Tabelas de páginas

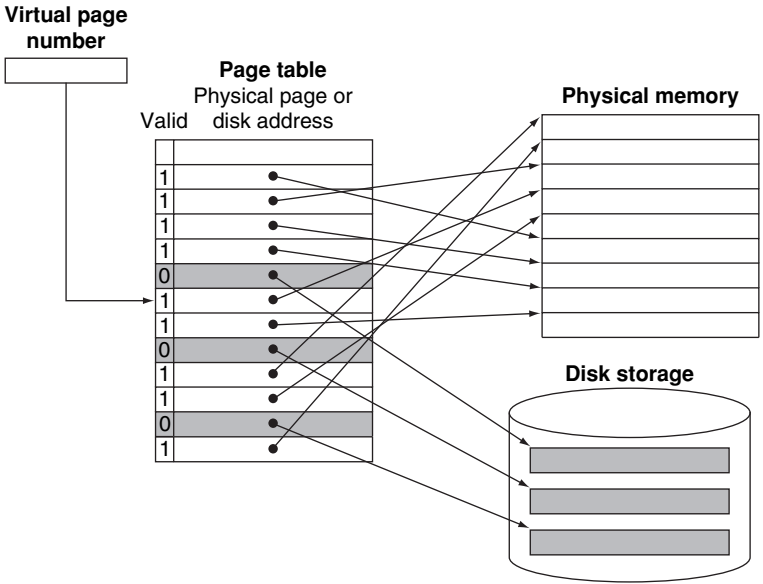
UFABC

- Tabelas de páginas (*page tables*) armazenam informações sobre a localização e estado das páginas
  - Pode ser vista como uma array de páginas indexadas pelo número virtual da página
  - O registrador da tabela de páginas na CPU aponta para a tabela de páginas armazenada na memória física
  - Se uma página está presente na memória
    - A PTE (page table entry) guarda a tradução do endereço virtual para o real
    - Além de outros bits de status como *referenced*, *dirty*, ...
  - Se não estiver presente
    - A PTE pode, por outro lado, apontar para uma localização no espaço de swap armazenado em disco.

Tradução de endereços usando uma tabela de páginas



Mapeando páginas para o dispositivo de armazenamento secundário



Substituição de páginas e escritas

UFABC

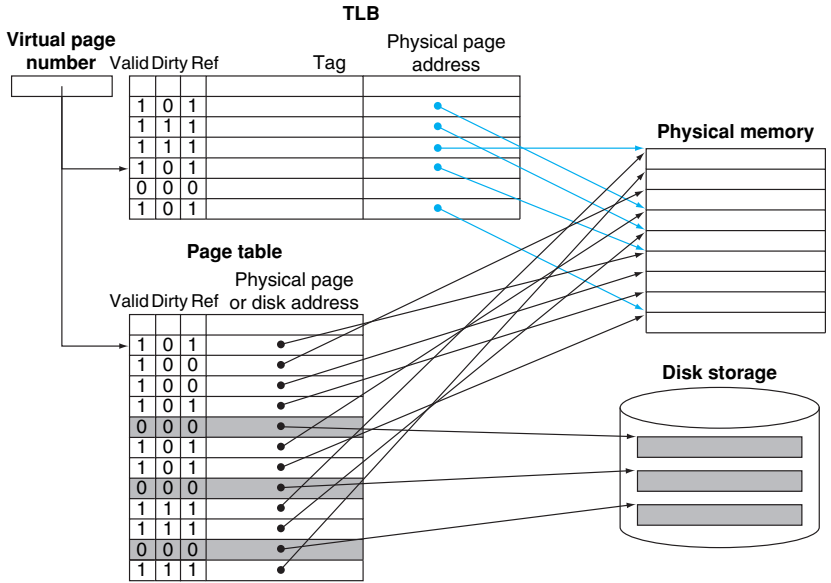
- Para reduzir a taxa de falhas de páginas, prefere-se em geral uma política LRU
  - Bit de referência (*reference bit*) é mantido na PTE e setado para 1 sempre que uma página é acessada
  - O SO periodicamente reseta o bits para 0
  - Uma página com o bit de referência 0, portanto, não foi usada recentemente
- Escritas no disco levam milhões de ciclos
  - Escritas ocorrem em blocos, não em posições individuais
  - Utilizar uma política de write-through é impraticável, logo, utiliza-se write-back
  - Bit sujo (*dirty*) na PTE é setado quando ocorrem escritas em uma página

Acelerando a tradução usando uma TLB

UFABC

- A tradução de endereços aparenta necessitar acessos adicionais à memória
  - Um acesso para ler a PTE
  - Um acesso para ler o dado propriamente dito
- Contudo, o acesso à tabela de páginas tem uma ótima localidade
  - Utilizamos, portanto, uma cache rápida apenas para as PTEs localizada dentro da CPU
  - Essa cache é chamada de Translation Look-aside Buffer (TLB)
  - Tamanho típico de 16-512 PTEs, 0.5-1 ciclo por hit, 10-100 ciclos por miss, 0.01%-1% miss rate
  - Misses podem ser tratadas tanto por software quanto por hardware

Acelerando a tradução usando uma TLB



Misses na TLB

UFABC

- Se a página estiver na memória
  - Carrega a PTE da memória e tenta novamente
  - Pode ser tratada em hardware
    - Pode ficar complexo se as estruturas da tabela de página forem complexas
  - Pode ser tratada em software
    - Levanta uma exceção para o SO que trata a miss
- Se a página não estiver na memória (*page fault*)
  - SO trata o problema carregando a página do disco e atualizando a tabela de páginas
  - Reinicia a execução da instrução que causou a page fault

Tratador de TLB Miss

UFABC

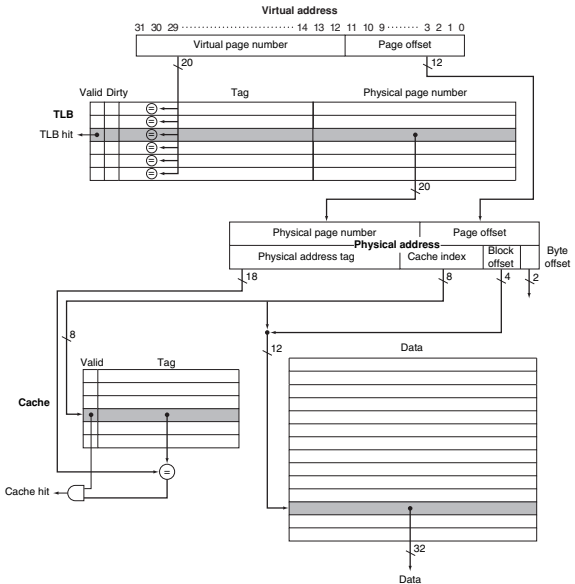
- TLB Miss indica
  - página presente, mas PTE não está na TLB
  - página não presente
- Deve reconhecer o TLB miss antes que o registrador de destino seja sobrescrito
  - Levanta Exceção
- Tratador copia a PTE da memória para a TLB
  - Então reinicia a instrução
  - Se a página não estiver presente, ocorre um page fault

Tratador de Page Fault

UFABC

- Utiliza o endereço virtual que causou a falha para encontrar a PTE
- Localiza a página no disco
- Escolhe a página a substituir
  - Se suja, escreve a página no disco antes
- Lê página para a memória e atualiza a tabela de páginas
- Reinicia a execução do processo
  - Recomeça a partir da instrução que causou a falha

Interação entre a TLB e a cache



- Se a tag da cache usa o endereço físico
  - É preciso traduzir o endereço antes de fazer a busca na cache
- Alternativa: usar o tag do endereço virtual
  - Pode complicar a implementação devido ao aliasing: múltiplos endereços virtuais para endereços físicos compartilhados

Exemplo: Intel i7-7500U

UFABC

```
$ cpuid | grep -i tlb
   cache and TLB information (2):
      0x63: data TLB: 2M/4M pages, 4-way, 32 entries
            data TLB: 1G pages, 4-way, 4 entries
      0x03: data TLB: 4K pages, 4-way, 64 entries
      0x76: instruction TLB: 2M/4M pages, fully, 8 entries
      0xb5: instruction TLB: 4K, 8-way, 64 entries
      0xc3: L2 TLB: 4K/2M pages, 6-way, 1536 entrie
      ...
```

Proteção da memória

UFABC

- Diferentes tarefas podem compartilhar partes dos seus endereços virtuais
  - Mas precisam ser protegidas de acessos não desejados
  - Necessita assistência do SO
- Suporte de hardware para proteção do SO
  - Modo de execução privilegiada - também conhecido como kernel mode
  - Instruções privilegiadas
  - Tabelas de páginas e outras informações de estado estão disponíveis apenas no modo kernel
  - System call exception (syscall no MIPS)

Resumo – A hierarquia de memória

UFABC

- Princípios comuns aplicados em todos os níveis da hierarquia de memória
  - Baseada em torno da ideia de caching
- Em cada nível da hierarquia
  - Política de disposição de blocos
  - Política de busca de blocos
  - Substituição de blocos em casos de misses
  - Política de escrita

Disposição dos blocos

UFABC

- Determinada pela associatividade
  - Mapeamento direto (associatividade de 1 via)
    - Opção única de disposição
  - Associativa em conjuntos de n-vias
    - n escolhas dentro de um conjunto
  - Completamente associativa
    - Qualquer posição
- Associatividade mais alta diminui a taxa de misses
  - Mas aumenta complexidade, custo e tempo de acesso

Localizando um bloco

Associatividade	Método de busca	Comparações
Mapeamento Direto	Índice	1
Associativa em conjuntos de n-vias	Índice (cjto), entradas no cjto	n
Completamente associativa	Busca todas as entradas Tabela de busca completa	Nº de entradas 0

- Caches de hardware
  - Reduzem as comparações para diminuir o custo
- Memória virtual
  - Tabela de busca completa permite o uso de uma estrutura completamente associativa
  - Se beneficia de uma taxa reduzida de misses

Substituição

- Escolha pela entrada a ser substituída após um miss
  - LRU
    - Complexo e custoso de fazer em hardware com alta associatividade
  - Aleatório
    - Próximo de LRU na prática, mais fácil de implementar
- Memória virtual
  - Aproximadamente uma LRU com suporte de hardware

Política de escritas

UFABC

- Write-through
  - Atualiza tanto os níveis mais altos quanto mais baixos
  - Simplifica a substituição, mas pode requerer um write buffer
- Write-back
  - Atualiza apenas os níveis mais altos
  - Atualiza o nível mais baixo quando o bloco é substituído
  - Precisa manter mais informações sobre o estado de cada bloco
- Memória virtual
  - Apenas write-back é viável, já que o tempo de aceso ao disco (latência) é muito alto

Fontes de misses

- Misses Compulsórios - ou cold start misses
  - Ocorrem no primeiro acesso a um bloco
- Misses por capacidade
  - Ocorrem pois a cache tem um tamanho finito...
  - ... e um bloco que havia sido carrego num momento anterior já foi substituído por necessidade de espaço
- Misses por conflito
  - Apenas possíveis em caches não completamente associativas
  - Ocorrem quando há disputa por entradas em um conjunto
  - Não ocorreriam em uma cache completamente associativa de mesmo tamanho (seria transformado em que tipo de miss?)

Trade-offs no design de caches

UFABC

Escolha de design	Efeito no miss rate	Desvantagem
Tamanho maior de cache	Diminui misses por capacidade	Pode aumentar o tempo de acesso
Maior associatividade	Diminui misses por conflito	Pode aumentar o tempo de acesso
Maior tamanho de bloco	Diminui misses compulsórios	Aumenta o miss penalty. Quando o tamanho do bloco for muito grande pode aumentar o miss rate devido à poluição da cache