Gerenciamento de memória

Leonardo Piccioni de Almeida
Matheus Preischadt Pinheiro

Introdução

- Desde o início da computação, houve a necessidade por mais memória do que disponível
- Principal solução? **Memória virtual**
  - Processos competem por memória
  - Memória entrega apenas quando há demanda verdadeira

Endereço virtual

- É constituído de duas partes
  - Número da página
  - Deslocamento (*offset*)
- Para converter de endereço virtual para físico:
  - Substitui número da página pelo endereço físico da página
    - Como? **Tabela de páginas**
  - Então efetua o deslocamento

Memória virtual

- Espaço de endereçamento maior do que memória disponível
- Proteção
  - Contra acesso por outros processos
  - Contra escrita (em algumas áreas)
- Mapeamento do endereço virtual para físico
- Fatiamento justo da memória
- Memória virtual compartilhada
  - Ex: bibliotecas
  - System V (IPC)

Modo de endereçamento físico

- Processos do *kernel* usam o modo de endereçamento físico
- SO gerenciando sua próprias tabelas de páginas seria um pesadelo!
- Endereços usados pelo *kernel* estão numa partição da memória principal acessível apenas em modo *kernel*.

Paginação

- A tradução de endereços virtuais para físicos é feito através de tabelas mantidas pelo SO
- Ambos os espaços de endereçamento são divididos em pedaços: **páginas**
- Para facilitar, o tamanho das páginas é idêntico em ambos os espaços
- Mapeamento de virtual para físico através de tabela de páginas

Paginação

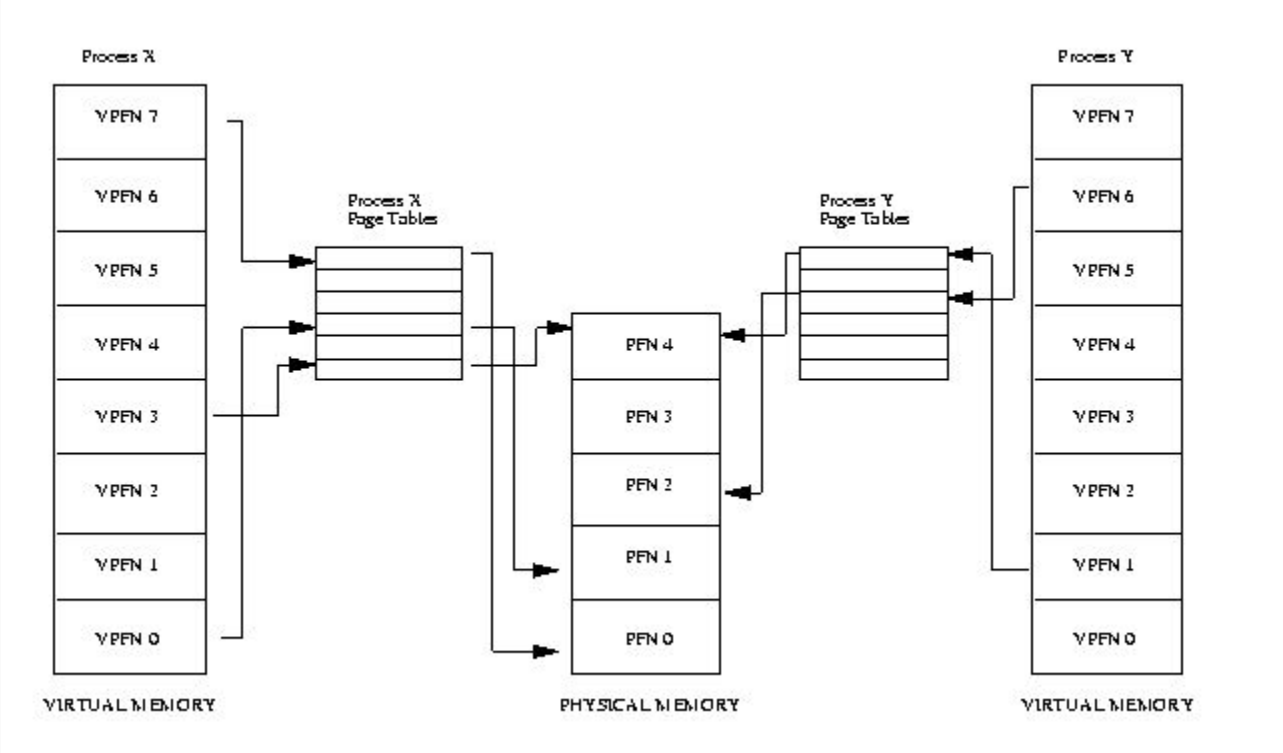

Process X
VPFN 7
VPFN 6
VPFN 5
VPFN 4
VPFN 3
VPFN 2
VPFN 1
VPFN 0
VIRTUAL MEMORY
Process X
Page Tables
PFN 4
PFN 3
PFN 2
PFN 1
PFN 0
PHYSICAL MEMORY
Process Y
Page Tables
Process Y
VPFN 7
VPFN 6
VPFN 5
VPFN 4
VPFN 3
VPFN 2
VPFN 1
VPFN 0
VIRTUAL MEMORY

Tabela de páginas

- Cada entrada contém:
  - Endereço físico da página
  - *Flag* de válido/inválido
  - Informações de acesso
    - Permite leitura ou somente escrita?
    - É executável?
- Índice: número da página

Tabela de páginas

- Processador utiliza a tabela de páginas para recuperar informação na memória física
- Se entrada não é válida, *page fault*
  - Se página não o pertence, sistema finaliza processo para evitar vazamento de memória
  - Senão, SO precisa trazer a página do disco
    - Requer tempo → bloqueante

Swapping

- Se SO precisa carregar página e não há páginas da memória física disponível, sistema precisa descartar outra para abrir espaço
- Se página descartada foi modificada, seu conteúdo vai para área de *swap*
- Um algoritmo de swap ruim causa *thrashing*
  - Linux usa LRU (*least recently used*)

Tabela de páginas

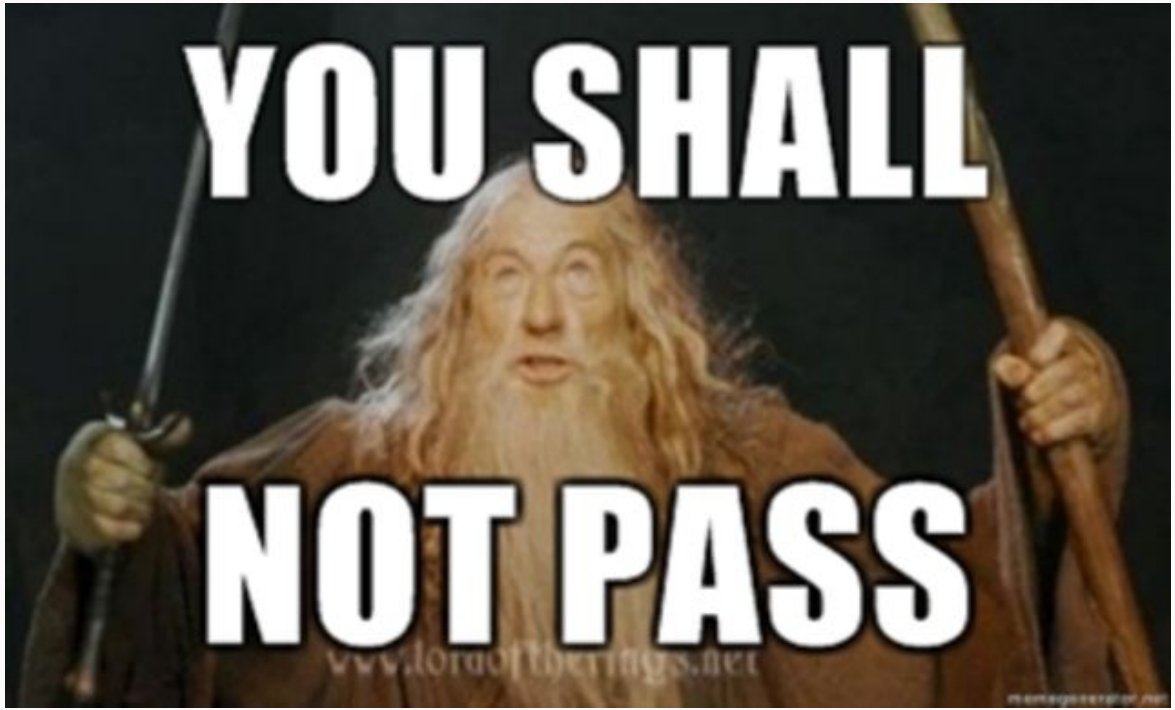
- Três níveis
- Endereço dividido em quatro partes:
  1. Tabela de nível 1

Memória virtual compartilhada

- Página aparece como entrada na tabela de páginas de dois ou mais processos
- Contador de processos

YOU SHALL NOT PASS!

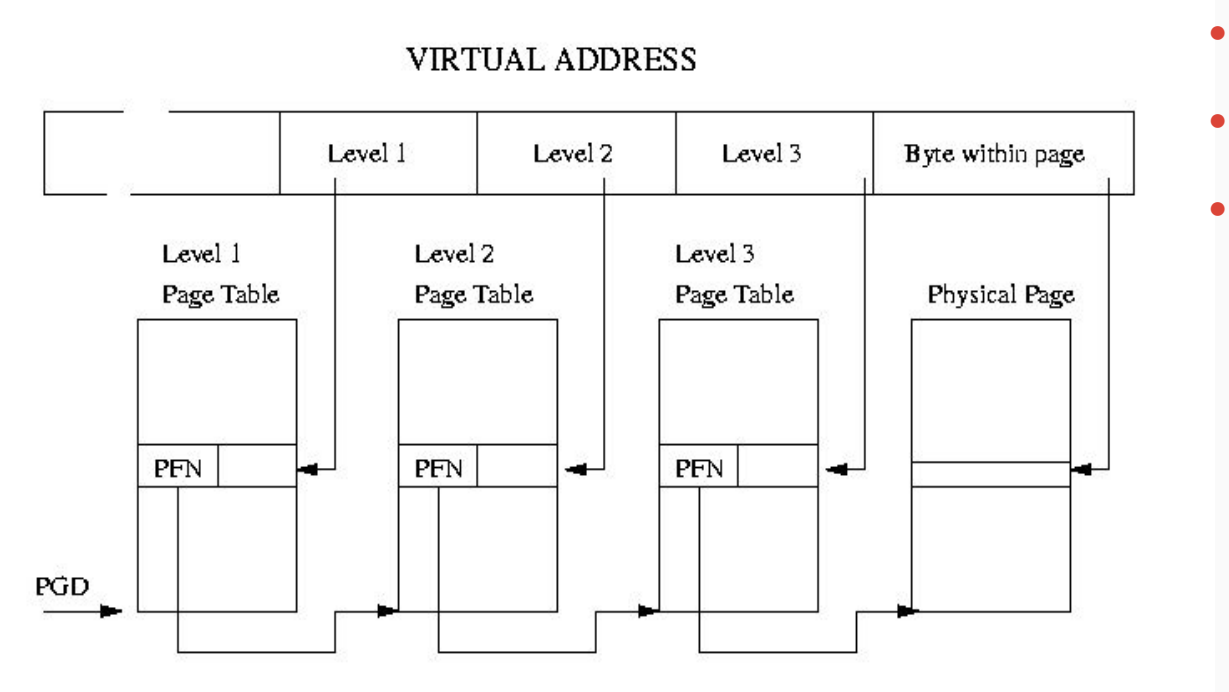
EITHER DIRECTIONS



Gerenciamento de memória em Linux

- Aplicação dos conceitos teóricos citados em anteriormente e vistos em aula no passado.
- Detalhes como tamanho da página, do offset e afins podem variar de acordo com a arquitetura do sistema.

Paginação no Linux



- 3 níveis de páginas, a anterior apontado para a próxima
- *Macros* de tradução são dadas pela plataforma
- Exemplo: Intel x86 e Alpha

Fonte: http://www.tldp.org/LDP/tlk/mm/memory.html

Paginação no Linux

- O kernel precisa saber quando uma página está sendo usada, quando pode ser alocada e afins
- Por esse motivo, as páginas do kernel são uma representação das páginas **físicas** da máquina, não **virtuais**
- Cada página física está diretamente ligada a uma instância da estrutura *page* do kernel

Paginação no Linux

```
struct page {
        page_flags_t flags;
        atomic_t _count;
        atomic_t  _mapcount;
        unsigned long private;
        struct address_space *mapping;
        pgoff_t index;
        struct list_head lru;
        void *virtual;
};
```

1. **flags:** uma série de flags de diferentes usos (dirty page, locked memory, etc)
2. **_count:** quantidade de *usos* da página (referências). Cursiodade: *page_count (page);*
3. **mapping:** ponteiro para um *adress_space* que indica se a página está sendo utilizada pelo cache
4. **virtual:** endereço virtual da página

Paginação no Linux

- O kernel atribui **zonas** às páginas alocadas
  - Páginas não podem ser tratadas todas da mesma maneira
  - Arquiteturas diferentes podem significar maneiras diferentes e tamanhos de memória diferentes na alocação de páginas
  - Zonas são costantes utilizadas para diferenciar as páginas, como explicado no próximo slide

Paginação no Linux

1. **ZONE_DMA:** Para páginas capazes de executar *Direct Memory Access*
2. **ZONE_NORMAL:** Páginas normais, mapeadas normalmente
3. **ZONE_HIGHMEM:** Páginas *high memory,* grandes de mais para serem mapeadas no *adress_space* do kernel permanentemente

Paginação no Linux

```
struct zone {
        spinlock_t lock;
        unsigned long free_pages;
        unsigned long pages_min;
        char *name;
};*
```

1. **lock:** flag que protege as estruturas de acesso concorrente (spin lock)
2. **free_pages**: quantidade de páginas nessa zona
3. **pages_free**: quantidade mínima para o campo *free_pages* quando possível (swapping)
4. **name**: nome da zona

*A estrutura é bem maior, foram removidos campos, deixados apenas os mais importantes.

Paginação no Linux

- O kernel aloca memória utilizando a função *alloc_pages* com a seguinte assinatura:
  - `struct page * alloc_pages(unsigned int gfp_mask, unsigned int order)`

- A função page_address converte uma página em seu endereço local
  - `void * page_address(struct page *page)`

Paginação no Linux

```
unsigned long pagina;

pagina = __get_free_pages(GFP_KERNEL, 3); //Aloca páginas e retorna o endereço da primeira.
if (!page) {
      return ENOMEM; //Memória insuficiente
}

//Variável pagina contem o endereço da primeira entre 8 páginas consecutivas.

free_pages(page, 3); //Paginas foram liberadas
```