Gerenciamento de memória

Leonardo Piccioni de Almeida
Matheus Preischadt Pinheiro

Primeiras Palavras

- 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

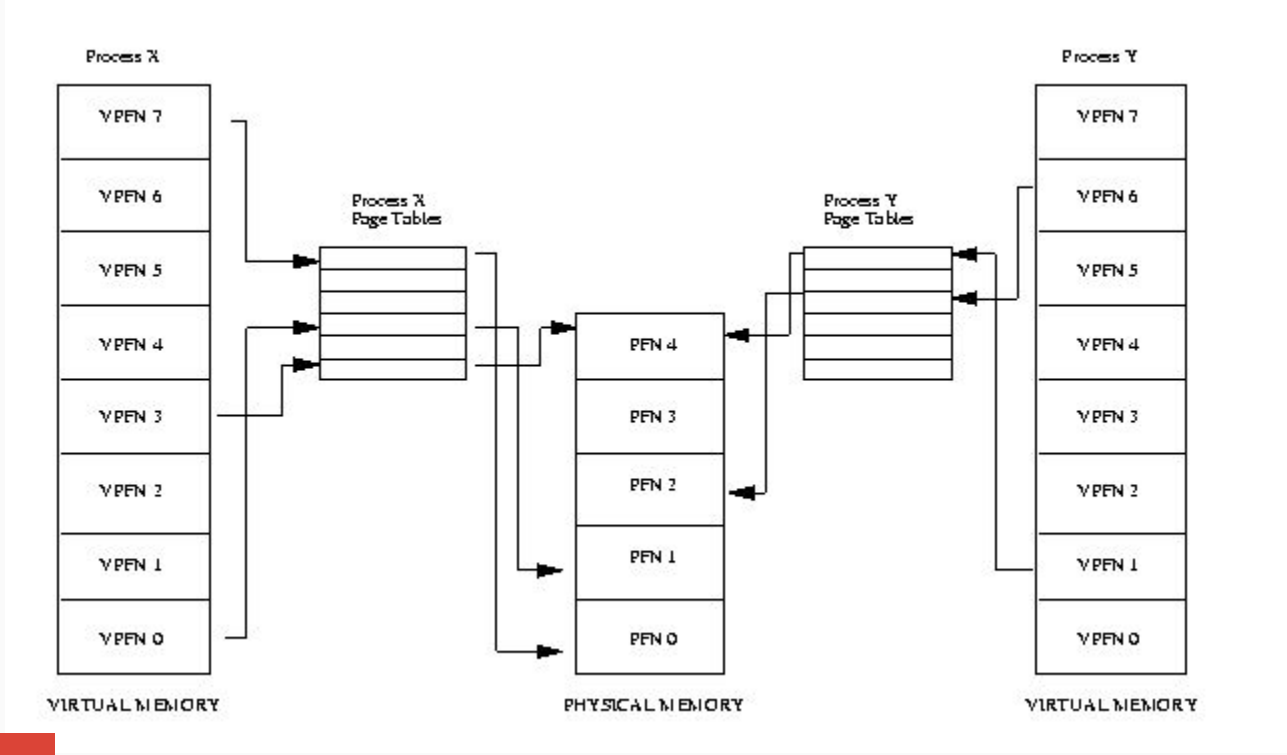
Memória virtual

- 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)

Conceitos Básicos

- 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

Conceitos Básicos



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

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*.

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

- Três níveis
- Endereço dividido em quatro partes:
  1. Tabela de nível 1
  2. Tabela de nível 2
  3. Tabela de nível 3
  4. Deslocamento na tabela de nível 3

Tabela de páginas

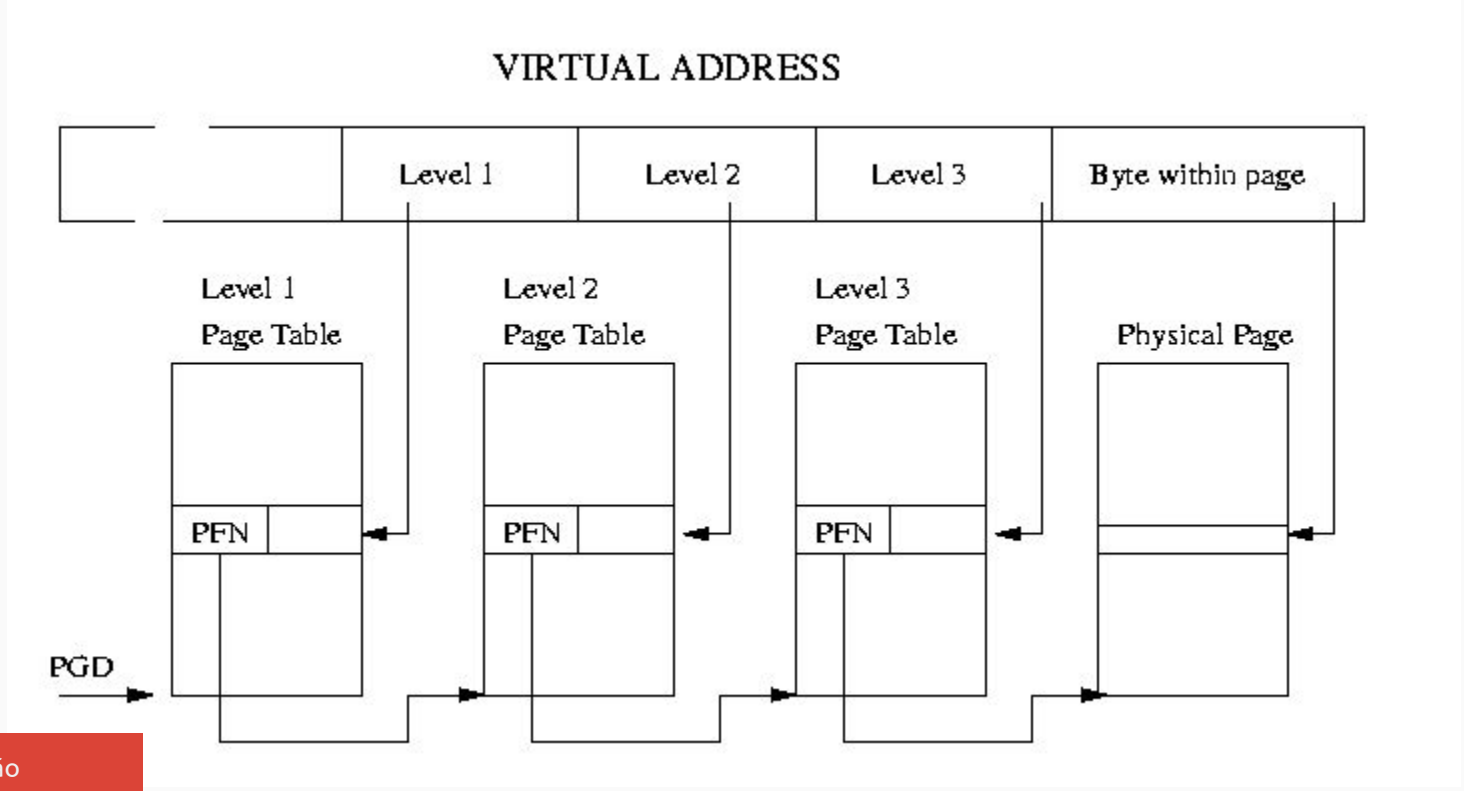


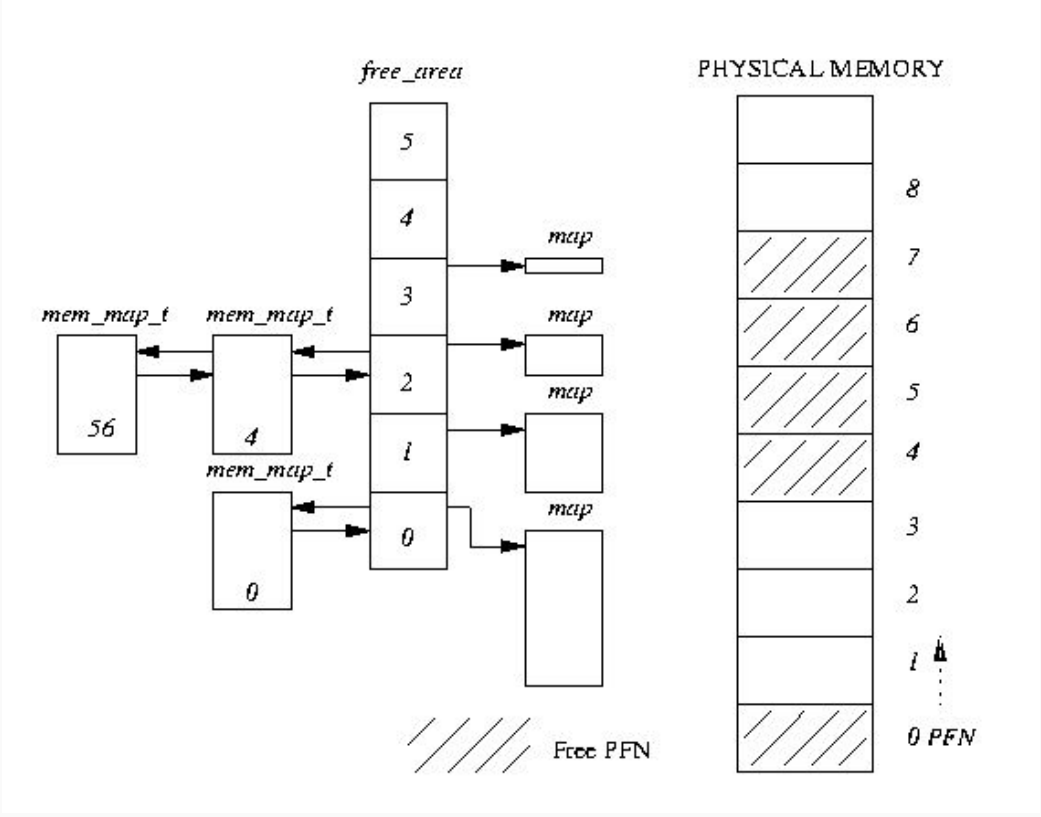
Tabela de páginas

- 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

Alocação de páginas

- Páginas livres são armazenadas no vetor `free_area`
- Cada posição *i* do vetor possui uma lista de blocos de páginas de tamanho 2^i
- Se não há bloco livres do tamanho desejado, tenta bloco com dobro de tamanho
  - A outra metade volta para o vetor de livres
- Na desalocação, se possível, blocos menores são recombinados

Alocação de páginas



Conceitos Básicos

- 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*)

Conceitos Básicos

- Quando memória física fica escassa, o *kernel* tenta liberar memória
- Varreduras periódicas em busca de três caminhos:
  - Reduzir o tamanho dos caches de páginas e *buffer*
  - Enviar páginas compartilhadas para área de *swap*
  - Enviar páginas descartadas para área de *swap*

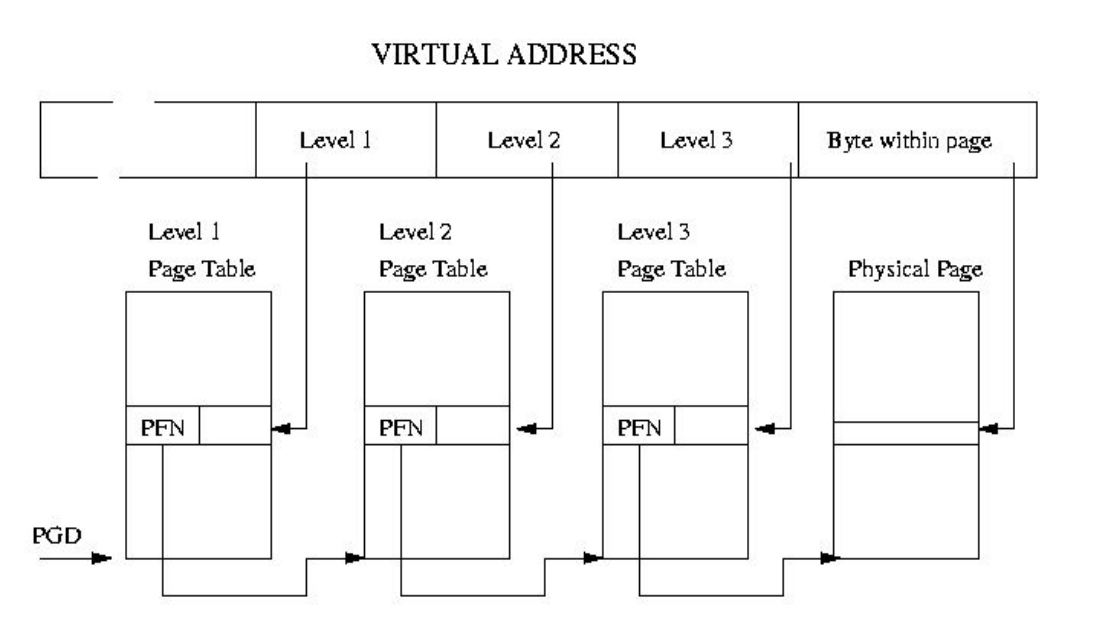
Caches

- Por questões de otimização, o Linux utiliza alguns caches:
  - Cache de *buffer*
  - Cache de páginas
  - Cache de *swap*
  - Cache a nível de *hardware*

Gerenciamento de memória

- 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.

Formato



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

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

Formato

- 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

Estrutura das Páginas

```
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). Curiosidade: `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

Zonas

- 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

Tipos de Zona

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

Estrutura das Zonas

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

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

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

Alocação

- 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)`

Exemplo

```
unsigned long pagina;

page = __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
```

Trivia

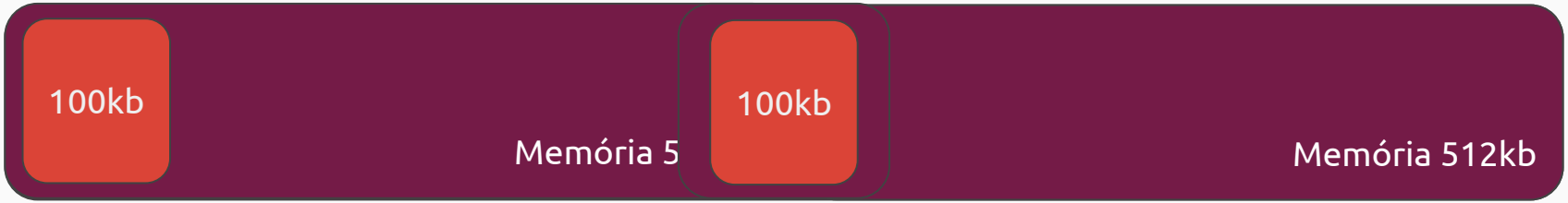
- **`kmalloc()`** para páginas consecutivas de baixo nível, função mais comum de alocação
- **`alloc_pages()`** para páginas `HIGH_MEMORY` citadas anteriormente
- **`vmalloc()`** para páginas virtualmente consecutivas

Algoritmo *Buddy*

O kernel se utiliza do algoritmo *buddy* para a separação das páginas:

- Tamanhos em ordem de 2
  a. Para ordens de 0 a 9, existem listas de áreas contendo 2^{ordem} páginas
  b. Se uma área menor é necessária e há apenas áreas *grandes* disponíveis, esta é separada em 2 *buddies* (repetidamente).
- Desperdício máximo de 50%

Exemplo



Tuning da memória virtual

- Acessaremos o diretório que contém o os arquivos que permitem o *tuning* da memória virtual
  - `$ cd /proc/sys/vm`
  - `$ ls -l`
- Note que todas as alterações que faremos daqui para frente podem causar instabilidade no sistema. Nesse caso, reverta as alterações.

Cache de páginas

- Verificaremos a porcentagem a partir da qual o cache de páginas é escrito no disco
  - `# sysctl vm.dirty_background_ratio`
- Aumentar esse número pode garantir *flushes* menos frequentes
  - `# sysctl -w vm.dirty_background_ratio=20`
- Em certas aplicações, como bancos de dados com grande volume e carga alta, pode-se desejar o contrário.

Cache de páginas

- Similarmente, gostaríamos de verificar quanto do cache de páginas um processo pode usar antes de ser bloqueado para *flush*
  - `# sysctl vm.dirty_ratio`
- Diminuiremos o teto, para melhor uso da memória
  - `# sysctl -w vm.dirty_ratio=25`

Swappiness

- A seguir, verificaremos quão frequentemente o sistema preferirá enviar a página para *swap* em vez de diminuir o cache
  - `# sysctl vm.swappiness`
- Se há memória suficiente, convém diminuir esse valor para 10 ou até mesmo 1.
- No entanto, se um processo grande dorme por muito tempo, talvez seja vantajoso mandá-lo para *swap*
  - `# sysctl -w vm.swappiness=100`

Persistência

- Para fazer as alterações permanentes, é preciso alterar o arquivo `/etc/sysctl.conf`
- Para que as alterações nesse arquivo sejam permanentes, rode o comando abaixo
  - `# sysctl -p`

Bibliografia

- **The Linux Kernel.** Rusling, A. David. © 1996-1999. Disponível em <http://www.tldp.org/LDP/tlk/tlk-toc.html>. Último acesso 3 de Junho de 2016.
- **Linux Tuning The VM Subsystem**. Gite, Vivek. © 2000-2016. Disponível em <http://www.cyberciti.biz/faq/linux-kernel-tuning-virtual-memory-subsystem/>. Último acesso 3 de Junho de 2016.
- **linux-insides**. Disponível em <https://github.com/0xAX/linux-insides>. Último acesso 3 de Junho de 2016.
- **The Linux Documentation Project.** Disponível em <http://tldp.org/>. Último acesso 3 de Junho de 2016.
- **Memory Management Reference.** *Ravenbrook Limited.* © 2016. Disponível em <http://www.memorymanagement.org/>. Último acesso 3 de Junho de 2016.