Aula 16

Memória Virtual: Linux

Gerência de Memória Física

- Como o usuário aloca memória?
  - Através do sistema de memória virtual

- Como o núcleo aloca memória?
  - Por exemplo, para uso interno, ou como o sistema de memória virtual aloca memória?
  - Como fazer para identificar blocos de memória disponíveis de tamanhos variáveis?

    - Através de um **buddy heap**

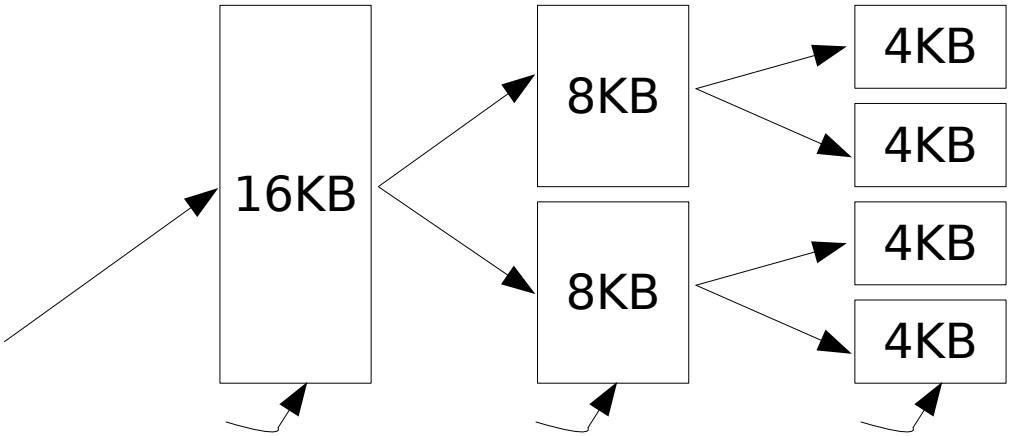
Buddy Heap (1)

Antes de mais nada, uma visita ao dicionário:

> **buddy** n. < 'b&dE > : 1. (Coloquial) **amigo** s.m.,f. 2. (Coloquial) **camarada** s.m.,f. 3. (Coloquial) **companheiro** s.m.,f.

Buddy Heap (2)

Páginas (ou conjuntos de páginas) são agrupados em blocos de 2:



Truque: uma lista encadeada **para cada tamanho** de bloco

- quando se quiser alocar bloco de tamanho **i**, busca-se um elemento na lista de tamanho imediatamente superior a **i**
- não esquecer de atualizar os blocos ascendentes ao escolhido

Caches do Núcleo

- Dois tipos de cache são mantidos no núcleo:
  - Buffer caches:
    - Guardam dados que vêm de dispositivos de bloco tais como discos
    - Assim, quando os dados chegarem ao sistema, eles podem ser guardados imediatamente (p.ex. por DMA), não é necessário esperar o escalonador escolher o processo destino dos dados.
  - Page caches:
    - Guardam páginas lidas do disco antes de elas serem movidas para seu destino final

Caches X Memória Virtual (1)

- A interação entre esses caches é estreita:
  - p.ex., um page fault (vindo da MV) se transforma em uma requisição para page cache
  - mas como essa página vem de um dispositivo de bloco, ela será colocada antes em um buffer cache.

Caches X Memória Virtual (2)

- Para evitar redundância de espaço e cópias, as páginas físicas são compartilhadas:
  - Mas isso não causa confusão?
  - Como saber se alguma página já foi usada e pode ser liberada ou se ainda existe alguém que precise dela?

Contadores de Referência (1)

- cria-se um contador c=0 (que surpresa...)
- a cada vez que uma entidade é criada, c++
  - entidades aqui são páginas
  - criar aqui é a criação da página por cada subsistema
- a cada vez que a entidade é destruída, c--
  - uma página é destruída quando o subsistema não precisa mais dela
- quando c==0, dealoca-se a entidade

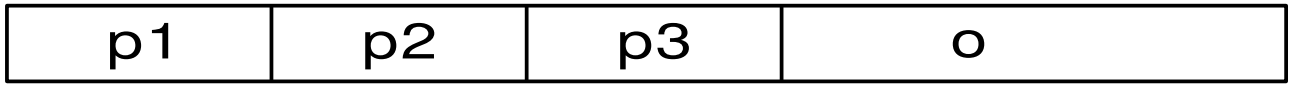
Contadores de Referência (2)

- Simples e eficiente, usado em 99,5% de todos os programas de computador:
  - Quem ainda não aprendeu a usar?
  - **Microsoft**, claro. Já tentaram desinstalar um programa no Windows 95?

Mapeamento Endereço Virtual -> Real (1)

- Paginação de 3 níveis:

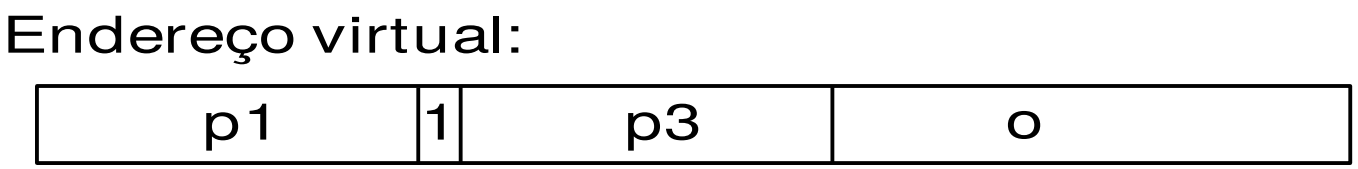
Endereço virtual:



- Motivo: é necessário por causa do espaço de endereçamento no Alpha (64 bits) – muito grande.

Mapeamento Endereço Virtual -> Real (2)

- Problema: os x86 só permitem paginação de 2 níveis.
- Solução: reduzir o nível do meio a um elemento só:



Diversos Sabores de Páginas

Páginas da memória virtual podem ser:

- Sem pistolão – **zero filled**: O primeiro acesso à página deve retornar uma página vazia
- Com pistolão – **backed**: Um arquivo é o "padrinho" da página. Acessos à página retornam páginas deste arquivo.
  - Arquivos mapeados em memória
  - Código

Diversos Sabores de Páginas

- Outra classificação:
  - Discretas – **private**: acesso por um único processo
  - Promíscuas – **shared**: acesso por vários processos.
    - Implementação:
      - permitir várias referências à mesma página;
      - usar reference count.

Compartilhamento de Memória

- Como fazer um fork?
  - Método ineficiente:
    - Criar um novo espaço de endereçamento – nova tarefa;
    - Copiar **todas** as páginas na memória para a nova tarefa.

  - Que horrível!... E se as duas tarefas compartilhassem as páginas?
    - Não dá, porque quando um processo modificar a memória, essa mudança seria visível ao outro.
    - Funciona para threads, por isso eles foram implementados

Copy on Write

- E que tal se a gente compartilhasse a página somente na leitura?
- As páginas marcadas com **copy-on-write** são:
  - Compartilhadas
  - Mas quando há uma escrita na página ela é **copiada** e marcada como private.
- Ou seja, um fork é feito simplesmente compartilhando as páginas:
  - basta copiar a tabela de páginas
  - escritas são tratadas de maneira automática
- O efeito é de compartilhamento máximo, tudo que pode ser compartilhado será:
  - Economia de memória e tempo (de cópia)

Compartilhar é preciso

- Copy-on-write compartilha por exemplo:
  - Código e dados de programas semelhantes **quando são "startados" hierarquicamente**
  - Ou quando o mesmo código (ou dados) é usado múltiplas vezes:
    - Pois o "backing store" é o mesmo
- Mas não compartilha quando, por exemplo:
  - Diversos programas usam subrotinas em comum

ELF x a.out

- O formato de binário **a.out** é antigo e usado por diversos Unix.
  - Mas todo código é “linkado” estaticamente, ou seja, todas as subrotinas têm que estar presentes no binário.
- O formato **ELF** usado por versões mais modernas do Linux é linkado dinamicamente:
  - Na hora da execução, o SO checa para ver se todas as subrotinas estão disponíveis e as carrega sob demanda
  - Isso permite que se implemente **shared libs**, ou seja, se a subrotina estiver na memória, o SO não carrega outra cópia

Shared Libs

- Cuidado a ser tomado:
  - Nenhuma shared lib pode fazer referência a endereços absolutos, todas as referências devem ser relativas.
- Por quê? Isso não tinha sido resolvido com paginação e segmentação?
  - Não, naquele caso os endereços podiam ser absolutos **dentro** do processo, pois o processo via a memória de forma contígua
  - Com shared libs, o mesmo código que está sendo executado pode ser visto por cada processo como estando em **endereços virtuais diferentes**.

Compartilhar é necessário

- Compartilhamento de memória permite um ganho significativo no desempenho.

- No Linux podemos identificar compartilhamento de memória:
  - Entre buffers de bloco e página;
  - Entre páginas através de copy-on-write;
  - Entre subrotinas sendo executadas.