Linux Memory Management

Presented By:
Rajesh Prodduturi
M.tech, cse

Under The Guidance Of:
Prof. D.B Phatak
Outline:

- Memory Management
  - Segmentation
  - Paging
  - zones
- Memory Allocation/ Deallocation
  - Buddy Algorithm
  - Slab Algorithm
- Page Cache
- Page frame Reclaiming Algorithm
Memory Management

- Segmentation in Linux: Linux prefers paging to segmentation. Because
  - Segmentation and paging are somewhat redundant
  - RISC architectures in particular have limited support for segmentation
  - The Linux 2.6 uses segmentation only when required by the 80x86 architecture.

- Four Main Linux Segments: Every process in Linux has these 4 segments
  - user code
  - user data
  - kernel code
  - kernel data
Segmentation in Linux:
Page Tables in Linux:
Page Tables in Linux (contd.)

- 4-level paging for both 32-bit and 64-bit
- 32-bit: two-level paging

It uses Page folding mechanism.

EX: 32-bit machine with page size 4KB
- Page Global Directory (10 bits)
- Page Upper Directory — 0 bits; 1 entry
- Page Middle Directory — 0 bits; 1 entry
- Page Table (10 bits)
- Page offset (12 bits)
RAM Classification:

ZONE_HIGHMEM
(High memory)

896 MB

ZONE_NORMAL
(Low memory)

16 MB

ZONE_DMA

0 MB
RAM Classification (contd.):

- Why we need ZONES:
  - Due hardware limitations in some architectures.
  - In some architectures DMA hardware can access 0-16MB address range in RAM.
  - No of zones in a system is dependent on hardware.
  - If DMA hardware can access any part of RAM, then we don't need DMA_ZONE.
  - Kernel address space is directly mapped into NORMAL_ZONE.
Memory Allocation/Deallocation:
Buddy Algorithm:

- It takes less time to allocate & deallocate
- Less external fragmentation
- It maintains different free lists with different sizes in each zone
  
  Ex: 4,8,16,32,64 group of free lists.
  
  i.e free list 4 contains (4 pages)

- Block size should be power of 2.
## Buddy Algorithm (contd.)

<table>
<thead>
<tr>
<th>Start</th>
<th>0</th>
<th>128k</th>
<th>256k</th>
<th>512k</th>
<th>1024k</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=70K</td>
<td>A</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td></td>
</tr>
<tr>
<td>B=35K</td>
<td>A</td>
<td>B</td>
<td>64</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>C=80K</td>
<td>A</td>
<td>B</td>
<td>64</td>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>A ends</td>
<td>128</td>
<td>B</td>
<td>64</td>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>D=60K</td>
<td>128</td>
<td>B</td>
<td>D</td>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>B ends</td>
<td>128</td>
<td>64</td>
<td>D</td>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>D ends</td>
<td>256</td>
<td>C</td>
<td>128</td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>C ends</td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td>512</td>
</tr>
<tr>
<td>end</td>
<td></td>
<td></td>
<td></td>
<td>1024k</td>
<td></td>
</tr>
</tbody>
</table>
Slab Algorithm

- It reduces internal fragmentation, inside pages.
- It works upon Buddy algorithm
- How it works:
  - It creates a cache for each frequently used kernel data structures.
    Ex: inode, task_struct, mm_struct etc.
  - Cache is a collection of slabs
  - Slab size may be one or two pages.
  - Slab contains a group of objects of similar type.
Slab Algorithm (contd)
Page cache:

- Linux logically divides physical memory into two types
  - One is Page cache
  - Another one is anonymous memory
- Linux allocates some pages to page cache from any zone.
- Size of page cache is dynamic
- It holds the contents of files etc.
- It provides sharing among different process.
- It uses copy-on-write mechanism
- All pages excluding page cache are anonymous memory
Page cache:

1. Two programs map scene.dat privately. Kernel deceives them and maps them both onto the page cache, but makes the PTEs read only.

2. Render tries to write to a virtual page mapping scene.dat. Processor page faults.

3. Kernel allocates page frame, copies contents of scene.dat #2 into it, and maps the faulted page onto the new page frame.

4. Execution resumes. Neither program is aware anything happened.
Page Frame Reclaiming Algorithm:

- If low memory, situation arises then kernel calls reclaiming algorithm
- Linux uses kswapd daemon for reclaiming frames.
- Each process maintains two lists
  - Active list: Holds frequently used pages
  - Inactive list: Holds less frequently used pages.
- Kswapd traverses each process and reclaimed pages until free memory size > some threshold
Page Frame Reclaiming Algorithm (contd)

- **Order of reclaiming frames:**
  - **Anonymous Memory:**
    - Non shared & non dirty
    - Non shared & dirty
    - Shared & non dirty
    - Shared & dirty
  - **Page cache:**
    - Non dirty
    - Dirty
- **In case of dirty pages, kswapd writes those dirty pages to swap space.**
References

- Gustavo Duarte, “Page Cache, the Affair Between Memory and Files”. Available at: http://duartes.org/gustavo/blog/category/internals/
- “Process address space”. Available at: http://kernel.org/doc/gorman/html/understand/understand007.html