Page Tables

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
The Problem

• Space overhead of page tables
  – A 32-bit address space with 4KB pages (4 bytes/PTE):
    \(2^{20} \times 4 = 4\text{MB}\) (per process)
  – A 64-bit address space with 8KB pages (8 bytes/PTE):
    \(2^{51} \times 8 = 2^{54} = 32\text{PB}\) (per process)

• How can we reduce this overhead?
  – Observation: Many invalid PTEs
  – Only need to map the portion of the address space actually being used which is a tiny fraction of entire address space
(Typical) Linear Page Table

Virtual address space

- code
- data
- heap
- heap
- stack

Physical memory

Huge waste!
Paging with Segmentation (1)

• A segment represents a region of valid address space
  – Segmentation:
    • Divide virtual address space into segments
    • Each segment can have variable length
  – Paging:
    • Divide each segment into fixed-sized pages
    • Each segment has a page table
    • Each segment tracks base (physical address) and limit of the page table for that segment

• Virtual address divided into three portions
Paging with Segmentation (2)

- Example: Multics address translation
Paging with Segmentation (3)

• Pros
  – Can decrease the size of page tables
  – Segments can grow without any reshuffling
  – Can run process when some pages are swapped to disk
  – Increases flexibility of sharing: share either single page or entire segment

• Cons
  – Page tables potentially can be large
    • e.g. A large but sparse-used heaps will have a lot of waste
  – External fragmentation due to page tables
    • Each page table should be allocated contiguously
Multi-level Page Tables (1)

Linear Page Table

<table>
<thead>
<tr>
<th>valid</th>
<th>prot</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rx</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>rx</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>rw</td>
<td>100</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>rw</td>
<td>86</td>
</tr>
<tr>
<td>1</td>
<td>rw</td>
<td>15</td>
</tr>
</tbody>
</table>

Multi-level Page Table

<table>
<thead>
<tr>
<th>valid</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>201</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>204</td>
</tr>
</tbody>
</table>

PDBR 200

<table>
<thead>
<tr>
<th>valid</th>
<th>prot</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>rx</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>rx</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>rw</td>
<td>100</td>
</tr>
</tbody>
</table>

The Page Directory

<table>
<thead>
<tr>
<th>valid</th>
<th>prot</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[Page 1 of PT: Not Allocated]

[Page 2 of PT: Not Allocated]
Multi-level Page Tables (2)

- Allow each page table to be allocated non-contiguously
- Virtual addresses have 3 parts

- Outer page table: outer page number → secondary page table
- Secondary page table: secondary page # → page frame #

<table>
<thead>
<tr>
<th>Outer page #</th>
<th>Secondary page #</th>
<th>Page offset</th>
</tr>
</thead>
</table>

logical address

$p_1$ $p_2$ $d$

$p_1$ → outer page table

$p_2$ → page of page table

d
Multi-level Page Tables (3)

- Example
  - 32-bit address space, 4KB pages, 4 bytes/PTE
  - Want every page table fit into a page
Multi-level Page Tables (4)

- Address translation in Alpha AXP architecture
  - Three-level page tables
  - 64-bit address space divided into 3 segments (coded in bits 63/62)
    - seg0 (0x): user code
    - seg1 (11): user stack
    - kseg (10): kernel
  - Alpha 21064
    - Page size: 8KB
    - Virtual address: 43 bits
    - Each page table is one page long
Multi-level Page Tables (5)

- Address translation in Intel 64 architecture
  - 48-bit “linear” address $\rightarrow$ 52-bit physical address (4KB page)
Multi-level Page Tables (6)

• Pros
  – Compact while supporting sparse-address space
    • Page-table space in proportion to the amount of address space used
  – Easier to manage physical memory
    • Each page table usually fits within a page
  – Easier for hardware to walk though page tables
  – No external fragmentation

• Cons
  – More memory accesses on a TLB miss
  – More complex than a simple linear page-table lookup
Inverted Page Tables (1)

• Reverse mapping from PFN $\rightarrow$ <VPN, PID>
  – One entry for each page frame in physical memory
  – Entry consists of the virtual page number with information about the process that owns that page
  – Need to search through the table to find match
  – Use hashing to limit the search to one, or at most a few, page-table entries

• Pros & Cons
  – Decrease memory needed to store page tables: No need to have per-process page tables
  – Increase time needed to search the table on a TLB miss
Inverted Page Tables (2)
Paging Page Tables

• Store page tables in virtual address space
  – Cold (unused) page table pages can be paged out to disk
  – But, now addressing page tables requires translation
  – Outer page table is usually pinned into physical memory
  – Outer page table points to the virtual addresses (in the kernel address space) of secondary page tables
  – Need to handle nested page faults

• What if we page kernel code and data too?
(cf.) IA-32 VM Architecture