Paging in x86 and TLB

COMS W4118

References: Operating Systems Concepts (9e), Linux Kernel Development, previous W4118s
Copyright notice: care has been taken to use only those web images deemed by the instructor to be in the public domain. If you see a copyrighted image on any slide and are the copyright owner, please contact the instructor. It will be removed.
x86 paging at a glance

• 1-level paging in x86-32 with 4KB pages possible?
  – 20-bit page number \( \rightarrow \) 1 million page table entries \( \rightarrow \) 4MB page table per process \( \rightarrow \) way too big!

• 2-level paging in x86-32
  – Default x86-32 paging mode: 10 + 10 + 12

• 3-level paging in x86-32
  – Physical Address Extension (PAE): 2 + 9 + 9 + 12
  – Still limited to 4GB per process but up to 64GB RAM

• 4-level paging in x86-64
  – Default x86-64 paging mode: 9 + 9 + 9 + 9 + 12

• 5-level paging in x86-64
  – Available in Intel Ice Lake processors: 9 + 9 + 9 + 9 + 9 + 12
  – CONFIG_X86_5LEVEL kernel option from Linux 4.14
Avoiding extra memory accesses

- **Observation**: locality
  - **Temporal**: access locations accessed just now
  - **Spatial**: access locations adjacent to locations accessed just now
  - Process often needs only a small number of vpn→ppn mappings at any moment!

- Fast-lookup hardware cache called **associative memory** or translation look-aside buffers (TLBs)
  - Fast parallel search (CPU speed)
  - Small
Paging hardware with TLB

The diagram illustrates the process of address translation in a paging system with a Translation Lookaside Buffer (TLB). The logical address is first checked in the TLB. If a hit is found, the physical address is returned. If a miss occurs, the page table is consulted to find the corresponding physical page, and the process continues as usual.
Effective access time with TLB

• Assume memory cycle time is 1 unit time
• TLB Lookup time = \( \varepsilon \)
• TLB Hit ratio = \( \alpha \)
  – Percentage of times that a vpn\( \mapsto \)ppn mapping is found in TLB

• Effective Access Time (EAT)

\[
EAT = (1 + \varepsilon) \alpha + (2 + \varepsilon)(1 - \alpha) \\
= \alpha + \varepsilon \alpha + 2 + \varepsilon - \varepsilon \alpha - 2\alpha \\
= 2 + \varepsilon - \alpha
\]
What happens to TLB on context switches?

Option 1: flush entire TLB
- x86
  - “load cr3” (load page table base) flushes TLB

Option 2: attach process ID to TLB entries
- ASID: Address Space Identifier
  - MIPS, SPARC

x86 “INVLPG addr” invalidates one TLB entry
Page sharing

process $P_1$

<table>
<thead>
<tr>
<th>ed 1</th>
<th>ed 2</th>
<th>ed 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

page table for $P_1$

process $P_2$

<table>
<thead>
<tr>
<th>ed 1</th>
<th>ed 2</th>
<th>ed 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

data 1

process $P_3$

<table>
<thead>
<tr>
<th>ed 1</th>
<th>ed 2</th>
<th>ed 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

page table for $P_2$

1

data 1

data 3

ed 1

ed 2

ed 3

data 2

data 2

data 3
Copy-On-Write (COW)

• In `fork()`, parent and child often share significant amount of memory
  – Expensive to copy all pages

• COW Idea: exploit VA to PA indirection
  – Instead of copying all pages, share them
  – If either process writes to shared pages, only then is the page copied

• Used in virtually all modern OS
Before Process 1 Modifies Page C
After Process 1 Modifies Page C