Date: 11/31/23

Lost of Address Translation (Paging)

- Single array: page # => frame #
- Large page: internal fragmentation
- Inverted page table: frame # => page #
- Multilevel page table

Page Table needs to allocate entry for each page

Page sharing the same frame
21 bit virtual address

\[ 0x1000 - 0x1fff \] code
\[ 0x8000 - 0x8fff \] stack

2 page tables needed to map just the code page (one 1st level + one 2nd level)
16 + 32 = 48 entries

3 page tables total when we map the stack page (one 1st level + two 2nd level)
16 + 32 + 32 = 80 entries

21 bit virtual address

\[ 0x1000 - 0x1fff \] code
\[ 0x8000 - 0x8fff \] stack

4KB page size
9 bits for page #
\[ 2^9 = 512 \text{ pages max} \]

Single level PT always allocate 512 entries

4 bits 5 bits 12 bits

1st level idx 2nd level idx offset

\[ \begin{array}{c}
0000 \\
0100
\end{array} \]

frame #
unallocated entries

frame #
unallocated entries

index 0

index 4

addr of 1st level PT

addr of 2nd level PT

addr of 2nd level PT

\[ 2^4 = 16 \text{ entries} \]

\[ 2^5 = 32 \text{ entries} \]
21 bit virtual address

<table>
<thead>
<tr>
<th>1st level PT entries</th>
<th>2nd level PT entries</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st level PT = $2^4 = 16$ entries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd level PT = $2^5 = 32$ entries</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Not always smaller than single-level

# of pages used (contiguous pages)
X86-64 Address Translation

- 4 level page table, each page table takes up a full page in memory

\[ 2^9 = 512 \text{ entries} \]

\[ 8 \text{ bytes per entry} \]

\[ 512 \times 8 = 4096 \]

\[ \text{x86-64} \text{ mm} \cdot \text{c} (\times 16) \]
<table>
<thead>
<tr>
<th>Bit Position(s)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (P)</td>
<td>Present; must be 1 to map a 4-KByte page</td>
</tr>
<tr>
<td>1 (R/W)</td>
<td>Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)</td>
</tr>
<tr>
<td>2 (U/S)</td>
<td>User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)</td>
</tr>
<tr>
<td>3 (PWT)</td>
<td>Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)</td>
</tr>
<tr>
<td>4 (PCD)</td>
<td>Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)</td>
</tr>
<tr>
<td>5 (A)</td>
<td>Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)</td>
</tr>
<tr>
<td>6 (D)</td>
<td>Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)</td>
</tr>
<tr>
<td>7 (PAT)</td>
<td>Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)</td>
</tr>
<tr>
<td>8 (G)</td>
<td>Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise</td>
</tr>
<tr>
<td>11:9</td>
<td>Ignored</td>
</tr>
<tr>
<td>(M-1):12</td>
<td>Physical address of the 4-KByte page referenced by this entry</td>
</tr>
<tr>
<td>51:M</td>
<td>Reserved (must be 0)</td>
</tr>
<tr>
<td>58:52</td>
<td>Ignored</td>
</tr>
<tr>
<td>62:59</td>
<td>Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise</td>
</tr>
<tr>
<td>63 (XD)</td>
<td>If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)</td>
</tr>
</tbody>
</table>
Page Fault

→ triggered on memory translation error: exception 14
   → page table walk encounters unallocated page table
   → page not mapped (page table entry's present bit not set)
   → mismatch permission (write to readonly page; kernel page accessed in user mode)

→ need to identify valid vs. invalid page faults

Demand paging  
  Stack growth  
  Low fork  
  Memory mapped files

Access to unallocated user address range

How does the kernel determine what kind of fault it is?

→ kernel has bookkeeping structures (machine independent) that track information about each page.
* Machine independent vs. machine dependent page table

(Lispacing, region, page-info)

\( 286 \, \text{64 pt} \rightarrow \text{info for address translation} \)

```c
struct vregion {
    enum vr_direction dir;       // direction of growth
    uint64_t va_base;            // base of the region
    uint64_t size;               // size of region in bytes
    struct vpi_page *pages;      // pointer to array of page_infos
};

struct vspace {
    struct vregion regions[NREGIONS]; // the regions for a process' virtual space
    pml4e_t *pgtbl;                // process' page table
};

struct vpage_info {
    short used;                   // whether the page is in use
    uint64_t ppn;                  // physical page number
    short present;                // whether the page is in physical memory
    short writable;               // does the page have write permissions
    // user defined fields
};
```