Multilevel Paging

- Each last level page table is a shard of the single array page table.

\[ \text{indirection takes up more space if most pages are used.} \]

\[ \text{TLB even more important!} \]
21 bit virtual address

4kB page size

9 bits for page #

\(2^9 = 512\) pages max

Single level PT always allocate 512 entries

2 page tables needed to map just the code page

\((\text{one 1st level + one 2nd level})\)

\(16 + 32 = 48\) entries

3 page tables total when we map the stack page

\((\text{one 1st level + two 2nd level})\)

\(16 + 32 \times 2 = 80\) entries

---

4 bits 5 bits 12 bits

1st level idx 2nd level idx offset

0000 0001 --

0111 1110 --

---

[0x1000-0xffff] code
[0xfee0-0xfffe] stack
X86-64 Page Table: each level of page table has $2^n$ entries & lives in a page & byte entry.

Virtual Address

- Sign Extend
- Page-Map Level-4 Offset (PML4)
- Page-Directory Pointer Offset
- Page-Directory Offset
- Page-Table Offset
- Physical-Page Offset

<table>
<thead>
<tr>
<th>Sign Extend</th>
<th>Page-Map Level-4 Offset (PML4)</th>
<th>Page-Directory Pointer Offset</th>
<th>Page-Directory Offset</th>
<th>Page-Table Offset</th>
<th>Physical-Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>47</td>
<td>3938</td>
<td>30 29</td>
<td>21 20</td>
<td>12 11 0</td>
</tr>
</tbody>
</table>

- 1001001--
- invalid memory format => general protection fault (13)

- all 0s or all 1s
- Page-Map Level-4 Table
- Page Directory Pointer Table
- Page Directory Table
- Page Table
- 4 Kbyte Physical Page

- access to an unallocated entry => page fault
- top level
- 2nd level
- 3rd level

- This is an architectural limit. A given processor implementation may support fewer bits

- CR3
### Bit Position(s)

<table>
<thead>
<tr>
<th>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 4.5); otherwise, reserved (must be 0)</td>
</tr>
</tbody>
</table>
Page Fault (page table walk or TLB)

- 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)

- Types of page faults
  - Demand paging (valid)
    - Stack growth
    - Heap growth
    - Memory mapped files
  - Permission Mismatch
    - New access
    - Actual violation (user access kernel address)
      - User access kernel address, need to read only page.

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

- Kernel has bookkeeping structures (machine independent) that track information about each page.
struct vspace {
    struct vregion regions[NREGIONS]; // the regions for a process' virtual space
        // process' page table
    pml4e_t* pgtbl;
};

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 vpi_page {
    struct vpage_info infos[VPIPPAGE]; // info struct for the given page
    struct vpi_page *next;             // the next page
};

linked list of vpage_info array

vspace_update() updates the
machine dependent
PT based on
machinindependent
region content
VK VM structures

```c
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
};
```

per page metadata