Paging

\[ \text{divide virtual & physical memory into fixed size pages} \]

\[ \text{page level translation} \]

\[ \begin{array}{c|c}
52\text{ bits} & 12\text{ bits} \\
\hline
\text{page} & \text{offset} \\
\hline
\end{array} \]

\[ \text{virtual addr} \]

\[ \begin{array}{c|c}
12\text{ bits} & \\
\hline
\text{frame} & \text{offset} \\
\hline
\end{array} \]

\[ \text{physical addr} \]

\[ \begin{array}{c|c}
\text{offset} & \text{remains the same} \\
\hline
\end{array} \]

\[ \begin{array}{c|c}
\text{page} & \text{frame} \\
\hline
A & 4 \\
\hline
\end{array} \]
Page Table: stores translations for every page (hw page table walk)

Kernel sets up the page table, translation performed by hw

Page Table lives in memory for every virtual memory access, we now have to make 2 physical memory access

Page Table base register

%cr3 in x86
Translation Lookaside Buffer (TLB) - caches the result of memory translation.

- Also caches the permission of the page mapping (may be out of sync from the pagetable).

Diagram:
- Virtual Address
  - Page# Offset
  - Translation Lookaside Buffer (TLB)
    - Hit
      - Virtual Page Page Frame Access
    - Matching Entry
    - Miss
      - Page Table Lookup

Physical Memory
- Frame Offset
  - Update TLB with new translation result.
The cost of single array page table

- per process data structure
- $2^{52}$ entries take up a lot of space

How to reduce the page table size?

- Larger page size: 2 MB & 1 GB page size (large super pages)

  \[
  \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
  \text{less pages, fewer entries} \quad \text{smaller page tables}
  \]

  \[2^{52} \Rightarrow 2^{44} \text{ entries}\]

but... 1 GB wasted

- internal fragmentation!
Maybe the problem is too many page tables?

→ Inverted page table: tracks frame mapping instead (global)

- Inverted page table indexed by frame:

<table>
<thead>
<tr>
<th>page#, pid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

→ How to look up given a virtual address?

  → search through each entry of the array until we find a matching page # to pid.

  works but very slow 😞

→ Use a hash function to place pages

  → lookup = hash(page#, pid) % frames => frame #

What about hash collisions?

  shared memory?

  cost of unmapped page?
Page ⇨ frame mapping still better for look up!

Multilevel Page Tables: use indirection to only allocate entries for pages in use