What if VA space is really big?

• 64 bit architectures: \(2^{64}\) bytes of VA space
  ~ 7 levels of Page Tables
  Effective Access Time is way too slow
  Multiple Pi’s PTs take up way too much of mem

• Solution: Inverted Page Tables
  – Idea: instead of mapping from VA to frame #
    keep mappings of frame# to VA
  – PA space much smaller than VA space
    inverted PT small (one entry per page FRAME)
  – Single, global PT for all processes in the system
Inverted Page Table

Effective Access Time = h(TLB time) + (1-h)((#search refs)(mem time) + TLB time)
• Problem: What if page isn’t in memory
  ➔ an entry for it is not in inverted PT
• Solution: We still need regular per-process PTs, but should rarely need to use them
• Problem: What if 2 or more processes share a physical page of memory
  (ex. system V shared memory)
  – For regular page tables this is not a problem
    • each sharing process’ PT entry points to same physical frame
  – For inverted PTs we now need a list of info in each PT entry (pid1, page#), (pid2, page#), (pid3, page3) …
    OR we just keep one Pi’s mapping in PT and handle other mappings as if the page was not in memory
    • Neither of these solutions is ideal
      more mem refs to either traverse a list or to traverse reg. PT levels
Segmentation

Another way to break-up a Pi’s address space
– Better fits the way in which we view Pi’s address space

Address space broken up into logical parts (segments):
– segment for heap, stack, data, code, …
– each segment is a logically contiguous chunk of Pi
– Can set different permissions on different segments
  • e.g. read only access to code segment
– Each segment can be a different size
– Sharing parts of address space is easier
  • Put all objects want to share in the same segment
    – HW will check protections and length of shared segments
  • e.g. two running emacs programs can share same code segment

• Need HW support: segment table base & limit regs
However: problem of variable sized contiguous segments in memory
⇒ External Fragmentation
Solution: Paged Segmentation

Example: Multics (1970 MIT)

VA:

<table>
<thead>
<tr>
<th>seg #</th>
<th>offset: p</th>
<th>d'</th>
</tr>
</thead>
</table>

STBR

Segment Table Entry

seg len | seg PT base addr

Page Table for seg #

Can also page the segment table:

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
</table>

s1: index into PT for seg, contains segment table page address
s2: index to get Seg. Table entry
d1: index into PT for segment get f
d2: offset into physical page