Lectures 22

Virtual Memory cont

- Because different processes will have different mappings from virtual to physical addresses, two programs can freely use the same virtual address.
- By allocating distinct regions of physical memory to A and B, they are prevented from reading/writing each other's data.

Virtual Memory

V.A. \rightarrow P.A.

Page Table picture

- Each entry contains a valid bit, and a physical page number (PPN)
- The PPN is concatenated with the page offset to get the physical address
- No tag is needed because the index is the full VPN.

Finding the right page

- If it is fully associative, how do we find the right page without scanning all of memory?
  - Use an index, just like you would for a book.
- Our index happens to be called the page table:
  - Each process has a separate page table
  - A "page table register" points to the current process's page table
  - The page table is indexed with the virtual page number (VPN)
  - The VPN is all of the bits that aren't part of the page offset.
- Each entry contains a valid bit, and a physical page number (PPN)
- The PPN is concatenated with the page offset to get the physical address
- No tag is needed because the index is the full VPN.

How big is the page table?

- From the previous slide:
  - Virtual address space = 32 bits
  - Physical page number is 12 bits = valid bit + round up to 32 bits (4 MB)

- How about for a 64b architecture?
  - 64 bits = \( 2^{6} \times 4 = 4 \) MB
  - 52 bits = \( 2^{5} \times 4 = 16 \) MB

Dealing with large page tables

- Multi-level page tables
  - "Any problem in CS can be solved by adding a level of indirection" (Yttrium)
  - Page Table
  - Base Pointer
  - 3rd level page table

- Why does this work?
  - Processes do not need entire V.A. space.
Memory Protection

- In order to prevent one process from reading/writing another process's memory, we must ensure that a process cannot change its virtual-to-physical translations.
- Typically, this is done by:
  - Having two processor modes: user & kernel.
  - Only the O/S runs kernel mode.

Sharing Memory

- Paged virtual memory enables sharing at the granularity of a page, by allowing two page tables to point to the same physical addresses.
- For example, if you run two copies of a program, the O/S will share the code page between the programs.

Waitaminute!

- We've just replaced every memory access V[N][E] with:
  - \( \text{INDEX}(\text{TAG}[	ext{INDEX} + \text{offset}]) \)
- i.e., a memory access...
- And we haven't talked about the bad case yet (i.e., page faults)...

  "Any problem in CS can be solved by adding a level of indirection"  
  except too many levels of indirection...

- How do we deal with too many levels of indirection?

\[
\text{Cache} \quad \text{VA} \rightarrow \text{PA}
\]
Summary

- Virtual memory is great:
  - It means that we don’t have to manage our own memory.
  - It allows different programs to use the same memory.
  - It provides protection between different processes.
  - It allows controlled sharing between processes (albeit somewhat
    inefficiently).
- The key technique is indirection:
  - Yet another classic CS trick you’ve seen in this class.
  - Many problems can be solved with indirection.
- Caching made a few appearances, too:
  - Virtual memory enables using physical memory as a cache for disk.
  - We used caching (in the form of the Translation Lookaside Buffer) to
    make Virtual Memory’s indirection fast.