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.

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:
  - $V.A. \space space \Rightarrow \space 32 \space bits$
  - $P.A. \space space \Rightarrow \space 512 \space bits$
  - \[ (2 \times 4) = 8 \space MB \]
  - $\text{Page size} \Rightarrow \text{4 bytes}$
  - $\text{Virtual address} \Rightarrow \text{4 bytes}$
  - $\text{Physical address} \Rightarrow \text{4 bytes}$
  - $2^{12} = 4 \space KB$ physical address space

- How about for a 64-bit architecture?
  - $64 \times 32 \space bits \Rightarrow \space V.P.N.$
  - $32 \times 5 = 160 \space bits$

Virtual Memory

- $V.A. \rightarrow P.A.$

Page Table picture

Dealing with large page tables

- Multi-level page tables
  - "Any problem in CS can be solved by adding a level of indirection"
  - Why does this work?
    - Processes do not care about the $V.A.$ space.
Caching Translations

- Virtual to Physical translations are cached in a Translation Lookaside Buffer (TLB).

What about a TLB miss?

- If we miss in the TLB, we need to “walk the page table”
  - In HP9000, an exception is raised and software fills the TLB
  - In x86, a “hardware page table walker” fills the TLB

- What if the page is not in memory?
  - This situation is called a page fault.
  - The operating system will have to request the page from disk.
  - It will need to select a page to replace.
  - The O/S tries to approximate LRU (see CS451)
  - The replaced page will need to be written back if dirty.

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 in kernel mode.
  - Only allowing kernel mode to write to the virtual memory state, e.g.,
    - The page table
    - The page table base pointer
    - The TLB

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.
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 inflexibly).

- 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.