Address Translation

February 24, 2013
Anthony D. Joseph
http://inst.eecs.berkeley.edu/~cs162
Important Aspects of Memory Multiplexing

- **Controlled overlap:**
  - Processes should not collide in physical memory
  - Conversely, would like the ability to share memory when desired (for communication)

- **Protection:**
  - Prevent access to private memory of other processes
  - Different pages of memory can be given special behavior (Read Only, Invisible to user programs, etc.)
  - Kernel data protected from User programs

- **Translation:**
  - Ability to translate accesses from one address space (virtual) to a different one (physical)
  - When translation exists, process uses virtual addresses, physical memory uses physical addresses
Multi-step Processing of a Program for Execution

• Preparation of a program for execution involves components at:
  – Compile time (i.e., “gcc”)
  – Link/Load time (UNIX “ld” does link)
  – Execution time (e.g., dynamic libs)

• Addresses can be bound to final values anywhere in this path
  – Depends on hardware support
  – Also depends on operating system

• Dynamic Libraries
  – Linking postponed until execution
  – Small piece of code, stub, used to locate appropriate memory-resident library routine
  – Stub replaces itself with the address of the routine, and executes routine
Simple Base and Bounds (CRAY-1)

- Could use base/limit for **dynamic address translation** (often called “segmentation”) – translation happens at execution:
  - Alter address of every load/store by adding “base”
  - Generate error if address bigger than limit
- This gives program the illusion that it is running on its own dedicated machine, with memory starting at 0
  - Program gets continuous region of memory
  - Addresses within program do not have to be relocated when program placed in different region of DRAM
Implementation of Multi-Segment Model

- Segment map resides in processor
  - Segment number mapped into base/limit pair
  - Base added to offset to generate physical address
  - Error check catches offset out of range
- As many chunks of physical memory as entries
  - Segment addressed by portion of virtual address
  - However, could be included in instruction instead:
    - x86 Example: mov [es:bx],ax.
- What is “V/N” (valid / not valid)?
  - Can mark segments as invalid; requires check as well
5min Break
How to Implement Paging?

• Page Table (One per process)
  – Resides in physical memory
  – Contains physical page and permission for each virtual page
    » Permissions include: Valid bits, Read, Write, etc

• Virtual address mapping
  – Offset from Virtual address copied to Physical Address
    » Example: 10 bit offset ⇒ 1024-byte pages
  – Virtual page # is all remaining bits
    » Example for 32-bits: 32-10 = 22 bits, i.e. 4 million entries
    » Physical page # copied from table into physical address
  – Check Page Table bounds and permissions
Multi-level Translation

- A tree of tables
  - Lowest level page table \( \Rightarrow \) memory still allocated with bitmap
  - Higher levels often segmented
- Could have any number of levels. Example (top segment):

  - What must be saved/restored on context switch?
    - Contents of top-level segment registers (for this example)
    - Pointer to top-level table (page table)
Implementation of Multi-Segment Model

- Segment map resides in processor
  - Segment number mapped into base/limit pair
  - Base added to offset to generate physical address
  - Error check catches offset out of range
- As many chunks of physical memory as entries
  - Segment addressed by portion of virtual address
  - However, could be included in instruction instead:
    - x86 Example: mov [es:bx],ax.
- What is “V/N” (valid / not valid)?
  - Can mark segments as invalid; requires check as well
How to Implement Paging?

• Page Table (One per process)
  – Resides in physical memory
  – Contains physical page and permission for each virtual page
    » Permissions include: Valid bits, Read, Write, etc

• Virtual address mapping
  – Offset from Virtual address copied to Physical Address
    » Example: 10 bit offset ⇒ 1024-byte pages
  – Virtual page # is all remaining bits
    » Example for 32-bits: 32-10 = 22 bits, i.e. 4 million entries
    » Physical page # copied from table into physical address
  – Check Page Table bounds and permissions
Two-level page table

Physical Address:

Virtual Address:

PageTablePtr

- Tree of Page Tables
- Tables fixed size (1024 entries)
  - On context-switch: save single PageTablePtr register
- Valid bits on Page Table Entries
  - Don’t need every 2nd-level table
  - Even when exist, 2nd-level tables can reside on disk if not in use
Inverted Page Table

• With all previous examples ("Forward Page Tables")
  – Size of page tables is at least as large as amount of virtual memory allocated to ALL processes
  – Physical memory may be much, much less
    » Much of process’ space may be out on disk or not in use

• Answer: use a hash table
  – Called an “Inverted Page Table”
  – Size is independent of virtual address space
  – Directly related to amount of phy mem (1 entry per phy page)
  – Very attractive option for 64-bit address spaces (IA64, PowerPC, UltraSPARC)

• Cons: Complexity of managing hash chains in hardware
<table>
<thead>
<tr>
<th>Address Translation Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
</tr>
<tr>
<td><strong>Segmentation</strong></td>
</tr>
<tr>
<td><strong>Paging (single-level page)</strong></td>
</tr>
<tr>
<td><strong>Paged segmentation</strong></td>
</tr>
<tr>
<td><strong>Two-level pages</strong></td>
</tr>
<tr>
<td><strong>Inverted Table</strong></td>
</tr>
</tbody>
</table>