Virtual Memory
Announcements

• Lab 4 report deadline extended by a week
Main Memory Organization

- **Assumptions**
  - Byte addressable
Main Memory Organization

- **Assumptions**
  - Byte addressable
  - 2 bytes per word
Main Memory Organization

- **Assumptions**
  - Byte addressable
  - 2 bytes per word
  - 4 bytes per block
Load Word (LW) from Addr 0 to R0

- **Assumptions**
  - Byte addressable
  - 2 bytes per word
    - Each register holds a word
  - 4 bytes per block
Extending and Sharing Main Memory?

- What if one program needs more than the amount of installed main memory (i.e., physical memory)?

- How do multiple programs share the same main memory address space (multiprogramming / multitasking)?
Extending Memory Hierarchy

- Main memory (MM) is managed similar to a cache
  - Data are brought into MM as requested
  - If MM is full, older data get swapped out to disk

<table>
<thead>
<tr>
<th>Memory Level</th>
<th>Amount (X)</th>
<th>Speed Relative to MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Cache (KB)</td>
<td>~2-5X</td>
<td>Slower than L2 Cache</td>
</tr>
<tr>
<td>L2 Cache (MB)</td>
<td>~10-50X</td>
<td>Slower than MM</td>
</tr>
<tr>
<td>Main Memory (GB)</td>
<td>100-10000X</td>
<td>Slower than L2 Cache</td>
</tr>
<tr>
<td>Permanent Disk Storage (TB)</td>
<td>100-10000X</td>
<td>Slower than MM</td>
</tr>
</tbody>
</table>
Sharing Main Memory

• How to enable multiple programs to share the same physical MM?

Requirements
– *Transparency*: a program should not know other programs are sharing the same MM
– *Protection*: a program must not be able to corrupt other programs

Solutions (Virtualizing MM)
– Each program operates in its own virtual address space
– The set of physical MM addresses for each program is dynamically allocated and managed
Virtual Memory

• The *hardware and software* mechanisms that dynamically manage the memory hierarchy

• Extends memory hierarchy to incorporate large permanent storage
  – Hide physical size of MM from the program
  – Moves large blocks (in unit of pages) between MM and permanent storage as needed

• Allows multiple programs to share the main memory and provides protection among programs
  – Programs run in virtual address space
Multiprogramming with Virtual Memory

- In this example, each application starts at virtual address 0
- Without virtual memory, could have only one program in MM at a time
Virtual and Physical Addresses

• When a program is compiled, the instruction and data addresses are **virtual**
  – They need to translated to the **physical** addresses

• **Virtual addresses** refer to the addresses used by the programs
  – With a N-bit virtual address, the size of the virtual address space is $2^N$ bytes

• **Physical addresses** refer to the real addresses used by hardware to access the physical MM
  – With a M-bit physical address, the size of the physical address space is $2^M$ bytes (typically, $M < N$)
Paging

- Virtual/physical address space is divided into equal sized pages
  - A page contains N bytes where N is a power of 2
    - N = 4096 is a typical size
  - A whole page is read or written during data transfer between MM and disk

- Each page in virtual memory space has a unique index called virtual page number (VPN)
- Similarly, each page in physical memory space has a unique physical page number (PPN)
View of Virtual Memory with 32b Address

Assuming 32-bit virtual addresses
(4 GB of virtual address space)

Page size = 4KB
Virtual Memory and Physical MM

- During a program execution, only a subset of its virtual pages need to be in physical MM at a time
  - When requested, an entire page is brought into a physical MM location
  - The correspondence (mapping) between virtual to physical pages is saved in a directory (*page table*)
    * When the same virtual address is encountered, it is *translated* using this saved mapping information in the directory
Address Translation

Assuming 1GB physical memory here
(30-bit physical address)
Address Translation Using a Page Table

Page table register

Virtual address

31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Virtual page number Page offset

VPN analogous to the index for a cache

Page table stored in MM

If 0 then page is not present in memory

Valid

Physical page number

Page table entries (PTEs)

Physical page number Page offset

Special CPU register for locating the page table

Indicates that a page contains $2^{12} = 4$ KB

Lecture 24: 17
Page Table Operation (1)

The page table register (PTR), a special register in CPU, holds the base (physical) address of the page table in MM.

PTR points to the very first page table entry (PTE)
Page Table Operation (2)

PTR+VPN form the physical address of the page table entry (PTE) to access
If Valid = 1, then the PPN and Page Offset are concatenated to form the physical address.
If Valid = 0, a miss (page fault) has occurred, and the page is read from disk into MM (replacing another page if the MM is full)
Page Faults and Page Replacement

• Miss penalty on a page fault is significant
  – Up to ~100M cycles

• Low miss (page fault) rates are essential
  – Fully associative page placement (put anywhere in MM)
  – LRU replacement of a page when MM is full

• The Operating System (OS) handles page placement
Example: Page Table Access

- Given the following page table and virtual address stream (in decimal), identify the potential page faults 128, 2048, 4096, 8192

<table>
<thead>
<tr>
<th>Valid</th>
<th>Physical Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Page Table
Example: Page Table Access

- Given the following page table and virtual address stream (in decimal), identify the potential page faults:
  - 128 (VPN=0), 2048 (VPN=0), 4096 (VPN=1), 8192 (VPN=2)

<table>
<thead>
<tr>
<th>Valid</th>
<th>Physical Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Page Table

Virtual memory

Lecture 24: 24
Page Replacement and Write Policy

• Too expensive to do true LRU (100K-1M pages); Use LRU approximation
  – Each PTE has a Reference bit (ref)
  – Reference bit is set when a page is accessed
  – OS periodically clears all Reference bits
  – OS chooses a page with a Reference bit of 0

• Write back policy is used (instead of write through)
  – Dirty bit in PTE is set on a write to main memory
  – Page with set Dirty bit is written to disk if replaced
Faster Address Translation

• Must access the page table before an instruction can be fetched and before data cache/memory can be accessed

• Page table accesses have good locality
  ⇒ Cache the most recent PTEs within the CPU
Translation Lookaside Buffer (TLB)

- Small cache of recently accessed PTE (typically 16-512 entries, fully associative)

Diagram:
- TLB:
  - Virtual page number
  - Tag: virtual page number
  - Valid Dirty Ref
  - Data: physical page number

- Page table:
  - Valid Dirty Ref
  - Physical page or disk address

- Physical memory

- Disk storage

Accessed only on a TLB miss
TLB Miss Scenarios

- **TLB miss, page table hit**
  - Bring in the PTE information from page table to TLB
  - Retry the access
  - Usually completely performed by hardware

- **TLB miss, page fault**
  - Bring in the page from disk (orchestrated by OS)
  - Load the page table and TLB (orchestrated by OS)
  - Retry the access
    - Cache miss will definitely occur!
Before Next Class

• H&H 6.7.2, 7.7

Next Time

Exceptions
Inputs/Outputs