Virtual Memory
Announcements

• Final exam: Saturday May 11, 2pm @ PHL 101, 100 mins
  – Cumulative, with an emphasis on material from Lecture 17 onward
    • closed book, closed notes, closed Internet
  – Inform instructor about schedule conflict before Thursday 5/2
  – Sample final is posted on CMS
  – More detail on review session and OHs will be announced soon
Review: Performance Tradeoff (Example 2)

Complex Instruction vs. Simple Instruction

ADD 0(R2),0(R1),4(R1)  vs.  LW  R3,0(R1)
ADD  R3,R3,R4
SW  R3,0(R2)

1 cycle  4 cycles

Does Amdahl’s Law apply in this context?

Note: Other advanced computer architecture courses will delve into additional differences and trade-offs between CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer).
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

![Memory Hierarchy Diagram]
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
Virtual Memory Layout of a Program*

* Supplementary material (not included in the final exam)

**Code Section** (or Text) contains the executable code of the program, i.e., instructions to be executed by the processor.

**Initialized Data Section** contains initialized global & static variables used by the program.

**Uninitialized Data Section** contains uninitialized global & static variables.

**Heap** is a memory area where memory is dynamically allocated & deallocated at runtime.

**Stack** is used for local variables and function call information, which grows & shrinks dynamically as functions are called & return.
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, $M < N$)
Multiprogramming with Virtual Memory

- In this example, each App (program) starts at virtual address 0

- At any given time, only portions of each app's virtual memory need to reside in physical MM, due to data locality

- Without virtual memory, could have only one App in MM at a time
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)

Virtual address
(in decimal)

Page size = 4KB

Virtual page 0
(VPN = 0)

Virtual page 1
(VPN = 1)

8 bits

0
4095
4096
8191
8192

4,294,967,295
Virtual Memory and Physical MM

- During a program execution, only a subset of its virtual pages need to be in physical main memory (MM) at a time
  - When requested, a whole page is loaded from disk into a physical MM location, if it is not already present in MM
  - 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)
Virtual Memory Intuition
Address Translation Using a Page Table

A special CPU register for locating the page table

VPN analogous to the index for a cache

Page table stored in physical MM

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

Page table entries (PTEs)
Page Table Operation (1)

The page table register (PTR), a special register in CPU for locating the page table

PTR holds the physical address of the very first page table entry (PTE)
Virtual page number (VPN) is used to index the page table;

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)
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></th>
<th>Valid</th>
<th>Physical Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTE0</td>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td>PTE1</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>PTE2</td>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Page Table

Virtual memory

Lecture 24: 21
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>PTE0</th>
<th>Valid</th>
<th>Physical Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTE1</th>
<th>Valid</th>
<th>Physical Page#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTE2</td>
<td>Valid</td>
<td>Physical Page#</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Disk</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Virtual memory

8 bits

Page Table
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**
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)
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