Virtual Memory
Announcements

• **HW7 due tomorrow**

• **Final Exam: Sunday May 23, 7:30pm, 110 mins**
  - Open book, open notes
  - Covers the entire course

• **Final grade determination**
  - Median (●)
  - Std Deviation (σ)
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]

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

- How to enable multiple programs 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

- **Virtual addresses** refer to the addresses used by the programs
  - When a program is compiled, the instructions and data addresses are *virtual*
    - Do not correspond to where they will be placed in MM
  - 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

virtual addresses

\[
\begin{array}{c}
0 \\
\vdots \\
4095 \\
4096 \\
\vdots \\
8191 \\
8192 \\
\vdots \\
4,294,967,295 \\
\end{array}
\]

\[
\begin{array}{c}
\{ \text{page 0 (VPN = 0)} \} \\
\{ \text{page 1 (VPN=1)} \} \\
\end{array}
\]

page size = 4 KB

assumes 32-bit virtual address
(4 GB of virtual memory space)
Virtual Memory and Physical MM

- When requested, a \textit{page} is brought into a physical MM location.
- The correspondence (mapping) between virtual to physical addresses is saved.
  - When the same virtual address is encountered, it is \textit{translated} using this saved mapping information.
Address Translation

Virtual address

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

Virtual page number

Translation

MM address

Physical page number

Page offset

Physical address

Assuming 1GB physical memory here
(30-bit physical address)

Lecture 23: 12
Address Translation Using a Page Table

- **VPN (Virtual Page Number)**: analogous to the index for a cache.
- **Page Table Register**: special CPU register.
- **Page Table**: stored in MM.
- **Page Table Entries (PTEs)**: 
  - Valid bit: if 0, page is not present in memory.
  - $2^{12} = 4$ KB: indicates that a page contains $2^{12}$ bytes.
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.
Page Table Operation (2)

PTR+VPN form the physical address of the page table entry (PTE) in MM

Virtual address

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

Virtual page number

Page offset

Page table register

Valid

Physical page number

Page table
Page Table Operation (3)

If Valid = 1, then the PPN and Page Offset are concatenated to form the physical address.
Page Table Operation (4)

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)

---

If 0 then page is not present in memory
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, 8193
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), 8193 (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 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
Before Next Class

• H&H 6.7.2, 7.7

Next Time

Exceptions
Inputs/Outputs