Lecture Topics

- Today: Virtual Memory
  (Stallings, chapter 8.1-8.4)
- Next: Exam #2

Announcements

- Self-Study Exercise #8
- Project #8 (due 11/15)
Exam #2

- Thursday, 11/8 during lecture
- 80 minutes, 18% of course grade
- Topics:
  - the memory hierarchy
  - cache memory
  - main memory
  - virtual memory
- Study suggestions on course website

Recap: Virtual Memory

- Provides illusion of very large main memory
  - sum of memory for all processes can be larger than physical memory
  - address space of one process can be larger than physical memory
- Allows main memory to be efficiently utilized
- Simplifies memory management (relocation, protection and sharing)
- Exploits locality of reference to keep average memory access time low
A virtual memory system has the following characteristics:

Virtual address: 32 bits
Physical address: 40 bits
Size of one page: 8 kilobytes
TLB organization: fully associative, 256 entries

a) Number of bits in 1 virtual page offset?
   
   13 bits: 8 kilobytes = $2^{13}$ bytes

b) Number of bits in 1 virtual page number?
   
   19 bits: 19 + 13 = 32 (virtual address size)

A virtual memory system has the following characteristics:

Virtual address: 32 bits
Physical address: 40 bits
Size of one page: 8 kilobytes
TLB organization: fully associative, 256 entries

c) Number of bits in 1 physical page offset?
   
   13 bits: virtual page and physical page are same size

 d) Number of bits in 1 physical page number?
   
   27 bits: 27 + 13 = 40 bits (physical address size)
A virtual memory system has the following characteristics:

Virtual address: 32 bits
Physical address: 40 bits
Size of one page: 8 kilobytes
TLB organization: fully associative, 256 entries

e) Assume that a TLB entry includes a valid bit, a referenced bit, and a modified bit. Number of bits in a TLB entry?

3 bits (V, R, M)  
+ 19 bits (virtual page number)  
+ 27 bits (physical frame number)  
= 49 bits

Linux Virtual Memory (32 bit addresses)

Memory layout:
- 3 GB in user space
- 1 GB in kernel space

Loaded from file:
- machine language
- static data

Created during execution:
- stack
- heap
Linux Address Translation

Linux uses a two-level page table for 32-bit addresses (additional levels for 64-bit addresses)

- Page size: 4096 bytes
- Upper 10 bits used to identify page directory
- Middle 10 bits used to identify page table
- Lower 12 bits are the page offset
- Page table entries are 4 bytes, so 1024 PTEs per page (same with PDEs)
Linux Address Translation

Two-level page table efficient:

- virtual memory space is sparsely populated (large sections unused)
- many PDEs will be unused
- top-level table stored in one page
- each second-level table stored in one page
- page faults take two memory accesses, but relatively rare

Linux Address Translation

4-level page table for 64-bit addresses:
Recap: Memory Hierarchy

Take advantage of the principle of locality of reference to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology.

Memory Hierarchy: terminology

- Pairs of levels in the memory hierarchy:
  - Cache and RAM
  - RAM and Disk
- Block: unit of data which is transferred between levels (also called a line)
Memory Hierarchy: terminology

- Hit rate: fraction of accesses found in upper level
- Miss rate: fraction of accesses not found in upper level (1 – hit rate)
- Hit time: time to access upper level, determine if data is present or not
- Miss penalty: time to copy block from lower level to upper level, satisfy request

Memory Hierarchy: Common Framework

All levels fit into the same framework:

- Question #1: how is a block located?
- Question #2: where can a block be placed when there is a miss?
- Question #3: which block should be replaced when there is a miss?
- Question #4: how are writes handled?
Question #1 (block identification)

How is a specific block identified?

- Indexing (direct mapped cache)
- Limited search (set associative cache)
- Full search (fully associative cache)
- Table lookup (page table)

Question #2 (block placement)

When a block is copied from the lower level to the upper level, where can it be placed?

- One place (direct mapped)
- A few places (set associative)
- Any place (fully associative)
Question #3 (block replacement)

When a block is copied from the lower level, which block should it replace in the upper level?

- Only meaningful if there is a choice (set associative or fully associative)
- Optimal: block which is not needed for the longest time (not possible to implement)
- LRU: approximate optimal by looking at the blocks which were used in the past

Question #3

When a block is copied from the lower level, which block should it replace in the upper level?

- NUR: approximate LRU by looking at the blocks which have not been used recently (reset "used" bit periodically)
- Random: miss rate is only about 10% higher than LRU in studies of TLBs
Question #3

When a block is copied from the lower level, which block should it replace in the upper level?

- Caches (including TLBs) use NUR or FIFO or Random – the victim must be selected quickly by the hardware.
- Virtual memory uses NUR – miss penalty is huge, so even small improvements in the miss rate are important.

Question #4 (write policy)

How are writes handled? That is, what happens when a block in the upper level has been modified?

- Write-through: whenever a write occurs, both the upper and lower levels are updated.
- Write-back: whenever a write occurs, only the upper level is updated immediately; the lower level is updated when the block is evicted from the upper level.
Write-through

When a write occurs, the block in the upper level and in the lower level are both updated

- Misses are simpler and cheaper because they never require a block to be copied back to the lower level (already done earlier)
- Easier to implement than write-back
- Only practical for caches

Write-back

When a write occurs, the block in the upper level is updated; when the block is evicted from the upper level, it is copied to the lower level

- Misses are expensive because they may require a block to be copied back to the lower level before that block can be replaced (use buffer to reduce cost)
- Multiple writes to a single block only require one copy to the lower level
Causes of Misses

Why do misses occur in the memory hierarchy?

- Compulsory misses
- Capacity misses
- Conflict misses

Compulsory Misses

- The first access to a block will always cause a miss (that block cannot be in the upper level yet)
- Also known as cold-start misses
- Increasing the block size will reduce compulsory misses (fewer blocks, so fewer accesses to a block for the first time)
Capacity Misses

- Upper level not large enough to hold all of the blocks that the process is currently using
- Occur when a block is replaced, then retrieved again later
- Increasing the size of the upper level (so that it can hold more blocks) will reduce capacity misses

Conflict Misses

- Does not apply to fully associative
- Misses which occur when multiple blocks compete for the same location (direct mapped or set associative)
- Also known as collision misses
- Increasing the associativity of the upper level will reduce conflict misses
Virtual Memory and Cache

- All modern architectures support at least one level of cache (most support 2 or 3)
- The virtual memory system must incorporate the cache(s)
- Most caches use physical addresses (virtual addresses are mapped to physical addresses before reaching the cache)

Virtual Memory and Cache

- VA (virtual address) mapped to PA (physical address)
- PA sent to cache (and possibly RAM)
Example: Intrinsity FastMATH

- Virtual addresses: 32 bits
- Physical addresses: 32 bits
- TLB: 16 entries (fully associative)
- Instruction cache: 256 entries (direct mapped)
- Data cache: 256 entries (direct mapped)
- Cache line: 64 bytes (16 four-byte words)
Example: read from data cache

- Assume hit in TLB (if miss, physical address mapped by accessing page table)
- Cache index (8 bits) used to index into "list" of tags in data cache
- Assume hit in data cache (if miss, 64 bytes copied from RAM into data cache, 4 bytes forwarded to CPU)
- Tags and data split for efficient access
Example: access to data cache

- Virtual address mapped to physical address
- Read (load): find data (4 bytes) in data cache (hit) or RAM (miss)
- Write (store): check write access bit (allowed to write), then find data (4 bytes) in data cache (hit) or RAM (miss)
- Uses "write through": every write to data cache causes cache line to be copied to RAM
Overall operation of memory hierarchy

- Levels: cache, RAM and disk
- MMU (Memory Management Unit): TLB and page table
- Virtual address mapped to physical address by MMU
- Physical address used to access cache
<table>
<thead>
<tr>
<th>TLB</th>
<th>P Table</th>
<th>Cache</th>
<th>Possible? Under what circumstances?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Hit</td>
<td>Yes, best case. Note: page table not checked if hit in TLB.</td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
<td>Yes, but data not in cache (must be copied from RAM). Note: page table not checked.</td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Hit/Miss</td>
<td>No, PTE cannot be in TLB if page is not in RAM.</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
<td>Yes, PTE found in page table, data found in cache (PTE evicted from TLB earlier).</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Yes, PTE found in page table, data not found in cache (must be copied from RAM).</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>No, data cannot be in cache if page is not in RAM.</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>Yes, page fault. After page fault processing, retry access (Miss Hit Miss).</td>
</tr>
</tbody>
</table>

**Memory Hierarchy: summary**

- **CPU speed demands fast memory access**
  - Solution: hierarchy of cache and RAM
  - Provides the illusion of fast access (access is fast most of the time, but slow occasionally)

- **Program size demands large address space**
  - Solution: virtual memory
  - Provides the illusion of large size
Example #1

A microprocessor has 32-bit machine language instructions and 32-bit physical addresses. The instruction cache is a direct-mapped cache which contains 8192 slots. The control bit for each slot is a valid bit. A cache block is 32 bytes.

ex: 01111111111011011100101110101000

offset: 5 bits
index: 13 bits
tag: 14 bits

Example #2

A microprocessor has 32-bit physical addresses. The data cache is a direct-mapped, write-back cache with 16 slots. The control bits for each slot are a valid bit (V) and a modified bit (M). A cache block contains 256 bytes.

ex: 0111111111011011011101010101000

offset: 8 bits
index: 4 bits
tag: 20 bits
The current cache entries are shown below (in hex). For clarity, the 256-byte data blocks are not shown.

<table>
<thead>
<tr>
<th>Index</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
<th>Index</th>
<th>V</th>
<th>M</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>-</td>
<td>-</td>
<td>-----</td>
<td>-----</td>
<td>-</td>
<td>-</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>FF641</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0004A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0003A</td>
<td>A</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>FF593</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>FFF7C</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>FFF7C</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>00EA1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>D</td>
<td>1</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>E</td>
<td>1</td>
<td>1</td>
<td>0003A</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>00014</td>
<td>F</td>
<td>1</td>
<td>1</td>
<td>0003A</td>
</tr>
</tbody>
</table>

Example #3

A virtual memory system has the following: 15-bit virtual addr, 24-bit physical addr, 12-bit page offsets.

Assume the process has a fixed allocation of 5 frames (70, 71, 72, 73, 74)
Example #4

A byte-oriented virtual memory system has the following characteristics:

- Virtual address: 32 bits
- Physical address: 36 bits
- Size of one page: 4 kilobytes

The system uses a one-level page table. Each page table entry has 3 control bits and 3 access bits.

The TLB is fully associative and contains 128 slots.

Example #4 (continued)

- Virtual address: 20 + 12 bits
- Physical address: 24 + 12 bits
- Page table entry: 3 + 3 + 24 bits
  - control bits + access bits + frame number
- TLB entry: 3 + 3 + 20 + 24 bits
  - control bits + access bits + page number (tag) + frame number