Caching and Demand-Paged Virtual Memory

Chapter 9 OSPP
Definitions

• Cache
  – Copy of data that is faster to access than the original
  – Hit: if cache has copy
  – Miss: if cache does not have copy

• Cache block
  – Unit of cache storage (multiple memory locations)

• Temporal locality
  – Programs tend to reference the same memory locations multiple times
  – Example: instructions in a loop

• Spatial locality
  – Programs tend to reference nearby locations
  – Example: data in a loop
Cache Concept (Read)

- Fetch Address
- Address In Cache?
  - Yes: Store Value in Cache
  - No: Fetch Address
Cache Concept (Write)

Write through: changes sent immediately to next level of storage

Write back: changes stored in cache until cache block is replaced
Memory Hierarchy

<table>
<thead>
<tr>
<th>Cache</th>
<th>Hit Cost</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st level cache/first level TLB</td>
<td>1 ns</td>
<td>64 KB</td>
</tr>
<tr>
<td>2nd level cache/second level TLB</td>
<td>4 ns</td>
<td>256 KB</td>
</tr>
<tr>
<td>3rd level cache</td>
<td>12 ns</td>
<td>2 MB</td>
</tr>
<tr>
<td>Memory (DRAM)</td>
<td>100 ns</td>
<td>10 GB</td>
</tr>
<tr>
<td>Data center memory (DRAM)</td>
<td>100 μs</td>
<td>100 TB</td>
</tr>
<tr>
<td>Local non-volatile memory</td>
<td>100 μs</td>
<td>100 GB</td>
</tr>
<tr>
<td>Local disk</td>
<td>10 ms</td>
<td>1 TB</td>
</tr>
<tr>
<td>Data center disk</td>
<td>10 ms</td>
<td>100 PB</td>
</tr>
<tr>
<td>Remote data center disk</td>
<td>200 ms</td>
<td>1 XB</td>
</tr>
</tbody>
</table>

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core
Main Points

- Can we provide the illusion of near infinite memory in limited physical memory?
  - Demand-paged virtual memory
  - Memory-mapped files
- How do we choose which page to replace?
  - FIFO, MIN, LRU, LFU, Clock
- What types of workloads does caching work for, and how well?
  - Spatial/temporal locality vs. Zipf workloads
Hardware address translation is a power tool

• Kernel trap on read/write to selected addresses
  – Copy on write
  – Fill on reference
  – Zero on use
  – Demand paged virtual memory
  – Memory mapped files
  – Modified bit emulation
  – Use bit emulation
Demand Paging (Before)

<table>
<thead>
<tr>
<th>Page Table</th>
<th>Physical Memory Page Frames</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Page B</td>
<td>Frame for B</td>
<td>Page A</td>
</tr>
<tr>
<td>Virtual Page A</td>
<td>Frame for A</td>
<td>Page B</td>
</tr>
<tr>
<td></td>
<td>Access: Invalid</td>
<td></td>
</tr>
</tbody>
</table>
Demand Paging (After)

Page Table:  
- Virtual Page B: Frame for B, R/W
- Virtual Page A: Frame for A, Invalid

Physical Memory Page Frames:
- Page B

Disk:
- Page A
- Page B
Demand Paging

1. TLB miss
2. Page table walk
3. Page fault (page invalid in page table)
4. Trap to kernel
5. Convert virtual address to file + offset
6. Allocate page frame
   – Evict page if needed
7. Initiate disk block read into page frame
8. Disk interrupt when DMA complete
9. Mark page as valid
10. Resume process at faulting instruction
11. TLB miss
12. Page table walk to fetch translation
13. Execute instruction
Allocating a Page Frame

• Select old page to evict
• Find all page table entries that refer to old page
  – If page frame is shared
• Set each page table entry to invalid
• Remove any TLB entries
  – Copies of now invalid page table entry
• Write changes on page back to disk, if necessary
How do we know if page has been modified?

• Every page table entry has some bookkeeping
  – Has page been modified?
    • Set by hardware on store instruction
    • In both TLB and page table entry
  – Has page been recently used?
    • Set by hardware on in page table entry on every TLB miss

• Bookkeeping bits can be reset by the OS kernel
  – When changes to page are flushed to disk
  – To track whether page is recently used
Keeping Track of Page Modifications (Before)
Keeping Track of Page Modifications (After)
Virtual or Physical Dirty/Use Bits

• Most machines keep dirty/use bits in the page table entry

• Physical page is
  – Modified if *any* page table entry that points to it is modified
  – Recently used if *any* page table entry that points to it is recently used
Emulating a Modified Bit (Hardware Loaded TLB)

• Some processor architectures do not keep a modified bit per page
  – Extra bookkeeping and complexity

• Kernel can emulate a modified bit:
  – Set all clean pages as read-only
  – On first write to page, trap into kernel
  – Kernel sets modified bit, marks page as read-write
  – Resume execution

• Kernel needs to keep track of both
  – Current page table permission (e.g., read-only)
  – True page table permission (e.g., writeable, clean)
Models for Application File I/O

• Explicit read/write system calls
  – Data copied to user process using system call
  – Application operates on data
  – Data copied back to kernel using system call

• Memory-mapped files
  – Open file as a memory segment
  – Program uses load/store instructions on segment memory, implicitly operating on the file
  – Page fault if portion of file is not yet in memory
  – Kernel brings missing blocks into memory, restarts process
Advantages to Memory-mapped Files

• Programming simplicity, esp for large files
  – Operate directly on file, instead of copy in/copy out

• Zero-copy I/O
  – Data brought from disk directly into page frame

• Pipelining
  – Process can start working before all the pages are populated

• Interprocess communication
  – Shared memory segment vs. temporary file
From Memory-Mapped Files to Demand-Paged Virtual Memory

• Every process segment backed by a file on disk
  – Code segment -> code portion of executable
  – Data, heap, stack segments -> temp files
  – Shared libraries -> code file and temp data file
  – Memory-mapped files -> memory-mapped files
  – When process ends, delete temp files

• Unified memory management across file buffer and process memory
Cache Replacement Policy

• On a cache miss, how do we choose which entry to replace?
  – Assuming the new entry is more likely to be used in the near future
  – In direct mapped caches, not an issue!

• Policy goal: reduce cache misses
  – Improve expected case performance
  – Also: reduce likelihood of very poor performance
A Simple Policy

• Random?
  – Replace a random entry

• FIFO?
  – Replace the entry that has been in the cache the longest time
  – What could go wrong?
FIFO in Action

Worst case for FIFO is if program strides through memory that is larger than the cache

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td>D</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MIN, LRU, LFU

• **MIN**
  – Replace the cache entry that will not be used for the longest time into the future
  – Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss

• **Least Recently Used (LRU)**
  – Replace the cache entry that has not been used for the longest time in the past
  – Approximation of MIN

• **Least Frequently Used (LFU)**
  – Replace the cache entry used the least often (in the recent past)
LRU/MIN for Sequential Scan

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td>D</td>
<td></td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td>D</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td>A</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>LRU</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A + + +</td>
<td>A + + +</td>
</tr>
<tr>
<td>2</td>
<td>B + + + C</td>
<td>B + + + C</td>
</tr>
<tr>
<td>3</td>
<td>C + D</td>
<td>C + D</td>
</tr>
<tr>
<td>4</td>
<td>D E + +</td>
<td>D E + +</td>
</tr>
<tr>
<td>Reference</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>-----------</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIFO**

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>

**MIN**

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>C</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>D</th>
<th>E</th>
<th>A</th>
<th>E</th>
<th>B</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Belady’s Anomaly

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td>D</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO (3 slots)

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td>D</td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>C</td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

FIFO (4 slots)

<table>
<thead>
<tr>
<th>Reference</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivate

- Hard to implement LRU
- Why?
Clock Algorithm: Estimating LRU

- Periodically, sweep through all pages
- If page is unused, reclaim
- If page is used, mark as unused
Recap

• MIN is optimal
  – replace the page or cache entry that will be used farthest into the future

• LRU is an approximation of MIN
  – For programs that exhibit spatial and temporal locality

• Clock/Nth Chance is an approximation of LRU
  – Bin pages into sets of “not recently used”
Working Set Model

- Working Set: set of memory locations that need to be cached for reasonable cache hit rate
- Thrashing: when system has too small a cache
Cache Working Set
Phase Change Behavior
Question

• What happens to system performance as we increase the number of processes?
  – If the sum of the working sets > physical memory?
Memory Hogs

• How many pages to give each process?
• Ideally their working set
• But a hog or rogue can steal pages
• Solution: self-page
• Problem?
Zipf Distribution

• Caching behavior of many systems are not well characterized by the working set model

• An alternative is the Zipf distribution
  – Popularity $\sim 1/k^c$, for kth most popular item, $1 < c < 2$
Zipf Distribution

\[
\frac{1}{k^\alpha}
\]
Zipf Examples

- Web pages
- Movies
- Library books
- Words in text
- Salaries
- City population
- ...  

Common thread: popularity is self-reinforcing
Zipf and Caching

![Graph showing the relationship between cache hit rate and cache size on a log scale.](image-url)
Cache Lookup: Fully Associative
Cache Lookup: Direct Mapped
Cache Lookup: Set Associative
Page Coloring

• What happens when cache size >> page size?
  – Direct mapped or set associative
  – Multiple pages map to the same cache line

• OS page assignment matters!
  – Example: 8MB cache, 4KB pages
  – 1 of every 2K pages lands in same place in cache

• What should the OS do?
Page Coloring

Diagram showing the relationship between Processors, Virtual Address, Address Mod K, Cache, and Memory. The diagram illustrates how addresses are mapped from the virtual address space to the cache and memory, with particular emphasis on the concept of page coloring.