Caching and Virtual Memory
Last Time

• Cache concept
  – Hardware vs. software caches

• When caches work and when they don’t
  – Spatial/temporal locality vs. Zipf workloads
Main Points

• Cache Replacement Policies
  – FIFO, MIN, LRU, LFU, Clock
• Memory-mapped files
• Demand-paged virtual memory
• Other applications of virtual addressing
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>E</td>
<td></td>
<td>D</td>
<td>C</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>A</td>
<td></td>
<td>E</td>
<td>D</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>B</td>
<td></td>
<td>A</td>
<td>E</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>C</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>
</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></td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td>C</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>A</td>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>E</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>C</td>
<td></td>
<td></td>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LRU**

**MIN**

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

<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>D</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>E</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>A</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>B</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>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<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>D</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>
<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>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td>E</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>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Belady’s Anomaly

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

**FIFO (3 slots)**

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

**FIFO (4 slots)**
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 file
  - 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
Web Server

1. Network socket read
2. Copy arriving packet (DMA)
3. Kernel copy
4. Parse request
5. File read
6. Disk request
7. Disk data (DMA)
8. Kernel copy
9. Format reply
10. Network socket write
11. Kernel copy from user buffer into network buffer
12. Format outgoing packet and DMA

Server

Kernel

Hardware

Network Interface

Disk Interface
Zero Copy I/O
Block Aligned Read/Write System Calls

Before zero copy

User buffer

Kernel buffer

After zero copy

Free page

User and kernel buffer
Demand Paging

Page Table

- Frame
- Access
  - invalid
  - rd/wr

Physical memory page frames

- old data

Disk

- new data
Demand Paging

1. TLB miss
2. Page table walk
3. Page fault (page invalid in page table)
4. Trap to kernel
5. Convert 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 to page 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 to page
    - In both TLB and page table entry
  - Has page been used?
    - Set by hardware on load or store instruction to page
    - In page table entry on a TLB miss

- Can be reset by the OS kernel
  - When changes to page are flushed to disk
  - To track whether page is recently used
Physical memory

Page frames

Page Table

Frame Access

dirty = 0

Physical memory page frames

Disk

data

dirty = 0

TLB
Physical memory page frames

Page Table

Frame | Access
------|-------
dirty = 1

dirty = 1

Physical memory page frames

Disk

data

new data
Emulating a Modified Bit

• Some processor architectures do not keep a modified bit in the page table entry
  – Extra bookkeeping and complexity

• OS can emulate a modified bit:
  – Set all clean pages as read-only
  – On first write, take page fault to kernel
  – Kernel sets modified bit, marks page as read-write
Emulating a Use Bit

• Some processor architectures do not keep a use bit in the page table entry
  – Extra bookkeeping and complexity

• OS can emulate a use bit:
  – Set all unused pages as invalid
  – On first read/write, take page fault to kernel
  – Kernel sets use bit, marks page as read or read/write
Clock Algorithm: Estimating LRU
Clock Algorithm: Estimating LRU

- Periodically, sweep through all pages
- If page is unused, reclaim
- If page is used, mark as unused
Nth Chance: Not Recently Used

- Periodically, sweep through all page frames
- If page hasn’t been used in any of the past N sweeps, reclaim
- If page is used, mark as unused and set as active in current sweep
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

• Provides the illusion of an infinite amount of memory to programs
  – Unified LRU across file buffer and process memory
Question

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

Throughput

Number of Active Processes
Virtual Machines and Virtual Memory

Guest Virtual Addresses

Guest Page Tables

Guest Physical Addresses

Host Page Tables

Host Physical Addresses
Shadow Page Tables

Guest Virtual Addresses

Guest Page Tables

Guest Physical Addresses

Host Page Tables

Host Physical Addresses

Shadow Page Tables
Memory Compression

Guest Process on VM #1
Guest Page Tables
Guest Physical Addresses, VM #1

Guest Process on VM #2
Guest Page Tables
Guest Physical Addresses, VM #2

Host Page Tables
Host Physical Addresses
page A
page B
delta relative to page A
Definitions

• Checkpoint
• Restart
Transparent Checkpoint

process

checkpoint (process copy)

execute instructions

failure

restart (process copy)

process

X

time
Question

• How long do we need to wait between starting the checkpoint and resuming the execution of the program?
Incremental Checkpoint

Memory Checkpoint
A
B
C
D
E

Incremental Checkpoint
P
Q

Incremental Checkpoint
R
S

Memory Checkpoint
A
P
R
Q
S
Question

• What if we restart the process on a different machine?
  – Process migration!

• What if we checkpoint only key data structures?
  – Recoverable virtual memory