How to evict an old page

• Select an old page to evict
  – How to select? Replacement algorithm
• Find all pages that refer to the old page frame, if the page frame is shared
• Set each page table entry to invalid
• Remove TLB entries
• Write changes on page back to disk, if it has been modified

Some slides are courtesy of Dr. Thomas Anderson
How do we know if a page has been modified?

• Every page table entry has some status bits
  – Dirty bit: has page been modified?
    • Set by hardware on store instruction
  – Accessed bit: has page been recently used?
    • Set by hardware on reference

• Status bits can be reset by the OS kernel
  – When changes to page are flushed to disk
  – When enforcing the clock algorithm (to be discussed)
Cache Replacement Policy

• On a cache miss, how do we choose which entry to replace?

• Policy goal: reduce cache misses
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?
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)
<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>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>
<td>+</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</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>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>+</td>
<td>C</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>+</th>
<th>B</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>E</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>A</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></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></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></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>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>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E</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>D</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>
Clock Algorithm: Estimating LRU

- Periodically, sweep through all pages
- If page is unused, it is reclaimed
- If page is used, mark as unused
- The name of the algorithm is because, logically, all the pages form a circle during sweeps
Nth Chance: Not Recently Used

• Keep an integer for each page
  – notInUseSince: number of sweeps since last use
• Periodically sweep through all page frames
  if (page is used) {
    notInUseSince = 0;
  } else if (notInUseSince < N) {
    notInUseSince++;
  } else {
    reclaim page;
  }
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”
Thrashing

- Thrashing: when system has too small a cache, most of the time is spent on evicting cache entries and copying data to cache
- Resident Set: the pages of a process that are in memory (rather than in disk)
- Thrashing in memory happens when
  - Working Set Size > Resident Set Size
Question

• What happens to system performance as we increase the number of active processes?
  – If the sum of the working sets > physical memory?
  – In this case thrashing will occur, and the system performance will degrade dramatically
Review

• Memory subsystem
  – How to allocate physical memory?
  – How to do address translation?
  – How to speed up access?
  – How to be lazy?
    • Copy-on-write
    • Demand paging
    • Zero-copy I/O (Memory-mapped file I/O)