Page Replacement (Review)

- When the OS needs a frame to allocate to a process, and all frames are busy, it must evict (copy to backing store) a page from its frame to make room in memory
  - Reduce overhead by having CPU set a *modified / dirty* bit to indicate that a page has been modified
    - Only copy data back to disk for dirty pages
    - For non-dirty pages, just update the page table to refer to copy on disk

- Which page to we choose to replace?
  Some page replacement policies:
  - Random
    - Pick any page to evict
  - FIFO
    - Evict the page that has been in memory the longest (use a queue to keep track)
    - Idea is to give all pages “fair” (equal) use of memory

Page Replacement Policy

- When OS needs a frame to use, and all are busy, which page does it evict?
  - Random
    - Pick any page to evict
  - FIFO
    - Evict the page that has been in memory the longest (use a queue to keep track)
  - Optimal (Minimal)
    - Evict the page that will be referenced the farthest into the future
      - Requires knowledge of future
    - Cannot really be implemented
      - Useful for evaluating other policies
  - Least-Recently-Used (LRU)
    - Use the past to predict the future
    - Evict the page that has been unreferenced for the longest period of time

Page Reference Example

- Assumptions: 4 pages, 3 frames
- Page references: ABCABDADBCB

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

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

<table>
<thead>
<tr>
<th>LRU</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>A</th>
<th>D</th>
<th>B</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame 1</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>frame 2</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>frame 3</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>
Implementing LRU

A perfect implementation would be something like this:

- Associate a clock register with every page in physical memory
- Update the clock value at every access
- During replacement, scan through all the pages and find the one with the lowest value in its clock register
- What’s wrong with all this?

Simple approximations:

- FIFO
- Not-recently-used (NRU)
  - Use an R (reference) bit, and set it whenever a page is referenced
  - Clear the R bit periodically, such as every clock interrupt
  - Choose any page with a clear R bit to evict

Implementing LRU (cont.)

Clock / Second Chance Algorithm

- Use an R (reference) bit as before
- On a page fault, circle around the “clock” of all pages in the user memory pool
  - Start after the page examined last time
  - If the R bit for the page is set, clear it
  - If the R bit for the page is clear, replace that page and set the bit
- Questions:
  - Can it loop forever?
  - What does it mean if the “hand” is moving slowly? …if the hand is moving quickly?

Least Frequently Used (LFU) / N-th Chance Algorithm

- Don’t evict a page until hand has swept by N times
- Use an R bit and a counter
- How is N chosen? Large or small?

Frame Allocation

How many frames does each process get (M frames, N processes)?

- At least 2 frames (one for instruction, one for memory operand), maybe more…
- Maximum is number in physical memory

Allocation algorithms:

- Equal allocation
  - Each gets M / N frames
- Proportional allocation
  - Number depends on size and priority

Which pool of frames is used for replacement?

- Local replacement
  - Process can only reuse its own frames
- Global replacement
  - Process can reuse any frame (even if used by another process)

Thrashing

Consider what happens when memory gets overcommitted:

- After each process runs, before it gets a chance to run again, all of its pages may get paged out
- The next time that process runs, the OS will spend a lot of time page faulting, and bringing the pages back in
  - All the time it’s spending on paging is time that it’s not getting useful work done
  - With demand paging, we wanted a very large virtual memory that would be as fast as physical memory, but instead we’re getting one that’s as slow as the disk!

This wasted activity due to frequent paging is called thrashing

- Analogy — student taking too many courses, with too much work due
Working Sets

- Thrashing occurs when the sum of all processes’ requirement is greater than physical memory
  - Solution — use local page frame replacement, don’t let processes compete
    - Doesn’t help, as an individual process can still thrash
  - Solution — only give a process the number of frames that it “needs”
    - Change number of frames allocated to each process over time
    - If total need is too high, pick a process and suspend it

- Working set (Denning, 1968) — the collection of pages that a process is working with, and which must be resident in main memory, to avoid thrashing
  - Always keep working set in memory
  - Other pages can be discarded as necessary