Page Replacement Algorithms

How do we choose a frame to swap out?

- **Optimal**
  - Evict the page that won’t be needed until furthest in the future
- **Not Recently Used (NRU)**
  - Evict the page that is the oldest, preferring pages that are not dirty:
    
    | Preference | Referenced | Dirty |
    |------------|------------|-------|
    | First Choice | 0 | 0 |
    | 1 | 1 |
    | Last Choice | 1 | 1 |

- **FIFO**
  - First in, First out

Second Chance Page Replacement

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
</tbody>
</table>

Key:
- referenced
- unreferenced

Clock Algorithm

Least Recently Used (LRU)

Look to the past to predict the future

Aging Scheme
Working Set

- Working set is the set of pages used by the \( k \) most recent memory references
- \( w(k,t) \) is the size of the working set at time \( t \)

Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT (Optimal)</td>
<td>Not implementable, but useful as a benchmark</td>
</tr>
<tr>
<td>NRU (Not Recently Used)</td>
<td>Crude</td>
</tr>
<tr>
<td>FIFO (First-In, First-Out)</td>
<td>Might throw out useful pages</td>
</tr>
<tr>
<td>Second chance</td>
<td>Big improvement over FIFO</td>
</tr>
<tr>
<td>Clock</td>
<td>Better implementation of second chance</td>
</tr>
<tr>
<td>LRU (Least Recently Used)</td>
<td>Excellent, but hard to implement exactly</td>
</tr>
<tr>
<td>NFU (Not Frequently Used)</td>
<td>Poor approximation to LRU</td>
</tr>
<tr>
<td>Aging</td>
<td>Good approximation to LRU, efficient to implement</td>
</tr>
<tr>
<td>Working Set</td>
<td>Somewhat expensive to implement</td>
</tr>
<tr>
<td>WSClock</td>
<td>Implementable version of Working Set</td>
</tr>
</tbody>
</table>

Modeling Page Replacement

- FIFO replacement on reference string

Belady’s Anomaly

- Try to reduce the number of page faults by supplying more memory
  - Use previous reference string and FIFO algorithm
  - Add another page to physical memory (total 4 pages)

Local vs. Global Allocation
Page Fault Frequency

Page Size
- For larger pages
  - Smaller page tables
  - Less frames in memory (smaller degree of multiprogramming?)
  - Internal fragmentation
- For smaller pages
  - Bigger page table
  - More levels of page tables
  - Less wasted space

I- and D-Spaces

Page Sharing
*Map multiple pages to a single frame*

When to Write to Disk
*Now or later?*

Implementation
- Process creation
- During process execution
- Page fault time
- Process termination time
Handling a Page Fault

1. Determine faulting virtual address
2. If the page is invalid, grow stack or heap, alternatively SEGFAULT on error
3. If physical memory is full, choose a frame to evict
4. Write frame to disk if dirty
5. Load requested page into now empty frame

Segmentation