Least Recently Used (LRU) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- How to implement??
LRU Algorithm (Cont.)

- **Counter implementation** (Needs hardware assistance)
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to determine which are to change

- **Stack implementation** - keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

Use Of A Stack to Record The Most Recent Page References

<table>
<thead>
<tr>
<th>reference string</th>
<th>4</th>
<th>7</th>
<th>0</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>7</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</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></td>
<td>1</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></td>
<td>0</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></td>
<td>7</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></td>
<td>4</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>

stack before a

stack after b

a

b
LRU Approximation Algorithms

- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace the one which is 0 (if one exists). We do not know the order, however.

- **Additional Reference bits**
  - 1 byte for each page: eg. 00110011
  - Shift right at each time interval

- **Second chance**
  - Need reference bit
  - Clock replacement
  - If page to be replaced (in clock order) has reference bit = 1 then:
    - set reference bit 0
    - leave page in memory
    - replace next page (in clock order), subject to same rules

---

Second-Chance (clock) Page-Replacement Algorithm

![Diagram](image.png)
Counting Algorithms

- Keep a counter of the number of references that have been made to each page

- **LFU Algorithm**: replaces page with smallest count

- **MFU Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Allocation of Frames

- Each process needs *minimum* number of pages

- Two major allocation schemes
  - fixed allocation
  - priority allocation
Fixed Allocation

- **Equal allocation** - For example, if there are 100 frames and 5 processes, give each process 20 frames.
- **Proportional allocation** - Allocate according to the size of process

\[ s_i = \text{size of process } p_i \]
\[ S = \sum s_i \]
\[ m = \text{total number of frames} \]
\[ a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m \]

\[ m = 64 \]
\[ s_i = 10 \]
\[ s_2 = 127 \]
\[ a_1 = \frac{10}{137} \times 64 \approx 5 \]
\[ a_2 = \frac{127}{137} \times 64 \approx 59 \]

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- If process \( P_i \) generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number
Global vs. Local Allocation

- **Global replacement** - process selects a replacement frame from the set of all frames; one process can take a frame from another
- **Local replacement** - each process selects from only its own set of allocated frames

Thrashing

- If a process does not have “enough” frames, the page-fault rate is very high. This leads to:
  - Replacement of active pages which will be needed soon again
  - **Thrashing** = a process is busy swapping pages in and out
- Which will in turn cause:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system
Thrashing (Cont.)

Locality in a Memory-Reference Pattern
Working-Set Model

- $\Delta$ = working-set window = a fixed number of page references
  Example: 10,000 instruction
- $WSS_i$ (working set of Process $P_i$) =
  total number of pages referenced in the most recent $\Delta$ (varies in time)
  - if $\Delta$ too small will not encompass entire locality
  - if $\Delta$ too large will encompass several localities
  - if $\Delta = \infty$ ⇒ will encompass entire program
- $D = \sum WSS_i$ = total demand frames
- if $D > m$ ⇒ Thrashing
- Policy if $D > m$, then suspend one of the processes

Working-set model

<table>
<thead>
<tr>
<th>page reference table</th>
</tr>
</thead>
<tbody>
<tr>
<td>... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 4 4 4 1 3 2 3 4 4 3 4 4 4 4 ...</td>
</tr>
<tr>
<td>$\Delta$</td>
</tr>
<tr>
<td>$t_1$</td>
</tr>
</tbody>
</table>

$WS(t_1) = \{1, 2, 5, 6, 7\}$
$WS(t_2) = \{3, 4\}$
Summary

- Virtual Memory
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms
    - (FIFO, LRU, Optimal etc)
  - Performance of Demand Paging

- Reading Assignment: Chapter 8 from Silberschatz.

Acknowledgements


- “Modern Operating Systems” book and supplementary material by A. Tanenbaum

- R. Doursat and M. Yuksel from UNR