Replacement Algorithms

- **Goal**: Select a page to be replaced when a new page must be swapped into memory

- **Basic Algorithms**
  - **Optimal (Impractical)**
    - Idea: Replace the page for which the time to the next reference is the longest.
    - Impossible to implement, but serves as a reference point
  - **First-In, First-Out (FIFO)**
    - Idea: Replace the oldest page
  - **Least Recently Used (LRU)**
    - Idea: Select the page that has not been referenced for the longest time in the past
    - **Clock**
      - Approximates LRU using a clock structure

- **Example Page Reference Stream**:
  - 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2

---

**Replacement Example (1)**

<table>
<thead>
<tr>
<th>OPT</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>FIFO</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>LRU</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Optimal and FIFO Page Replacement**

- **Optimal (Impractical)**
  - Idea: Replace the page for which the time to the next reference is the longest.
  - Impossible to implement, but serves as a reference point

- **First-In, First-Out (FIFO)**
  - Idea: Replace the page that has been in memory the longest (i.e., the oldest)
  - One of the simplest algorithms, but performs poorly
  - Treat page frames of a process as a circular buffer
  - Pages are removed in RR order
  - Implement as a pointer that cycles through the page frames of a process
LRU Page Replacement

- **Least Recently Used (LRU)**
  - **Idea:** Replace the page that hasn’t been referenced for the longest time in the past
  - Does almost as well as optimal algorithm on some reference sequences
  - Difficult to implement in hardware
    - Time stamp each page and replace the oldest one
    - Use a stack with the most recently referenced page on top

Replacement Example (2)

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>FIFO</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>CLOCK</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Clock Page Replacement

- **Clock**
  - **Idea:** Faster approximation of LRU
  - Select an unmarked frame in RR order
    - Cursor cycles through page frames looking for an unmarked page as the replacement page
  - When to **unmark** the page underneath the cursor:
    - If the page is marked, then go to next page
    - If all pages are marked, the cursor will return to the first one it unmarked
  - When to **mark** a page:
    - When loaded into main memory
    - When already in main memory and it has been referenced

Replacement Algorithm Variations

- **Counting Algorithms**
  - Least Frequently Used (LFU): Select page with smallest reference count
  - Most Frequently Used (MFU): Select page with largest reference count

- **Use Bit (U) and Modified Bit (M)**
  - U = 1 ➔ Page has been referenced recently
  - M = 1 ➔ Page has been modified and needs to be written to disk

- **Page Buffering**
  - Replaced page is assigned to either the free page list (if M is 0) or the modified list (otherwise)
    - Page remains in memory
    - Cluster page writes of modified pages
**The Working Set Model**

- $W(t, \Delta)$ is the working set at virtual time $t$ with a window size of $\Delta$ and is:
  - Defined over the page reference string for each process
  - The set of pages that have been referenced in the time interval $[t - \Delta, t]$.
- Example:
  $$W(4,4) = \{0, 3, 8, 9\}$$
  $$W(2,2) = \{0, 9\}$$
  $$W(15,5) = \{0, 2, 9\}$$

- $W(t, \Delta)$ varies over time $t$ even with a fixed window size $\Delta$

**Example**

- Reference String (RS): $(4,3,0,2,2,3,1,2,4,2,4,0,3)$
- Working Set ($\Delta = 4$; x indicates page is in memory)

```
Page 0 x 2 2 3 1 2 4 2 4 0 3
1 x x x x x x x
2 x x x x x x x
3 x x x x x x x
4 x x x x x x x
IN 2 1 4 0 3
OUT 4 0 3
```

- PFF ($F = 3$)

```
Page 0 x 2 2 3 1 2 4 2 4 0 3
1 x x x x x x x
2 x x x x x x x
3 x x x x x x x
4 x x x x x x x
IN 2 shrink 1 add 4 shrink 0 add
OUT F = 3 0, 4 F = 2 0, add 1, 3 From Shrink & Shaw
```

**The Working Set Strategy**

- **The Strategy**
  - Monitor $W(t, \Delta)$ for each process
  - Periodically remove pages from the resident set of a process that are not in its $W(t, \Delta)$
  - Schedule a process only if its working set is in main memory
- **Problems**
  - The past doesn't always predict the future
  - An exact measurement of $W(t, \Delta)$ is impractical because it requires a time-ordered queue of pages.
  - The optimal value of $\Delta$ is unknown

**Page-Fault Frequency (PFF) Algorithm**

- **Idea:** Adjust resident set size (RSS) according to page fault rate
- **Basic Algorithm**
  - Select a threshold $F$, the minimum time between page faults
  - Mark each page that is referenced with a use-bit (U) of 1
  - When a page fault occurs, compute the time $F'$ since the last page fault and adjust the resident set size:
    - $F' < F$: Add a page to the resident set
    - $F' \geq F$: Discard all pages with a use-bit (U) of 0, and shrink the resident set size
  - Reset all use-bits after a page fault
- **Flaw:** Poor performance during expanding transition periods
Variable-Interval Sampled WS

**Idea**
- Deal with transition periods by sampling more frequently and discarding unused pages when there are more page faults

**Algorithm Parameters**
- \( I^- \): Minimum duration of the sampling interval
- \( I^+ \): Maximum duration of the sampling interval
- \( F \): Number of page faults that are allowed to occur between sampling intervals

**Algorithm**
- Suspend process and scan the U-bits when:
  1) \( T = I^+ \) and 2) \( I^- < T < I^+ \) and \#Faults = \( F \)
- RSS can only shrink at end of sampling interval; it either remains the same or increases within each interval

Practical Systems

**Machines typically don’t support LRU or WS**
- Typically, have an R-bit (referenced)

**Portable kernel code**
- May not use all features of VM hardware

**Aging - a software solution**
- Based on NFU (Not Frequently Used) algorithm
- OS scans all pages at each clock interrupt (10 msec)
  - Right shift each R-bit into its 8-bit age counter
  - Then, zero (reset) each R-bit
- Replacement page is the one with the smallest age
- NFU
  - Adds R-bit to age counter
  - Problem is that past behavior can incorrectly effect replacement

### Aging Example

<table>
<thead>
<tr>
<th>Page</th>
<th>Tick 0</th>
<th>Tick 1</th>
<th>Tick 2</th>
<th>Tick 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000000</td>
<td>0100000</td>
<td>1100000</td>
<td>11110000</td>
</tr>
<tr>
<td>1</td>
<td>0000000</td>
<td>0000000</td>
<td>0100000</td>
<td>01100000</td>
</tr>
<tr>
<td>2</td>
<td>0100000</td>
<td>0100000</td>
<td>0010000</td>
<td>00100000</td>
</tr>
<tr>
<td>3</td>
<td>0000000</td>
<td>0000000</td>
<td>1000000</td>
<td>10100000</td>
</tr>
<tr>
<td>4</td>
<td>1000000</td>
<td>1010000</td>
<td>0110000</td>
<td>10110000</td>
</tr>
<tr>
<td>5</td>
<td>0000000</td>
<td>0100000</td>
<td>1010000</td>
<td>01010000</td>
</tr>
</tbody>
</table>

Source: Tanenbaum, Modern Operating Systems

### WS Clock Page Replacement

- \( R \)-bit = 1 if page has been referenced (read/write)

- **Age counter**

- \( M \)-bit = 1 if page has been modified (write)

- Each page falls into one of four \((R,M)\) classes
  - \((0,0)\): Neither recently used nor dirty (\textit{CLEAN})
  - \((0,1)\): Not used but dirty (want to convert to \((0,0)\))
  - \((1,0)\): Recently used and clean (maybe don’t replace)
  - \((1,1)\): Recently used and dirty (might be used again soon)

- **Replacement**
  - Page is clean \((R=0,M=0)\) and age counter < threshold
    - Evict and reduce WS
  - Page is \((R=0,M=1)\) Schedule write to disk
    - Limit number of writes
    - Avoids context switch
    - Hope to find clean page later in scan
## Belady's Anomaly

FIFO page replacement can produce more page faults when given more frames.

<table>
<thead>
<tr>
<th>Ref</th>
<th>Str</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Youngest</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Oldest</td>
<td></td>
</tr>
</tbody>
</table>

## Summary

- **Optimal**: Not implementable, but useful benchmark
- **FIFO**: Might throw out important pages
- **LRU**: Excellent but difficult to implement in HW
- **WS**: Expensive to implement
- **Aging**: Efficient approximate LRU algorithm
- **WSClock**: Good, efficient algorithm

### Interesting References