Page Replacement Algorithms (1)

- Example *Page Reference Stream (String)*
  » 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2
- Assume
  » Global Policy
    • Entire set of page frames is shared by all processes
    • Number of frames allocated to a process can vary over time
  » When free frame needed
    • All resident pages are candidates for eviction
  » Alternative is a *Local Replacement* policy
    • Number of pages allocated to a process is fixed (a constant)

Goal of global replacement algorithm
» Select a good page to be replaced when a new page must be swapped into memory

Page Replacement Algorithms (2)

- Basic Algorithms
  » *Optimal* (Impractical)
    • Select the P for which the time to the next reference is the longest.
  » *First-In, First-Out (FIFO)*
    • Select the oldest P
  » *Least Recently Used (LRU)*
    • Select the P that hasn’t been referenced for the longest time in the past
  » *Clock*
    • Approximates LRU using a clock structure
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

- **Least Recently Used (LRU)**
  - **Idea**: Replace the page that hasn’t been referenced for the longest time in the past
  - 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 (1)**

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

**Clock Page Replacement**

- **Clock (Second Chance Algorithm)**
  - **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

- **Enhanced Clock (Third Chance Algorithm)**
  - Scan 1: Search for \((R,M) = (0,0)\) for replacement
    - \((1,1) \to (0,1); (1,0) \to (0,0); (0,1) \to (0,0*)\) requires disk write
  - Scan 2: Search for \((R,M) = (0,0)\) for replacement

**LRU Page Replacement**
**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:**

\[
\begin{align*}
W(4,4) &= \{0, 3, 8, 9\} \\
W(2,2) &= \{0, 9\} \\
W(15,5) &= \{0, 2, 9\}
\end{align*}
\]

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

---

**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

---

**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)

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>OUT</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**PFF** (\( F = 3 \): expand RSS if inter-fault time < 3)

<table>
<thead>
<tr>
<th>Page</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>OUT</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

---

**Design Issues (Paging)**

- **Local versus Global policy**
  - Global policies tend to work better
  - Allow WS size to vary over a process’ lifetime to avoid thrashing

- **Load Control**
  - Sum of all WS sizes > Size of memory \( \rightarrow \) Thrashing likely
  - Approach: Swap out some processes

- **Page Size**
  - Selectable by OS (to some extent) (Typical: 4 or 8 KB)
  - Issues: internal fragmentation, PT size, paging time

- **Sharing Pages**

- **Cleaning Policy**
  - Keep a supply of free frames \( \rightarrow \) Better performance
  - Paging daemon: Two-handed clock algorithm evicts pages
### 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>R-bits</th>
<th>Page 0</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R01011</td>
<td>0000000</td>
<td>0000000</td>
<td>11100000</td>
<td>11110000</td>
</tr>
<tr>
<td>R10000</td>
<td>0000000</td>
<td>1000000</td>
<td>1000000</td>
<td>01100000</td>
</tr>
<tr>
<td>R10000</td>
<td>1000000</td>
<td>01000000</td>
<td>00100000</td>
<td>00010000</td>
</tr>
<tr>
<td>R00000</td>
<td>0000000</td>
<td>0000000</td>
<td>10000000</td>
<td>10000000</td>
</tr>
</tbody>
</table>

Source: Tanenbaum, Modern Operating Systems

### WS Clock Page Replacement (1)

- R-bit = 1 if page has been referenced (read/write)
- M-bit = 1 if page has been modified (write)
- Every page has a "time of last use" field
- After a clock interrupt
  - ALL R-bits are cleared
  - Set "time of last use" to virtual clock value of every page with R-bit = 1
  - Evict page (reduce WS) if page is clean (R=0,M=0) and age < threshold
    - age = current virtual clock value - time of last use

### WS Clock Page Replacement (2)

- Each page falls into one of four (R,M) classes
  - (0,0): Neither recently used nor dirty (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
  - Ideal replacement page is (R=0,M=0)
  - 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
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

Implementation Issues

- Paging Work Done by OS
  » process creation, execution and termination, page fault
- Handling a Page Fault
- Instruction Backup
  » Need to restart instruction causing a page fault
- Locking Pages in Memory
  » What if process reading file directly into buffer gets swapped out ??
- Backing Store Design
  » Where to put pages on disk ???
- Separating Policy From Mechanism

Mach Policy and Mechanism Separation