8. Virtual Memory

8.1 Principles of Virtual Memory

8.2 Implementations of Virtual Memory

- Paging
- Segmentation
- Paging With Segmentation
- Paging of System Tables
- Translation Look-aside Buffers

8.3 Memory Allocation in Paged Systems

- Global Page Replacement Algorithms
- Local Page Replacement Algorithms
- Load Control and Thrashing
- Evaluation of Paging
Principles of Virtual Memory

- For each process, the system creates the illusion of large contiguous memory space(s)
- Relevant portions of Virtual Memory (VM) are loaded automatically and transparently
- Address Map translates Virtual Addresses to Physical Addresses
Principles of Virtual Memory

• Single-segment Virtual Memory:
  – One area of $0..n-1$ words
  – Divided into fix-sized pages

• Multiple-Segment Virtual Memory:
  – Multiple areas of up to $0..n-1$ (words)
  – Each holds a logical segment (e.g., function, data structure)
  – Each logical segment
    • may be contiguous
    • may be divided into pages
Main Issues in VM Design

1. Address mapping
   – How to translate virtual addresses to physical addresses

2. Placement
   – Where to place a portion of VM needed by process

3. Replacement
   – Which portion of VM to remove when space is needed

4. Load control
   – How much of VM to load at any one time

5. Sharing
   – How can processes share portions of their VMs
VM Implementation via Paging

• **VM** is divided into **fix-sized pages** : \( \text{page\_size}=2^{|w|} \)

• **PM** (physical memory) is divided into \( 2^{|f|} \) **page frames** : \( \text{frame\_size}=\text{page\_size}=2^{|w|} \)

• System loads pages into frames and translates addresses

• Virtual address: \( \text{va} = (p,w) \)

• Physical address: \( \text{pa} = (f,w) \)

• \(|p|, |f|, \text{ and } |w|\)
  - \(|p| \) determines number of pages in VM, \( 2^{|p|} \)
  - \(|f| \) determines number of frames in PM, \( 2^{|f|} \)
  - \(|w| \) determines page/frame size, \( 2^{|w|} \)

Figure 8-2
Paged Virtual Memory

- Virtual address: $va = (p, w)$
- Physical address: $pa = (f, w)$
- $2^{|p|}$ pages in VM; $2^{|w|} = \text{page/frame size}$; $2^{|f|}$ frames in PM
Paged VM Address Translation

• Given \((p, w)\), how to determine \(f\) from \(p\) ?
• One solution: *Frame Table* :
  
  – One entry, \(FT[i]\), for each frame
    \(FT[i].pid\) records process ID
    \(FT[i].page\) records page number \(p\)
  
  – Given \((id, p, w)\), search for a match on \((id, p)\)
    \(f\) is the \(i\) for which \((FT[i].pid, FT[i].page) = (id, p)\)

  – Pseudocode for Frame Table lookup:

```c
address_map(id, p, w)
{
    pa = UNDEFINED;
    for (f=0; f<F; f++)
        if (FT[f].pid == id && FT[f].page == p) pa = f+w;
    return pa;
}
```
Address Translation via Frame Table

address_map(id,p,w) {
    pa = UNDEFINED;
    for (f=0; f<F; f++)
        if (FT[f].pid==id &&
            FT[f].page==p)
            pa=f+w;
    return pa;
}

• **Drawbacks**
  – Costly: Search must be done in parallel in hardware
  – Sharing of pages: difficult or not possible

Figure 8-4
Page Table for Paged VM

- **Page Table (PT)** is associated with each VM (not PM)
- **Page table register** $\text{PTR}$ points at PT at run time
- Entry $p$ of PT holds frame number of page $p$:
  - $*(\text{PTR}+p)$ points to frame $f$
- **Address translation:**
  
  ```
  address_map(p, w) {
    pa = *(PTR+p) + w;
    return pa
  }
  ```
- **Drawback:**
  Extra memory access

Figure 8-5
Demand Paging

- All pages of VM can be loaded initially
  - Simple, but maximum size of VM = size of PM
- Pages a loaded as needed: on demand
  - Additional bit in PT indicates a page’s presence/absence in memory
  - Page fault occurs when page is absent

```c
address_map(p, w)
{
    if (resident(*(PTR+p))) {
        pa = *(PTR+p)+w; return pa; }
    else page_fault;
}
```
VM using Segmentation

• Multiple contiguous spaces: *segments*
  – More natural match to program/data structure
  – Easier sharing (Chapter 9)

• Virtual address \((s, w)\) mapped to physical address (but no frames)

• Where/how are segments placed in physical memory?
  – Contiguous
  – Paged
Contiguous Allocation

- Each segment is contiguous in physical memory
- *Segment Table (ST)* tracks starting locations
- *Segment Table Register STR* points to segment table
- Address translation:
  
  ```
  address_map(s, w)
  {
    if (resident(*(STR+s))) {
      pa = *(STR+s)+w;
      return pa; }
    else segment_fault; }
  }
  ```

- Drawback: External fragmentation
Paging with segmentation

- Each segment is divided into fix-size pages
- \( va = (s, p, w) \)
  - \(|s|\) determines # of segments (size of ST)
  - \(|p|\) determines # of pages per segment (size of PT)
  - \(|w|\) determines page size
- \( pa = *(*(STR+s)+p)+w \)
- Drawback:
  - 2 extra memory references

Figure 8-7
Paging of System Tables

- ST or PT may be too large to keep in PM
  - Divide ST or PT into pages
  - Keep track by additional page table

- Paging of ST
  - ST divided into pages
  - Segment directory keeps track of ST pages
  - $va = (s_1, s_2, p, w)$
  - $pa = \ast(\ast(\ast(STR+s_1)+s_2)+p)+w$

- Drawback:
  3 extra memory references

Figure 8-8
Translation Look-aside Buffers

- *Translation Lookaside Buffer (TLB)* avoids some additional memory accesses
  - Keep most recently translated page numbers in associative memory:
    For any \((s,p,*)\); keep \((s,p)\) and frame number \(f\)
  - Bypass translation if match found on \((s,p)\)

- TLB ≠ cache
  - TLB keeps only frame numbers
  - Cache keeps data values

Figure 8-10
Memory Allocation with Paging

• Placement policy: Any free frame is OK
• Replacement: Goal is to minimize data movement between physical memory and secondary storage
• Two types of replacement strategies:
  – Global replacement: Consider all resident pages, regardless of owner
  – Local replacement: Consider only pages of faulting process
• How to compare different algorithms:
  – Use Reference String (RS) : \( r_0 r_1 \ldots r_t \ldots \)
    \( r_t \) is the (number of the) page referenced at time \( t \)
  – Count number of page faults
Global page replacement

• **Optimal (MIN):** Replace page that will not be referenced for the longest time in the future

<table>
<thead>
<tr>
<th>Time t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>Frame 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>Frame 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Frame 2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>Frame 3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e</td>
<td></td>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d</td>
<td></td>
<td>a</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Problem: Need entire reference string (i.e., need to know the future)
Global Page Replacement

• *Random Replacement:* Replace a randomly chosen page
  – Simple but
  – Does not exploit *locality of reference*
    • Most instructions are *sequential*
    • Most *loops* are short
    • Many data structures are *accessed sequentially*
Global page replacement

- **First-In First-Out (FIFO):** Replace oldest page

<table>
<thead>
<tr>
<th>Time t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>Frame 0</td>
<td>&gt;a</td>
<td>&gt;a</td>
<td>&gt;a</td>
<td>&gt;a</td>
<td>&gt;a</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>&gt;e</td>
</tr>
<tr>
<td>Frame 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>&gt;b</td>
<td>&gt;b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>&gt;a</td>
</tr>
<tr>
<td>Frame 2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>&gt;c</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Frame 3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>&gt;d</td>
<td>c</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e</td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a</td>
</tr>
</tbody>
</table>

- **Problem:**
  - Favors recently loaded pages, but
  - Ignores when program returns to old pages
## Global Page Replacement

- **LRU**: Replace Least Recently Used page

<table>
<thead>
<tr>
<th>Time t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>Frame 0</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>Frame 1</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>Frame 2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>Frame 3</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>c</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e</td>
<td></td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c</td>
<td></td>
<td>d</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q.end</td>
<td>d</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>Q.head</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>d</td>
<td>e</td>
<td>a</td>
</tr>
</tbody>
</table>
Global page replacement

- **LRU implementation**
  - **Software queue**: too expensive
  - **Time-stamping**
    - Stamp each referenced page with current time
    - Replace page with oldest stamp
  - **Hardware capacitor** with each frame
    - Charge at reference
    - Charge decays exponentially
    - Replace page with smallest charge
  - **n-bit aging register** with each frame
    - Shift all registers to right periodically (or at every reference to any page)
    - Set left-most bit of referenced page to 1
    - Replace page with smallest value
  - **Simpler algorithms** that approximate LRU algorithm
Global Page Replacement

• *Second-chance algorithm*
  – Approximates LRU
  – Implement *use-bit* $u$ with each frame
  – Set $u=1$ when page referenced
  – To select a page:
    • If $u==0$, select page
    • Else, set $u=0$ and consider next frame
  – Used page gets a second chance to stay in PM

• Algorithm is called *clock algorithm*:
  – Search cycles through page frames
Global page replacement

- Second-chance algorithm

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>b</td>
<td>e</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>...</td>
<td>&gt;a/1</td>
<td>e/1</td>
<td>e/1</td>
<td>e/1</td>
<td>e/1</td>
<td>&gt;e/1</td>
<td>d/1</td>
</tr>
<tr>
<td>...</td>
<td>b/1</td>
<td>&gt;b/0</td>
<td>&gt;b/1</td>
<td>b/0</td>
<td>b/1</td>
<td>b/1</td>
<td>&gt;b/0</td>
</tr>
<tr>
<td>...</td>
<td>c/1</td>
<td>c/0</td>
<td>c/0</td>
<td>a/1</td>
<td>a/1</td>
<td>a/1</td>
<td>a/0</td>
</tr>
<tr>
<td>...</td>
<td>d/1</td>
<td>d/0</td>
<td>d/0</td>
<td>&gt;d/0</td>
<td>&gt;d/0</td>
<td>c/1</td>
<td>c/0</td>
</tr>
<tr>
<td>...</td>
<td>e</td>
<td>a</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Global Page Replacement

• *Third-chance algorithm*
  – Second chance algorithm does not distinguish between read and write access
    • Write access more expensive
  – Give modified pages a third chance:
    • *use-bit* $u$ set at every reference (read and write)
    • *write-bit* $w$ set at write reference
    • *dirty-bit* needed to keep track of whether page has been modified
    • to select a page, cycle through frames, resetting bits, until $uw==00$:
      $u\ w \rightarrow u\ w$
      
      |   |   |   |
      | 1 | 1 | 0 | 1 |
      | 1 | 0 | 0 | 0 |
      | 0 | 1 | 0 | 0 * (set dirty bit to remember modification) |
      | 0 | 0 | (select page for replacement) |
Global Page Replacement

- Third-chance algorithm
  
  Read->10->00->Select
  Write->11->01->00*->Select

<table>
<thead>
<tr>
<th>...</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>c</td>
<td>a&lt;sup&gt;W&lt;/sup&gt;</td>
<td>d</td>
<td>b&lt;sup&gt;W&lt;/sup&gt;</td>
<td>e</td>
<td>b</td>
<td>a&lt;sup&gt;W&lt;/sup&gt;</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>&gt;a/10</td>
<td>&gt;a/10</td>
<td>&gt;a/11</td>
<td>&gt;a/11</td>
<td>&gt;a/11</td>
<td>a/00*</td>
<td>a/00*</td>
<td>a/11</td>
<td>a/11</td>
<td>&gt;a/11</td>
<td>a/00*</td>
<td></td>
</tr>
<tr>
<td>... b/10</td>
<td>b/10</td>
<td>b/10</td>
<td>b/10</td>
<td>b/11</td>
<td>b/00*</td>
<td>b/10*</td>
<td>b/10*</td>
<td>b/10*</td>
<td>b/10*</td>
<td>d/10</td>
<td></td>
</tr>
<tr>
<td>... c/10</td>
<td>c/10</td>
<td>c/10</td>
<td>c/10</td>
<td>c/10</td>
<td>e/10</td>
<td>e/10</td>
<td>e/10</td>
<td>e/10</td>
<td>e/10</td>
<td>&gt;e/00</td>
<td></td>
</tr>
<tr>
<td>... d/10</td>
<td>d/10</td>
<td>d/10</td>
<td>d/10</td>
<td>d/10</td>
<td>&gt;d/00</td>
<td>&gt;d/00</td>
<td>&gt;d/00</td>
<td>&gt;d/00</td>
<td>&gt;d/00</td>
<td>c/10</td>
<td>c/00</td>
</tr>
<tr>
<td>... IN</td>
<td></td>
<td>e</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>... OUT</td>
<td></td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d</td>
<td>b</td>
</tr>
</tbody>
</table>

CompSci 143A  
Spring, 2013
Local Page Replacement

• Measurements indicate that every program needs a minimum set of pages to be resident in memory
  – If too few, *thrashing* occurs
  – If too many, page frames are wasted

• The size of the minimum set varies over time

• Goal: attempt to maintain an optimal resident set of pages for each active process
  – Number of resident pages for each process changes over time
Local Page Replacement

• **Optimal (VMIN)**
  – Define a sliding window \((t, t+\tau)\)
  – \(\tau\) is a parameter (constant)
  – At any time \(t\), maintain as resident all pages visible in window

• Guaranteed to generate smallest number of page faults

• Requires knowledge of future
Local page replacement

- **Optimal (VMIN) with \( \tau=3 \)**

<table>
<thead>
<tr>
<th>Time ( t )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>d</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>Page a</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>Page b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Page c</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Page d</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Page e</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IN</td>
<td></td>
<td>c</td>
<td>b</td>
<td>e</td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT</td>
<td></td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Unrealizable without entire reference string (knowledge of future)**
Local Page Replacement

• *Working Set Model:*
  – Uses *principle of locality*: Memory requirement for a process in the near future is closely approximated by the process’s memory requirement in the recent past
  – Use trailing window (instead of future window)
  – Working set \( W(t, \tau) \) is all pages referenced during the interval \( (t-\tau, t) \)
  – At time \( t \):
    • Remove all pages not in \( W(t, \tau) \)
    • Process may run only if entire \( W(t, \tau) \) is resident
Local Page Replacement

- **Working Set Model with $\tau=3$**

<table>
<thead>
<tr>
<th>Time t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>a</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>Page a</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Page b</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Page c</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Page d</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Page e</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

| IN     |   | c | b | e | a | d |
| OUT    |   | e | a | d | b |   |

- **Drawback**: costly to implement
- **Approximate**: (aging registers, time stamps)
Local Page Replacement

• *Page fault frequency (PFF)*
• **Goals**
  – Keep frequency of page faults acceptably low
  – Keep resident page set from growing unnecessarily large
• **Uses a parameter** $\tau$
• **Only adjust resident set when a page fault occurs**
• **Rule: When a page fault occurs**
  – If time between page faults $\leq \tau$
    • Add new page to resident set
  – If time between page faults $> \tau$
    • Add new page to resident set
    • Remove all pages not referenced since last page fault
# Local Page Replacement

- **Page Fault Frequency with $\tau=2$**

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td></td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>e</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>Page a</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Page b</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Page c</td>
<td></td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Page d</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Page e</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td></td>
<td>c</td>
<td>b</td>
<td>e</td>
<td></td>
<td>a</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OUT</strong></td>
<td></td>
<td></td>
<td>ae</td>
<td>bd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Load Control and Thrashing

• Main issues:
  – How to choose the amount/degree of multiprogramming?
  – When level decreased, which process should be deactivated?
  – When new process reactivated, which of its pages should be loaded?
  – *Load Control:* Policy setting
    number and type of concurrent processes
  – *Thrashing:* Effort moving pages
    between main and secondary memory
Load Control and Thrashing

- Choosing degree of multiprogramming
- Local replacement:
  - Working set of any process must be resident
  - This automatically imposes a limit
- Global replacement
  - No working set concept
  - Use CPU utilization as a criterion
  - With too many processes, thrashing occurs

Figure 8-11

$L = \text{mean time between faults}$
$S = \text{mean page fault service time}$
Load Control and Thrashing

• How to find $N_{\text{max}}$?
  – $L=S$ criterion:
    • *Page fault service time* $S$ needs to keep up with *mean time between page faults* $L$
  – 50% criterion:
    • CPU utilization is highest when paging disk is 50% busy (found experimentally)
Load Control and Thrashing

• Which process to deactivate
  – Lowest priority process
  – Faulting process
  – Last process activated
  – Smallest process
  – Largest process

• Which pages to load when process activated
  – Prepage last resident set

Figure 8-12
Evaluation of Paging

Prepaging is important
– Initial set can be loaded more efficiently than by individual page faults

Figure 8-13(a)
Evaluation of Paging

Page size should be small. However, small pages need

- Larger page tables
- More hardware
- Greater I/O overhead

Figure 8-13(b)  Figure 8-13(c)
Evaluation of Paging

Load control is important

\[ W = \text{Minimum amount of memory to avoid thrashing.} \]
History
• Originally developed by Steve Franklin
• Modified by Michael Dillencourt, Summer, 2007
• Modified by Michael Dillencourt, Spring, 2009
• Modified by Michael Dillencourt, Winter, 2010