Page Replacement Algorithms
(Virtual Memory)
Servicing a Page Fault

1. running process
2. page table
3. operating system
4. physical memory
5. free frame
6. disk

03/31/2010  CSCI 315 Operating Systems Design
No free frame: now what?

• **Page replacement:** Are all those pages in memory being referenced? Choose one to swap back out to disk and make room to load a new page.
  
  – **Algorithm:** How you choose a victim.
  – **Performance:** Want an algorithm which will result in minimum number of page faults.

• Side effect: The same page may be brought in and out of memory several times.
Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.

- Use *modify (dirty) bit* to reduce overhead of page transfers – only modified pages are written to disk.

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.
Basic Page Replacement

1. Find the location of the desired page on disk.

2. Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a page replacement algorithm to select a victim frame.

3. Read the desired page into the (newly) free frame.
   Update the page and frame tables.

4. Restart the process.
Page Replacement Algorithms

- **Goal**: Produce a low page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (*reference string*) and computing the number of page faults on that string.
- The reference string is produced by tracing a real program or by some stochastic model. We look at every address produced and strip off the page offset, leaving only the page number. For instance:

```
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
```
Graph of Page Faults Versus The Number of Frames
FIFO Page Replacement

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.
- 3 frames (3 pages can be in memory at a time per process)
  
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

  9 page faults

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- 4 frames

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

  10 page faults

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- FIFO Replacement ⇒ Belady’s Anomaly: more frames, more page faults.
FIFO Page Replacement

<table>
<thead>
<tr>
<th>reference string</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
</tr>
<tr>
<td>7 7 7 2 2 2 4 4 4 0 0 7 7 7</td>
</tr>
<tr>
<td>0 0 0 3 3 3 2 2 2 1 1 1 0 0</td>
</tr>
<tr>
<td>1 1 1 0 0 0 3 3 3 2 2 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>page frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 3 3 3 2 2 2 1</td>
</tr>
</tbody>
</table>
FIFO (Belady’s Anomaly)
Optimal Algorithm

- Replace the page that will not be used for longest period of time. (How can you know what the future references will be?)
- 4 frames example:  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

```
1  2  3  4
   6 page faults
5
```
- Used for measuring how well your algorithm performs.
# Optimal Page Replacement

<table>
<thead>
<tr>
<th>Reference String</th>
<th>Page Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1</td>
<td>7 0 0 0 1 1 3 3 3 1 1</td>
</tr>
</tbody>
</table>

03/31/2010   CSCI 315 Operating Systems Design
LRU Algorithm

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

```
1  5
2
3  5  4
4  3
```

- Counter implementation:
  - 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.
LRU Page Replacement

<table>
<thead>
<tr>
<th>reference string</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 0 1 2 0 3 0 2 2 4 4 4 0 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>page frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 3 3 3 2 2 2 2 2 2 7</td>
</tr>
</tbody>
</table>
LRU Algorithm (Cont.)

• 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

reference string

4 7 0 7 1 0 1 2 1 2 7 1 2

stack before a  stack after b

2 7
1 2
0 1
7 0
4 4

a b

03/31/2010 CSCI 315 Operating Systems Design 16
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.

• **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

(a)

(b)

circular queue of pages

reference bits

pages

reference bits

pages

next victim

1

1

1

0

1

1

0

1

1

0

1

0

1

1
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 a minimum number of pages.

- There are two major allocation schemes:
  - fixed allocation
  - priority allocation
Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of process.
  
  - $s_i = \text{size of process } p_i$
  - $S = \sum s_i$
  - $m = \text{total number of frames}$
  - $a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

\[
\begin{align*}
  m &= 64 \\
  s_1 &= 10 \\
  s_2 &= 127 \\
  a_1 &= \frac{10}{137} \times 64 \approx 5 \\
  a_2 &= \frac{127}{137} \times 64 \approx 59
\end{align*}
\]
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” pages, the page-fault rate is very high. This leads to:
  - Low CPU utilization.
  - Operating system thinks that it needs to increase the degree of multiprogramming.
  - Another process added to the system.

- **Thrashing** ≡ a process is busy swapping pages in and out.
Thrashing

- Why does paging work?
  Locality model
  - Process migrates from one locality to another.
  - Localities may overlap.
- Why does thrashing occur?
  \( \Sigma \) size of locality > total memory size
Locality in Memory-Reference Pattern
Working-Set Model

- \( \Delta \equiv \text{working-set window} \equiv \) a fixed number of page references.

- \( WSS_i (\text{working set of Process } P_i) = \) 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 \( \Delta = \infty \Rightarrow \) will encompass entire program.

- \( D = \sum WSS_i \equiv \) total demand frames

- if \( D > m \Rightarrow \text{Thrashing} \)

- Policy if \( D > m \), then suspend one of the processes.
Working-set model

page reference table

\[
\begin{array}{cccccccccccccccccccccc}
\ldots & 2 & 6 & 1 & 5 & 7 & 7 & 7 & 5 & 1 & 6 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 4 & 3 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & \ldots \\
\end{array}
\]

\[\Delta\]

\[t_1\]

\[\Delta\]

\[t_2\]

\[WS(t_1) = \{1, 2, 5, 6, 7\}\]

\[WS(t_2) = \{3, 4\}\]
Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10,000$
  - Timer interrupts after every 5000 time units.
  - Keep in memory 2 bits for each page.
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set.
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units.
Page-Fault Frequency Scheme

Establish “acceptable” page-fault rate.
- If actual rate too low, process loses frame.
- If actual rate too high, process gains frame.