Lecture XIV
Virtual Memory - II

Teofik Koşar

Louisiana State University
March 27th, 2008

Background

- Virtual memory - separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed ⇒ reference to it
  - invalid reference ⇒ abort
  - not-in-memory ⇒ bring to memory

Valid-Invalid Bit

- With each page table entry a valid-invalid bit is associated
  (1 ⇒ in-memory and legal, 0 ⇒ not-in-memory or invalid)
- Initially valid-invalid bit is set to 0 on all entries
- Example of a page table snapshot:

  Frame # | valid-invalid bit
  ---|---
          | 1
          | 1
          | 1
          | 0
          | 0
          | 0

- During address translation, if valid-invalid bit in page table entry is 0 ⇒ page fault

Page Fault

- If there is ever a reference to a page, first reference will trap to OS ⇒ page fault
- OS looks at another table to decide:
  - Invalid reference ⇒ abort
  - Just not in memory.
- Get empty frame.
- Swap page into frame.
- Reset tables, validation bit = 1.
- Restart instruction: Least Recently Used
  - block move
  - auto increment/decrement location

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
Page Replacement Algorithms

• Want lowest 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
• In all our examples, the reference string is 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

First-In-First-Out (FIFO) Algorithm

• 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)

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

9 page faults

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

10 page faults

FIFO Replacement - Belady’s Anomaly
- more frames → more page faults

Least Recently Used (LRU) Algorithm

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

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

9 page faults

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

10 page faults

LRU Approximation Algorithms

• Reference bits
  - 1 byte for each page: eg. 00100111
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Shift right at each time interval
  - Replace the one with smaller value

• Second chance
  - Need single 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

Use Of A Stack to Record The Most Recent Page References

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Frame 2</th>
<th>Frame 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

9 page faults

Second-Chance (clock) Page-Replacement Algorithm

- Need single 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
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

Optimal Algorithm

- Replace page that will not be used for longest period of time
- 4 frames example
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Acknowledgements


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

- R. Doursat and M. Yuksel from UNR