Lecture - XVIII

Virtual Memory

Tevfik Koşar

Louisiana State University
October 29th, 2009

Roadmap

- Virtual Memory
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms (FIFO, LRU, Optimal etc)
  - Performance of Demand Paging
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:

<table>
<thead>
<tr>
<th>Frame #</th>
<th>valid-invalid bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- 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 not in memory, first reference will trap to OS ⇒ page fault
- OS looks at another table (in PCB) to decide:
  - Invalid reference ⇒ abort.
  - Just not in memory. ==> page-in
- Get an empty frame.
- Swap (read) page into the new frame.
- Set validation bit = 1.
- Restart instruction
Steps in Handling a Page Fault

What happens if there is no free frame?

- Page replacement - find some page in memory, but not really in use, swap it out
  - Algorithms (FIFO, LRU ..)
  - performance - want an algorithm which will result in minimum number of page faults
- Same page may be brought into 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

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></th>
<th></th>
<th>1</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>9 page faults</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
FIFO Illustrating Belady’s Anomaly

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)

```
  1 4 5
  2 1 3
  3 2 4
```
9 page faults

- 4 frames

```
  1 5 4
  2 1 5
  3 2
  4 3
```
10 page faults

- FIFO Replacement - Belady’s Anomaly
  - more frames ⇒ more page faults
Performance of Demand Paging

- Page Fault Rate 0 ≤ p ≤ 1.0
  - if p = 0 no page faults
  - if p = 1, every reference is a fault

- Effective Access Time (EAT)
  \[ EAT = (1 - p) \times \text{memory access} + p \times (\text{page fault overhead} + [\text{swap page out}] + \text{swap page in} + \text{restart overhead}) \]

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out
- Swap Page Time = 10 msec = 10,000 microsec

- EAT = (1 - p) \times 1 + p \times (10,000 + 1/2 \times 10,000) = 1 + 14,999 \times p \quad \text{(in microsec)}

- What if 1 out of 1000 memory accesses cause a page fault?
- What if we only want 30% performance degradation?
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>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

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

Summary

- Virtual Memory
  - Demand Paging
  - Page Faults
  - Page Replacement
  - Page Replacement Algorithms
    - (FIFO, LRU, Optimal etc)
  - Performance of Demand Paging

- **Reading Assignment:** Chapter 8 from Silberschatz.
Acknowledgements

• “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne

• “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings

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

• R. Doursat and M. Yuksel from UNR