Lecture - XIV

Virtual Memory - I

Tevfik Koşar

Louisiana State University
March 13th, 2008

Roadmap

- Virtual Memory
  - page replacement algorithms
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

Transfer of a Paged Memory to Contiguous Disk Space
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>...</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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, 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

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 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)

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

- 4 frames

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

- FIFO Replacement - Belady’s Anomaly
  - more frames ⇒ more page faults
Graph of Page Faults Versus The Number of Frames

FIFO Illustrating Belady’s Anomaly
Least Recently Used (LRU) Algorithm

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

• Needs hardware assistance
• 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 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
Prepaging

• prevent large number of page faults by initial paging of unreferenced pages as well
  - all pages they may be needed
  - e.g. all pages for small files

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