Virtual Memory Management

B. Ramamurthy

Demand Paging

Mapping between main memory and virtual memory is given by a page table.

LAS - Logical Address Space
Virtual address space

LAS 0
LAS 1
LAS 2

Executable code space

Main memory (Physical Address Space - PAS)
Page Tables (1)

Internal operation of MMU with 16 4 KB pages

Page Tables (2)

32 bit address with 2 page table fields
Two-level page tables
Page Tables (3)

Typical page table entry

- Caching disabled
- Modified
- Present/absent
- Page frame number
- Referenced
- Protection

Page Fault Handling (1)

- Hardware traps to kernel
- General registers saved
- OS determines which virtual page needed
- OS checks validity of address, seeks page frame
- If selected frame is dirty, write it to disk
Page Fault Handling (2)

- OS brings schedules new page in from disk
- Page tables updated
- Faulting instruction backed up to when it began
- Faulting process scheduled
- Registers restored
- Program continues

Backing Store

(a) Paging to static swap area
(b) Backing up pages dynamically
Sharing Pages: a text editor

Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

1. Process creation
   - determine program size
   - create page table
2. Process execution
   - MMU reset for new process
   - TLB flushed
3. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages
Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page

- Modified page must first be saved
  - unmodified just overwritten

- Better not to choose an often used page
  - will probably need to be brought back in soon

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable

- Estimate by ...
  - logging page use on previous runs of process
  - although this is impractical
Not Recently Used Page Replacement Algorithm

- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class

FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - in order they came into memory
- Page at beginning of list replaced

- Disadvantage
  - page in memory the longest may be often used
The Clock Page Replacement Algorithm

When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
- R = 0: Evict the page
- R = 1: Clear R and advance hand

Least Recently Used (LRU)

- Assume pages used recently will be used again soon
  - throw out page that has been unused for longest time

- Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference!!

- Alternatively keep counter in each page table entry
  - choose page with lowest value counter
  - periodically zero the counter
Simulating LRU in Software (1)

LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

Simulating LRU in Software (2)

The aging algorithm simulates LRU in software
Note 6 pages for 5 clock ticks, (a) – (e)
Working-Set Model

- \( \Delta \) = working-set window = a fixed number of page references
  
  Example: 10,000 instruction

- \( WSS_i \) (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.

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

- if \( D > m \) \( \Rightarrow \) Thrashing

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

---

Working-set model

![Page reference table](image)

- 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.

The Working Set Page Replacement Algorithm (2)

The working set algorithm
The WSClock Page Replacement Algorithm

Operation of the WSClock algorithm

Review of Page Replacement Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>Not implementable, but useful as a benchmark</td>
</tr>
<tr>
<td>NRU (Not Recently Used)</td>
<td>Very crude</td>
</tr>
<tr>
<td>FIFO (First-In, First-Out)</td>
<td>Might throw out important pages</td>
</tr>
<tr>
<td>Second chance</td>
<td>Big improvement over FIFO</td>
</tr>
<tr>
<td>Clock</td>
<td>Realistic</td>
</tr>
<tr>
<td>LRU (Least Recently Used)</td>
<td>Excellent, but difficult to implement exactly</td>
</tr>
<tr>
<td>NFU (Not Frequently Used)</td>
<td>Fairly crude approximation to LRU</td>
</tr>
<tr>
<td>Aging</td>
<td>Efficient algorithm that approximates LRU well</td>
</tr>
<tr>
<td>Working set</td>
<td>Somewhat expensive to implement</td>
</tr>
<tr>
<td>WSClock</td>
<td>Good efficient algorithm</td>
</tr>
</tbody>
</table>
Modeling Page Replacement Algorithms

Belady's Anomaly

FIFO with 3 page frames
FIFO with 4 page frames
P's show which page references show page faults

Stack Algorithms

State of memory array, $M$, after each item in reference string is processed
Page Size (1)

Small page size

**Advantages**
- Less internal fragmentation
- Better fit for various data structures, code sections
- Less unused program in memory

**Disadvantages**
- Programs need many pages, larger page tables

Page Size (2)

**Overhead due to page table and internal fragmentation**

\[
\text{overhead} = \frac{s \cdot e}{p} + \frac{p}{2}
\]

**Where**
- \( s \) = average process size in bytes
- \( p \) = page size in bytes
- \( e \) = page entry

Optimized when

\[
p = \sqrt{2se}
\]
**TLBs – Translation Lookaside Buffers**

<table>
<thead>
<tr>
<th>Valid</th>
<th>Virtual page</th>
<th>Modified</th>
<th>Protection</th>
<th>Page frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>140</td>
<td>1</td>
<td>RW</td>
<td>31</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>R X</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>130</td>
<td>1</td>
<td>RW</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
<td>1</td>
<td>RW</td>
<td>62</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
<td>R X</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>0</td>
<td>R X</td>
<td>45</td>
</tr>
<tr>
<td>1</td>
<td>860</td>
<td>1</td>
<td>RW</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>861</td>
<td>1</td>
<td>RW</td>
<td>75</td>
</tr>
</tbody>
</table>

A TLB to speed up paging

**Inverted Page Tables**

Comparison of a traditional page table with an inverted page table