Lecture Topics

- Today: Virtual Memory
  (Stallings, chapter 8.1-8.4)
- Next: continued

Announcements

- Self-Study Exercise #7 (this week)
- Project #5 (due 3/2)
- Project #6 (due 3/16)
Virtual Memory

- Memory management technique where virtual addresses (logical addresses as seen by a process) are mapped to physical addresses (actual addresses in RAM).

- Only a subset of the virtual address space for a process needs to be in RAM for it to execute (the virtual addresses which are currently being used by the process).

Virtual Memory: Costs and Benefits

- Every virtual address must be translated to a physical address (takes time, needs hardware support).

- CPU utilization can be increased by having more processes resident in RAM.

- The logical address space of a process can be larger than the physical address space of the system.
Virtual Memory and Paging

- Most virtual memory systems use paging: keep a subset of the pages in primary storage (RAM) and the remainder in secondary storage (disk).

- When necessary, move pages from disk to RAM (and from RAM to disk).

- Page table must keep track of current location of each page (RAM or disk).
Page Table

Page table used to map page number (from virtual address) to frame number (for physical address)

![Page Table Diagram]

Each page table entry contains:

- Valid bit (PTE is being used)
- Present bit (page present in RAM)
- Modified bit (page has been modified)
- Other control bits (ex: access permissions)
- Frame number
Page Table

Example: 32 bits to 30 bits
Address Translation (during execution)

Efficiency Considerations

Page table is often quite large

- Keep part of page table on disk (page table uses virtual memory, just like user data structures)
- Use multi-level page tables?
- Use inverted page tables?
Efficiency Considerations

Address translation is too slow if page table is kept in RAM (or worse yet, on disk)

- TLB (Translation Look-aside Buffer): special cache of page table entries

- First memory access requires look-up in page table, but subsequent accesses use TLB (until that PTE gets evicted)

TLB organizations

(a) Direct mapping
(b) Associative mapping
TLB Processing

- Hit in TLB: map page number to frame (within 1 clock cycle)
- Miss in TLB, page in RAM: load page table entry into the TLB, then restart instruction (10-100 clock cycles)
- Miss in TLB, page not in RAM: page fault (millions of clock cycles)
Page Fault Processing

- Move process from Running to Blocked
- Choose page to replace (if all frames full)
- If victim page has been modified ("dirty"), copy victim page from RAM to disk
- Copy page from disk to RAM
- Update page table entry
- Move process from Blocked to Ready

Efficiency Considerations

- A page fault takes millions of clock cycles to process
- OS handles page faults (not hardware)
- Algorithm to select victim page must be effective (make good choice)
OS Policies

- Fetch policy
- Placement policy
- Replacement policy

Fetch Policy

Determines when a page should be brought into main memory

- Demand paging: copy a page into main memory when a reference is made to a location on that page
- Prepaging: copy more than one page into main memory in response to a page fault
Demand Paging

- When a reference is made to a virtual address on that page, the page is copied from disk to RAM (if it is not already present).
- When a process first begins executing, there will usually be a flurry of page faults.
- Once enough pages have been loaded into RAM, the page fault rate will drop to a very low level (locality of reference).

Prepaging

- More than one page is copied into RAM at once when a page fault occurs.
- Takes advantage of characteristics of disk drives (relatively inexpensive to access adjacent disk sectors).
- Not particularly effective: difficult to predict which pages to load, some pages loaded unnecessarily.
Placement Policy

- Determines where to place page that is being loaded from disk.
- Paging: all frames can be accessed in the same amount of time, so not relevant
- Segmentation: important for same reasons as variable partitioning

Replacement Policy

- Determines which page should be selected as the "victim", if there are no empty frames

Related ideas (discussed later):
  - how many frames should be allocated to the current process?
  - should the victim be selected from within the frames allocated to the process, or from within all frames?
Replacement Policy

- **Optimal:** the page which will not be used for the longest period of time is selected as the victim (results in fewest page faults)
- Not practical to implement – OS does not have perfect knowledge of future events
- Useful as benchmark

Example

- Process with 5 pages, 3 frames allocated

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>F</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

- Page address stream condensed: many references to same page
- Diagram only shows faults which result in replacement of a page ("F")
Least Recently Used (LRU)

- Replace the page which has not been used for the longest time
- Locality of reference suggests that page will not be used in the near-term future
- Impractical to implement: expensive to tag each page with time of last reference

Example (cont)

Compare optimal and LRU:

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
<tr>
<td>LRU</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>
First-In, First-Out (FIFO)

- Replace the page which has been in RAM for the longest time
- If a page has been in RAM for a long time, perhaps it is no longer needed (not always true)
- Easy to implement: treat page frames as a circular buffer

Example (cont)

Compare optimal, LRU and FIFO:

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>OPT</th>
<th>LRU</th>
<th>FIFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

32
Clock

- LRU not practical, FIFO performs poorly
- Find an algorithm which approximates LRU, but can be implemented easily
- Clock policy: bit associated with each frame to help keep track of the fact that it has been used recently (the use bit or referenced bit)

Clock

- Use bit set to 1:
  - when page is initially loaded into RAM
  - whenever the page is referenced again (used)
- Use bit set to 0:
  - when OS scans for a page to replace
Clock

Example (cont)

<table>
<thead>
<tr>
<th>Page address stream</th>
<th>2</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>3</th>
<th>2</th>
<th>5</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>LRU</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>FIFO</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>CLOCK</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Experimental Comparisons

- Fixed number of frames, victim selected locally
- FIFO worse than Optimal by a factor of 2
- Clock approximates LRU (particularly if improved)

Advanced Algorithms

- Many variations on the Clock algorithm: "second chance" algorithms
- Ideal victim: page has not been modified (does not have to be copied to disk) and has not been referenced recently
- Paging system already has "modified" bit; check that bit as part of replacement algorithm
Advanced Algorithms (cont)

- Four combinations
  
<table>
<thead>
<tr>
<th>U</th>
<th>M</th>
</tr>
</thead>
</table>
  | 0 | 0 | Not used recently, not modified
  | 0 | 1 | Not used recently, modified
  | 1 | 0 | Used recently, not modified
  | 1 | 1 | Used recently, modified

- Better to select page with $U = 0$, even if $M = 1$ (unlikely to be referenced again in the near future)

UNIX: Variation on Clock

- Front hand sets $U$ bit back to 0, back hand looks for pages where $U$ is still 0
- Scanrate – rate at which the two hands scan the list (pages per sec)
- Handspread – the gap between front hand and backhand