CS 355
Operating Systems
Virtual Memory and Paging

Virtual Memory

• Split into overlays.
  – done by programmer
• Virtual memory: the OS is in charge of keeping parts of the program in use in memory, and the rest in disk.
• Requires an MMU (Memory Management Unit)
• Memory is mapped on a per process basis

Virtual Memory

The position and function of the MMU

Virtual Addresses

• Virtual memory gives a complete separation of logical and physical addressing spaces
• Typically a virtual address is 32-bit or 64-bit
  – allows a process to have 4GB/16EB of virtual memory
• Virtual Memory Structures:
  – paging: divide the address space into fixed-sized pages
  – segmentation: divide the address space into variable-sized segments

Paging

• Virtual memory is divided into fixed-sized units called pages
  – typically, $2^{20}$ pages in virtual memory for 32-bit
• Physical memory also divided into fixed-sized units called page frames
  – on Pentium, each page frame is 4KB
• Pages and page frames are always the same size
• Transfers between memory and disk is always in units of a page

Paging

The relationship between virtual addresses and physical memory addresses given by the page table
Virtual Address

• A virtual address is really a pair \((p, o)\) of addresses
  – Low-order bits give an offset within the page
  – High-order bits specifies page number – this is the part that gets translated
• The job of the MMU is to translate the page number \(p\) to a frame number
  – The physical address is then \((f, o)\) – this is what goes on the memory bus
• For every process, there is a page table
  – \(p\) is just an index into this array for the translation

Page Table Issues

• The page table can be extremely large
  – Modern computers use virtual addresses of at least 32-bits – with 4KB page size, this gives \(2^{20} \approx 1\) million pages
  – Recall that each process keeps its own page table!
• The mapping must be fast
  – virtual-to-physical mapping is computed how often?

Paging in Linux

Linux uses three-level page tables

Page Table Entry

Typical page table entry

Internal operation of MMU with 16 4-KB pages.

Multilevel Page Tables

• 32 bit address with 2 page table fields
• Two-level page tables
Translation Lookaside Buffers (TLB)

- The page tables are kept in memory
- Access to memory is slow compared to CPU
- Page tables generate many extra memory references
- Programs tend to make a large number of references to a small number of pages
- TLB is a small hardware device for mapping virtual addresses to physical addresses without going through page table.

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>

Inverted Page Tables

Comparison of a traditional page table with an inverted page table

Page Replacement Algorithms

- When a virtual address translates to a page not currently in memory, it generates a page fault
- Page fault forces choice
  - need to make room for incoming page
  - which page must be removed?
- 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 used often

Second Chance Page Replacement Algorithm

- Operation of a second chance
  - pages sorted in FIFO order
    - Page list if fault occurs at time 20, A has R bit set
      (numbers above pages are loading times)

The Clock Page Replacement Algorithm

- Assume pages used recently will 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 Hardware

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