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)
Simple Paging

- The address space of a process is viewed as a sequence of pages
- Each page is placed in a page frame in physical memory

Simple Paging

- OS manages a page table for each process
- Page table maps logical addresses to physical addresses
Simple Paging

- A logical address in a process is viewed as a page number and byte offset
  - Assume addresses are 16 bits
  - Assume pages are 1 K (1024 bytes)
  - Addresses: \texttt{PPPPPPxxxxxxxxxx}

- OS maintains a page table for each process to map page numbers to frame numbers

Simple Paging

Every logical address must be mapped to a physical address (page number is the index into the page table)
Paging: Address Translation

- Pages and page frames must be same size

- Use page size which is a power of 2 (allows logical address to be subdivided easily)

- Logical addresses and physical addresses don’t have to be the same size (page number and frame number can have different number of bits)

Paging: Address Translation

- Assume 15-bit logical addresses and 28-bit physical addresses

- Assume 12-bit page displacements
### Example

Current page table:    Address translations:

<table>
<thead>
<tr>
<th>Page</th>
<th>P</th>
<th>Frame</th>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0961</td>
<td>3c34</td>
<td>0754c34</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>09E2</td>
<td>3590</td>
<td>0754590</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>E493</td>
<td>3000</td>
<td>0754000</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>B263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0685</td>
<td>3FFF</td>
<td>0754FFF</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0755</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example (2)

Current page table:    Address translations:

<table>
<thead>
<tr>
<th>Page</th>
<th>P</th>
<th>Frame</th>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0961</td>
<td>3500</td>
<td>0754500</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0751</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>09E2</td>
<td>2500</td>
<td>09E2500</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0754</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>E493</td>
<td>020C</td>
<td>096120C</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>B263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0685</td>
<td>6744</td>
<td>not valid</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0755</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OS manages page table in RAM

Direct mapped: page number is index

Problem: page table can be large:
  • P bits in page number, $2^P$ page table entries
  • All page table entries allocated (even if unused)

Solution: use multi-level page tables or similar technique to reduce table size

Problem: page table in RAM, so every address translation requires a memory access (doubles the number of accesses)

Solution: cache of page table entries (TLB – translation look-aside buffer)

Typical configuration:
  • Instruction cache (instructions)
  • Data cache (data values)
  • TLB (page table entries)
Summary: Simple Paging

- Transparent to user: system manages paging without user having to be aware
- Simple for the OS to manage
- Supports swapping: page frames copied to and from disk
- No external fragmentation (but there is internal fragmentation)
Virtual Memory

- Recall the principle of locality of reference: in the short term, most references to addresses are within a subset of the entire address space of a process

- Therefore, only a subset of the entire address space needs to be in RAM for a process to execute

Virtual Memory

- Assume virtual memory system uses 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)
Virtual Memory

Virtual Memory: Benefits

- More processes can be in primary storage, ready to run. Thus, CPU utilization can be increased.

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

- Each virtual address must be translated to a physical address, which takes time and requires hardware support.
- A process will get interrupted arbitrarily because some page is not in RAM; the OS will block the process until that page has been copied from disk to RAM.

Virtual Memory with Paging

- Virtual address: page number and offset
- Physical address: frame number and offset
- Page table: map page to frame
Virtual Memory with Paging

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

- Assume 32-bit virtual addresses, 30-bit physical addresses, and 12-bit offsets
  - Page numbers: 20 bits
  - Frame numbers: 18 bits
- Virtual address: 0001704C
- Physical address: 2513204C (assuming that page 00017 is in frame 25132)
Example (2)

Virtual address space: $2^{32}$ bytes

Physical address space: $2^{30}$ bytes

Page number used to index page table:

- 20 bits, so $2^{20}$ entries (1,048,576)

Page table entry: 4 bytes (one word)

- frame number: 18 bits
- control bits: 3+ bits

Example (3)
Efficiency Considerations

- Page table for each process is large
  (4 MB in example)

- 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

- Too slow: with page table, every memory access takes twice as long (access page table in RAM, then access the desired address in RAM)

- 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
**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)
- TLB usually fully associative
- Algorithm to select victim page must be effective