Lecture Topics

- Today: The Memory Hierarchy
  (P&H 5.1-5.11)
- Next: continued

Announcements

- Milestone #7 (due 3/23)
- Milestone #8 (due 4/6)
Exam #2

- Wednesday, 3/29 (3:00-4:20 PM)
- Location (see email):
  - 226 Erickson Hall
  - 1257 Anthony Hall
- 80 minutes, 18% of course grade
- Study suggestions on course website

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
Paging

- RAM is managed as sequence of page frames (all the same size)
- Each process is viewed as integer number of pages (same size as page frames)
- A process no longer needs a large contiguous block of physical memory; instead it is given a set page frames (which might be scattered around RAM)

Paging

- OS manages a page table for each process
- Page table maps logical addresses to physical addresses
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: \( \text{PPP} \ldots \text{ xxxxxxx} \)

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

Paging

Every logical address must be mapped to a physical address (page number is the index into the page table)
Example

- System uses 1 Kbyte pages
- Process is 2.7 Kbytes long
- OS views process as 3 pages
- Last page is not fully used (internal fragmentation)
- Logical address viewed as page number and byte offset

Paging

OS can manage RAM efficiently

- unnecessary to find large contiguous blocks of RAM – each process subdivided into pages
- all page frames are identical and can be used interchangeably
- processes can be swapped out to disk, swapped back in later – and placed in different page frames
Paging: Address Translation

- Pages and page frames 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 Example

- Logical address has 16 bits:
  - Page: 4 bits to identify page
  - Offset: 12 bits to identify byte within page
- Physical address has 32 bits:
  - Frame: 20 bits to identify frame
  - Offset: 12 bits to identify byte within frame
Paging Example (2)

- Assume page 5 is in frame 20CE7
- Logical address 5408 maps to physical address 20CE7408:

![Diagram](image)

Paging Example (3)

- How many entries in the page table?
  - page number is 4 bits
  - $2^4 = 16$ entries

- How many bytes in one page (and one frame)?
  - page offset is 12 bits
  - $2^{12} = 4096$ bytes
Paging Example (4)

- How many frames in RAM?
  - frame number is 20 bits
  - $2^{20} = 1,048,576$ frames

- How large is RAM?
  - physical address is 32 bits
  - $2^{32} = 4,294,967,296$ bytes = 4 GB

---

Paging Example (5)

<table>
<thead>
<tr>
<th>I</th>
<th>V</th>
<th>Frame</th>
<th>I</th>
<th>V</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>FF641</td>
<td>8</td>
<td>0</td>
<td>0004A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>00014</td>
<td>9</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0003A</td>
<td>A</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>FF593</td>
<td>B</td>
<td>0</td>
<td>FFF7C</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>FFF7C</td>
<td>C</td>
<td>0</td>
<td>00EA1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>20EC7</td>
<td>D</td>
<td>0</td>
<td>00028</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>00014</td>
<td>E</td>
<td>0</td>
<td>0003A</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>00014</td>
<td>F</td>
<td>0</td>
<td>0003A</td>
</tr>
</tbody>
</table>
Paging Example (6)

- Consider the entry at index 2:
  
  1  0003A

  First and last logical addresses?
  
  first address:  2000  
  last address:   2FFF
  
  First and last physical addresses?
  
  first address:  0003A000  
  last address:  0003AFFF

Paging Example (7)

- Request for logical address 3A14

  Page 3, so check entry at index 3:
  
  1  FF593

  V = 1, so mapping is valid
  
  physical address FF593A14
Paging Example (8)

- Request for logical address 92E8
  
  Page 9, so check entry at index 9:
  
  0 00028

  V = 0, so mapping is not valid
  invalid logical address

Page Table Implementation

- 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
Page Table Implementation

- 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)
Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes (text editors, compilers, etc)
  - Shared code must appear in same location in the logical address space of all processes

- Private code and data
  - Each process has a separate copy of the private code and data (can be anywhere in logical address space)
Summary: 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)

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:

<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>

### Address translations:

<table>
<thead>
<tr>
<th>Logical</th>
<th>Physical</th>
</tr>
</thead>
<tbody>
<tr>
<td>3500</td>
<td>0754500</td>
</tr>
<tr>
<td>2500</td>
<td>09E2500</td>
</tr>
<tr>
<td>020C</td>
<td>096120C</td>
</tr>
<tr>
<td>6744</td>
<td>not valid</td>
</tr>
</tbody>
</table>

---

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 Address
| Page Number | Offset |
```

Virtual Memory with Paging

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

```
Page Table Entry
| Other Control Bits | Frame Number |
```
Virtual Memory with Paging

- 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 (4)

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, 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