Lecture 13
Memory Management (contd.)

March 21, 2001

Outline

• Announcements
  – Lab 4 due March 28th, demos on 29th, 30th (please sign up)
  – Lectures 14 and 15 will be given by Eric Freudenthal
  – Questions?

• Memory management
  – Paging (cont’d)
  – Segmentation
  – Segmentation with paging
  – Example memory organizations

[Silberschatz/Galvin: Sections 8.6 – 8.8]
[Review] Page Table Structure

- Page table typically stored in memory
  - a single page table base register that
    - points to the beginning of the page table
    - $pi$ is now the offset into this table
  - problem
    - requires two accesses to memory for each value
    - even with caches, can become very slow

- Solution: Translation Lookaside Buffer (TLB)
  - a portion of the page table is cached in the TLB
    - little performance degradation if a value is a hit in the TLB
    - if not: a memory access is needed to load the value into the TLB
      - an existing value must be flushed if the TLB is full
  - E.g.: Average memory access time for a system with 90% hit rate in TLB
    $\text{Access}_{\text{prop TLB}} = 0.9\times(\text{Access}_{\text{TLB}} + \text{Access}_{\text{mem}}) + 0.1\times(\text{Access}_{\text{mem}} + \text{Access}_{\text{prop mem}})$
    $\approx 1.1\times(\text{Access}_{\text{mem}})$

Multi-level Page Tables

- Rationale: Modern systems support a very large logical address space
  - page tables themselves become very large
    - e.g., for a system with 32-bit logical addresses and 4K pages
      - we need $2^{30}$ page table entries (4 bytes per PTE implies 4 MB of space)
  - Solution: page the page table itself
    - cost: additional memory accesses (but caching helps)

Page Tables and Sharing

- Page tables permit different virtual addresses (frames of different processes) to map to the same physical address
  - convenient sharing of common code (dynamically-linked system libraries)
  - shared data segments for IPC

Inverted Page Tables

- Observation
  - usually, only a portion of all the pages from the system’s memory can be stored in the physical memory
  - so while the required page table for all of logical memory might be massive, only have a small subset of it contains useful mappings

- We can take advantage of this fact in both TLB and page table design
Inverted Page Tables (cont’d)

- **Efficiency considerations**
  - the inverted page table is organized based on physical addresses via frame numbers
    - searching for the frame number can be very slow
  - use a hash table based on
    - the PID and logical page number as keys
  - recently located entries of the inverted page table can be stored in a TLB-like structure based on associative registers

- **Main disadvantage of inverted page tables:** sharing
  - each process that shares an object will have its own (disjoint) space where the shared object is mapped
  - not possible to maintain with standard inverted page tables
    - since space for only one \(<\text{PID}, \text{page number}>\) tuple

Memory Mapping: Protection Issues with Paging

- **Partition protection scheme**
  - Check that address lies between base and base+limit
  - Cannot be used on page-based systems: WHY?

- **Special bits in the page table entry enforce per-frame protection**
  - an **accessibility** bit
    - whether a page is invalid, readable, writable, executable
  - a **valid/invalid** bit to indicate whether a page is in the user’s (logical) space
  - system support
    - all addresses are interpreted by the MMU
    - OS intervention required to manipulate page tables and TLBs

- **Sometimes, the hardware may support a page-table length register**
  - specifies size of the process page table
    - trailing invalid pages can be eliminated
    - useful when processes are using a small fraction of available address space

Memory Mapping: Segmentation

- A segment is a **logical** piece of the program
  - e.g., the code for the program functions, its data structures, symbol tables

- Segmentation views logical memory is broken into such segments
  - segments are of **variable size** (unlike pages)

- **Accessing a segment**
  - the logical address is regarded as two-dimensional
    - a segment pointer to an entry in the **segment table**
    - a displacement into the segment itself

- **Allocating a segment**
  - a segment is a **partition with a single base-limit pair**
    - the limit attribute stores the segment length
      - prevents programs from accessing locations outside the segment space

Memory Mapping: Segment Table Lookup

- **Mapping logical addresses to physical addresses**
  - the mapping is maintained by the **segment table**
  - the segment number s# is used to **index** into the (process”) segment table
    - where the corresponding segment size and base address are stored
Memory Mapping: Segmentation Hardware

- **Segment registers**
  - some designs (e.g. Intel x86) provide registers to identify segments
    - loading a segment register loads a (hidden) segment specification register from the segment table
    - construction of the logical address is done explicitly
- **TLBs**
  - some designs, such as the MIPS 2000, only provide a TLB
    - the OS is responsible for loading this, and doing appropriate translation
- **Traditional approach: Store the segment table in memory**
  - segment table base register (STBR), segment table length register (STLR)
    - saved and restored on each context switch
  - translation of address \( (s,d) \)
    - check that \( s \) is valid: \( s < \text{STLR} \)
    - Look up base address, limit: segment table entry at address \( (\text{STBR} + s) \)
    - check that offset \( d \) is valid: \( d < \text{length} \)
    - compute physical address

Segmentation: Pros and Cons

- **Pros**
  - protection in terms of ensuring that illegal address accesses are avoided, comes for free
    - the segment length check plays an important role here
  - sharing segments across programs is straightforward by loading identical segment table base register values
- **Cons**
  - external fragmentation is potentially a big problem
  - contrast this with paging where only internal fragmentation is possible

Memory Mapping: Segmentation and Paging

- **Overlay a segmentation scheme on a paging environment**
  - several examples
    - originally proposed for GE 645 / Multics
    - Intel x86 uses segment registers to generate 32-bit logical addresses, which are translated to physical addresses by an optional multi-level paging scheme
  - alleviates the problem of external fragmentation

Memory Mapping: Examples

- **Multics (c. 1965)**
  - 34-bit logical address
    - 18-bit segment number, 16-bit offset
    - [8-bit major segment, 10-bit minor segment], [6-bit page, 10-bit offset]
    - Both the segment table and segment itself are paged!
  - **Segmentation structure**
    - Segment table is paged
    - major segment number indexes page table for segment table
    - minor segment number is offset within the page of the segment table
    - this gives the page table of the desired segment and the segment length
  - **Paging structure**
    - one-level page table, 1KB pages
  - **TLB**
    - 16 entries; key = 24-bit (seg# & page#); value = frame#
Memory Mapping: Examples (cont’d)

- OS/2 (on Intel 386+): Segmentation with paging

<table>
<thead>
<tr>
<th>Logical Address:</th>
<th>16 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>selector</td>
<td>local/global</td>
<td>offset</td>
</tr>
<tr>
<td>8K segments</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linear Address:</th>
<th>10 bits</th>
<th>12 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>directory</td>
<td>page</td>
<td>offset</td>
</tr>
<tr>
<td>24-bit base</td>
<td>20-bit limit</td>
<td></td>
</tr>
<tr>
<td>3/21/2001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OS/2 (on i386+) Memory Mapping (cont’d)

- Very flexible addressing scheme
  - pure paging
    - All segment registers set up with the same selector
    - Descriptor for this selector has base = 0, limit = 0
    - Offset becomes the address
  - pure segmentation
    - How can this be done?
  - options in between

Next Lecture

- Virtual Memory
  - background: process working sets
  - demand paging
  - page replacement
  - frame allocation
  - thrashing

Reading

- Silberschatz/Galvin: Chapter 9

3/21/2001