Address Translation

Chapter 8 OSPP
Part I: Basics
Main Points

• **Address Translation Concept**
  – How do we convert a virtual address to a physical address?

• **Flexible Address Translation**
  – Base and bound
  – Segmentation
  – Paging
  – Multilevel translation

• **Efficient Address Translation**
  – Translation Lookaside Buffers (TLB)
  – Virtually and physically addressed caches
Address Translation Concept
Address Translation Goals

- Memory protection
- Memory sharing
  - Shared libraries, IPC
- Sparse addresses
  - Multiple regions of dynamic allocation (heaps/stacks)
- Efficiency
  - Memory placement
  - Runtime lookup
  - Compact translation tables
- Portability
Bonus Feature

• What can you do if you can (selectively) gain control whenever a program reads or writes a particular virtual memory location?

• Examples:
  – Copy on write
  – Zero on reference
  – Fill on demand
  – Demand paging
  – Memory mapped files
Virtually Addressed Base and Bounds
Question

• With virtually addressed base and bounds, what is saved/restored on a process context switch?
Virtually Addressed Base and Bounds

- Pros?
- Cons?
Segmentation

• Segment is a contiguous region of *virtual* memory
• Each process has a segment table (in hardware)
  – Entry in table = segment
• Segment can be located anywhere in physical memory
  – Each segment has: start, length, access permission
• Processes can share segments
  – Same start, length, same/different access permissions
  – Great for shared libraries
Segmentation

Processor’s View

Virtual Memory

Processor

Virtual Address

Code

Data

Heap

Stack

Implementation

Segment Table

Segment

Offset

Base

Bound

Access

Read

R/W

R/W

R/W

Physical Memory

Base 3

Stack

Base + Bound 3

Base 0

Code

Base + Bound 0

Base 1

Data

Base + Bound 1

Base 2

Heap

Base + Bound 2

Physical Address

Raise Exception
Question

• With segmentation, what is saved/restored on a process context switch?
UNIX fork and Copy on Write

• UNIX fork
  – Makes a complete copy of a process

• Segments allow a more efficient implementation
  – Copy segment table into child
  – Mark parent and child segments read-only
  – Start child process; return to parent
  – If child or parent writes to a segment (ex: stack, heap)
    • trap into kernel
    • make a copy of the segment and resume
Zero-on-Reference

• Dynamic segments
  – When program uses memory beyond end of stack
  – Segmentation fault into OS kernel
  – Kernel allocates some memory
    • How much?

• Zeros the memory
  – avoid accidentally leaking information!

• Modify segment table

• Resume process
Segmentation

• Pros?
• Cons?
Paged Translation

• Manage memory in fixed size units, or pages
• Finding a free page is easy
  – Bitmap allocation: 0011111100000001100
  – Each bit represents one physical page frame
• Each process has its own page table
  – Stored in physical memory
  – Hardware registers
    • pointer to page table start
    • page table length
Paged Translation (Abstract)
Paged Translation (Implementation)
Process View

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>F</td>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
<td>K</td>
<td>L</td>
</tr>
</tbody>
</table>

Physical Memory

Page Table

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Paging Questions

• With paging, what is saved/restored on a process context switch?
• What if page size is very small?
• What if page size is very large?
Paging and Copy on Write

• Can we share memory between processes?
  – Set entries in both page tables to point to same page frames
  – Need *core map* of page frames to track which processes are pointing to which page frames (e.g., reference count)

• UNIX fork with copy on write
  – Copy page table of parent into child process
  – Mark all pages (in new and old page tables) as read-only
  – Trap into kernel on write (in child or parent)
  – Copy page
  – Mark both as writeable
  – Resume execution
Fill On Demand

• Can I start running a program before its code is in physical memory?
  – Set all page table entries to invalid
  – When a page is referenced for first time, kernel trap
  – Kernel brings page in from disk
  – Resume execution
  – Remaining pages can be transferred in the background while program is running
Data Breakpoints

• Please trace variable A
• Mark page P containing A as read-only
• If P is changed, trap into kernel, and see if A actually changed
Page Table Issue

• 64 bit machines