Discussion 5: Examples of segmentation and paging
Agenda

• Memory access quick recap
• Example of segmentation
• Example of paging
• Address translation overview
“All problems in computer science can be solved by another level of indirection”

David Wheeler
Q1. How many address bits does a system have?

Depends on the CPU architecture
Examples

• X86 architecture – 32 bit
  • Page based 32-bit virtual memory system
  • 4GB ($2^{32} = 4$GB) of virtual memory.

• X86-64 architecture
  • Page based 64-bit virtual memory system

• Actual available physical memory might be quite less (e.g., 1GB RAM).
int main() {
    char arr[3] = {0};
    arr[1]=2;
    arr[2]=3;
    return 0;
}
int main()
{
    char arr[3] = {0};
    arr[1]=2;
    arr[2]=3;
    return 0;
}
Latency examples

<table>
<thead>
<tr>
<th>System Event</th>
<th>Actual Latency</th>
<th>Scaled Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>One CPU cycle</td>
<td>0.4 ns</td>
<td>1 s</td>
</tr>
<tr>
<td>Level 1 cache access</td>
<td>0.9 ns</td>
<td>2 s</td>
</tr>
<tr>
<td>Level 2 cache access</td>
<td>2.8 ns</td>
<td>7 s</td>
</tr>
<tr>
<td>Level 3 cache access</td>
<td>28 ns</td>
<td>1 min</td>
</tr>
<tr>
<td>Main memory access (DDR DIMM)</td>
<td>~100 ns</td>
<td>4 min</td>
</tr>
<tr>
<td>Intel® Optane™ DC persistent memory access</td>
<td>~350 ns</td>
<td>15 min</td>
</tr>
<tr>
<td>Intel® Optane™ DC SSD I/O</td>
<td>&lt;10 µs</td>
<td>7 hrs</td>
</tr>
<tr>
<td>NVMe SSD I/O</td>
<td>~25 µs</td>
<td>17 hrs</td>
</tr>
<tr>
<td>SSD I/O</td>
<td>50–150 µs</td>
<td>1.5–4 days</td>
</tr>
<tr>
<td>Rotational disk I/O</td>
<td>1–10 ms</td>
<td>1–9 months</td>
</tr>
<tr>
<td>Internet call: San Francisco to New York City</td>
<td>65 ms[3]</td>
<td>5 years</td>
</tr>
<tr>
<td>Internet call: San Francisco to Hong Kong</td>
<td>141 ms[3]</td>
<td>11 years</td>
</tr>
</tbody>
</table>

https://www.prowesscorp.com/computer-latency-at-a-human-scale/
Segmentation example

- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
Segmentation example

- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.

- Code, Data, and Stack for a program or System data structures
A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.

- Code, Data, and Stack for a program or System data structures
Segmentation example

- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures
- Different models of segmentation
Segmentation example

- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.
- Code, Data, and Stack for a program or System data structures
- Different models of segmentation
Segmentation example

- A mechanism for dividing the processor’s addressable memory space (called the linear address space) into smaller protected address spaces called segments.

- Code, Data, and Stack for a program or System data structures

- Different models of segmentation
Segmentation example

- Illustrate the memory organization of the x86 logical address translation through a simple example.

  Assume that the hardware translates the logical address ‘0xb00002005’.

  The GDT register value is ‘0x7095’ and base address of the segment involved in the translation of this logical address is 0xc00000.

  Draw a diagram representing the state of the GDT in physical memory and the process of translation.

- Assume: Padding left : 0 whenever applicable
Logical address (0xb00002005)
Logical address *(0xb00002005)*

<table>
<thead>
<tr>
<th>Segment Selector</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 b</td>
<td>0000 0000 0000 0000 0010 0000 0000 0000 0000 0101</td>
</tr>
</tbody>
</table>


GDT lookup

Segment Selector

0 0 0 b
0000 0000 0000 1 0 11

Table Indicator
0 = GDT
1 = LDT
Requested Privilege Level (RPL)

Figure 3-6. Segment Selector
GDT lookup

Segment Selector

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1 0 11</td>
</tr>
</tbody>
</table>

GDTR -> 0x7095

NULL segment
GDT lookup

Segment Selector

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1 0 11</td>
</tr>
</tbody>
</table>

Segment Descriptor

NULL segment

GDTR -> 0x7095
GDT lookup

Segment Selector

<table>
<thead>
<tr>
<th>Segment Selector</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Limit</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1 0 11</td>
</tr>
</tbody>
</table>

(Recall: base address of the segment involved in the translation of this logical address is 0xc00000.)

Base | Limit | Access Control
---|---|---
0x00c00000 |  |  |
**Linear address generation?**

<table>
<thead>
<tr>
<th>Segment Selector</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 b</td>
<td>0 0 0 0 0 2 0 0 0 0 5</td>
</tr>
<tr>
<td>0000 0000 0000 1 0 11</td>
<td>0000 0000 0000 0000 0010 0000 0000 0101</td>
</tr>
</tbody>
</table>

---

### Segment Descriptor

<table>
<thead>
<tr>
<th>Base</th>
<th>Limit</th>
<th>Access Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00c00000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

The linear address is calculated as follows:

- Linear address
- \(= 0x00c00000 \text{ (Base)} + 0x000002005 \text{ (Offset)}\)
- \(= 0x00c02005\)
Linear address translation

Figure 3-5. Logical Address to Linear Address Translation
Q: Where is the GDT?
Q: Where is the GDT?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster?
Q: Where is the GDT?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster?

Yes, with registers to store 6 most recent translations using Segment Registers
Question

• Illustrate the memory organization of the x86, 4K, 32bit page table through a simple example.

Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0. Draw a diagram representing the state of the page table in physical memory and the process of linear address space paging.
Virtual address : 0xc02005

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>c</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>1100</td>
<td>0000</td>
<td>0010</td>
<td>0000</td>
<td>0000</td>
<td>0101</td>
</tr>
</tbody>
</table>
Virtual address : 0xc02005

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>c</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>1100</td>
<td>0000</td>
<td>0010</td>
<td>0000</td>
<td>0000</td>
<td>0101</td>
</tr>
</tbody>
</table>

0000 0000 1100 0000 0010 0000 0000 0101  
Size of a page = 4K (specified already)
Virtual address : 0xc02005

2^Number of bits in offset = Size of page
Virtual address: 0xc02005

2^Number of bits in offset = Size of page

2^Number of bits in offset = 4kB

0000 0000 1100 0000 0010 0000 0000 0101
Virtual address: $0xc02005$

- $2^{\text{Number of bits in offset}} = \text{Size of page}$
- $2^{12} = 4\text{kB}$
- Number of bits in offset = 12
What do the bits represent?

- 0000 0000  1100 0000  0010 0000 0000 0101
What do the bits represent?

- 0000 0000 1100 0000 0010 0000 0000 0101
  - 0000 0000 11
  - 00 0000 0010
  - 0000 0000 10
What do the bits represent?

- 0000 0000 1100 0000 0010 0000 0000 0101
  
  Offset for Page Table Directory (Level 1)
  
  Offset for Page Table (Level 2)
  
  Offset for a byte/word within a page.
0xc0205 =

\[\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array}\]
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xC02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Looking back: Assume that the hardware translates the virtual address '0xc02005' into the physical address '0x4005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are respectively 0x1000 and 0x0.
Q: Where are the Page Tables?
Q: Where are the Page Tables?
In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster?
Q: Where are the Page Tables?

In Main Memory

Q: Main memory access is slow. Can we do anything to make access faster?

Yes, with Translation Lookaside Buffers
Address translation overview:
Putting everything together

Figure 3-1. Segmentation and Paging
“All problems in computer science can be solved by another level of indirection”

David Wheeler

“...except for the problem of too many layers of indirection.”
Questions