VIRTUALIZING MEMORY:
PAGING

Questions answered in this lecture:
Review segmentation and fragmentation
What is paging?
Where are page tables stored?
What are advantages and disadvantages of paging?

ANNOUNCEMENTS

• P1
  • Due officially by Friday, 5pm; unofficially by Saturday 8am
  • Lots of test scripts available; run ta-contest scripts too
  • Lots of office hours through Friday 4:30
• Project 2: Available on Monday
  • Due two weeks later: Monday, Oct 5
  • Can work with project partner in your discussion section (unofficial)
  • Two parts:
    • Linux: Shell – fork() and exec(), file redirection, history
    • Xv6: Scheduler – simplistic M1FQ
    • Two discussion videos again; watch early and often!
• Exam 1: Two weeks, Thu 10/1 7:15 – 9:15
  • Class time that day for review
  • Look at homeworks / simulations for sample questions
• Reading for today:
  • Chapter 18
**REVIEW: MATCH DESCRIPTION**

<table>
<thead>
<tr>
<th>Description</th>
<th>Name of approach (covered previous lecture):</th>
</tr>
</thead>
<tbody>
<tr>
<td>• one process uses RAM at a time</td>
<td>Segmentation</td>
</tr>
<tr>
<td>• rewrite code and addresses before running</td>
<td>Static Relocation</td>
</tr>
<tr>
<td>• add per-process starting location to virt addr to obtain phys addr</td>
<td>Base</td>
</tr>
<tr>
<td>• dynamic approach that verifies address is in valid range</td>
<td>Base+Bounds</td>
</tr>
<tr>
<td>• several base+bound pairs per process</td>
<td>Time Sharing</td>
</tr>
</tbody>
</table>

**REVIEW: SEGMENTATION**

Assume 14-bit virtual addresses, high 2 bits indicate segment

Where does segment table live?

<table>
<thead>
<tr>
<th>Seg</th>
<th>Base</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x4000</td>
<td>0xfff</td>
</tr>
<tr>
<td>1</td>
<td>0x5800</td>
<td>0xfff</td>
</tr>
<tr>
<td>2</td>
<td>0x6800</td>
<td>0x7ff</td>
</tr>
</tbody>
</table>

Segments:
- 0 => code
- 1 => heap
- 2 => stack.
**REVIEW:**

**MEMORY ACCESES**

<table>
<thead>
<tr>
<th>Segment</th>
<th>Base</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x4000</td>
<td>0xffff</td>
</tr>
<tr>
<td>1</td>
<td>0x5800</td>
<td>0xffff</td>
</tr>
<tr>
<td>2</td>
<td>0x6800</td>
<td>0x7ff</td>
</tr>
</tbody>
</table>

%rip: 0x0010

1) Fetch instruction at logical addr 0x0010
   - Physical addr: 0x4010

   Exec, load from logical addr 0x1100
   - Physical addr: 0x5900

2) Fetch instruction at logical addr 0x0013
   - Physical addr: 0x4013

   Exec, no load

3) Fetch instruction at logical addr 0x0019
   - Physical addr: 0x4019

   Exec, store to logical addr 0x1100
   - Physical addr: 0x5900

Total of 5 memory references (3 instruction fetches, 2 movl)

---

**PROBLEM:**

**FRAGMENTATION**

Definition: Free memory that can't be usefully allocated

Why?
- Free memory (hole) is too small and scattered
- Rules for allocating memory prohibit using this free space

Types of fragmentation
- External: Visible to allocator (e.g., OS)
- Internal: Visible to requester (e.g., if must allocate at some granularity)

![Diagram of memory segments](image)
**PAGING**

Goal: Eliminate requirement that address space is contiguous
- Eliminate external fragmentation
- Grow segments as needed

Idea: Divide address spaces and physical memory into fixed-sized pages
- Size: $2^n$, Example: 4KB
- Physical page: page frame

---

**TRANSLATION OF PAGE ADDRESSES**

- How to translate logical address to physical address?
  - High-order bits of address designate page number
  - Low-order bits of address designate offset within page

<table>
<thead>
<tr>
<th>20 bits</th>
<th>12 bits</th>
<th>32 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>page number</td>
<td>page offset</td>
<td>Logical address</td>
</tr>
<tr>
<td>frame number</td>
<td>page offset</td>
<td>Physical address</td>
</tr>
</tbody>
</table>

No addition needed; just append bits correctly...

How does format of address space determine number of pages and size of pages?
**Quiz: Address Format**

Given known page size, how many bits are needed in address to specify offset in page?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
</tr>
</tbody>
</table>

**Quiz: Address Format**

Given number of bits in virtual address and bits for offset, how many bits for virtual page number?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
<th>Virt Addr Bits</th>
<th>High Bits (vpn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
<td>32</td>
<td>12</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
<td>16</td>
<td>5 7</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
<td>32</td>
<td>20</td>
</tr>
</tbody>
</table>

Correct?
**QUIZ: ADDRESS FORMAT**

Given number of bits for vpn, how many virtual pages can there be in an address space?

<table>
<thead>
<tr>
<th>Page Size</th>
<th>Low Bits (offset)</th>
<th>Virt Addr Bits</th>
<th>High Bits (vpn)</th>
<th>Virt Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bytes</td>
<td>4</td>
<td>10</td>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>1 KB</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>1 K</td>
</tr>
<tr>
<td>1 MB</td>
<td>20</td>
<td>32</td>
<td>12</td>
<td>4 K</td>
</tr>
<tr>
<td>512 bytes</td>
<td>9</td>
<td>16</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>4 KB</td>
<td>12</td>
<td>32</td>
<td>20</td>
<td>1 MB</td>
</tr>
</tbody>
</table>

---

**VIRTUAL => PHYSICAL PAGE MAPPING**

Number of bits in virtual address format does not need to equal number of bits in physical address format.

How should OS translate VPN to PPN?

For segmentation, OS used a formula (e.g., phys addr = virt_offset + base_reg)

For paging, OS needs more general mapping mechanism

What data structure is good? Big array: pagetable
THE MAPPING

Page Tables:

Virt Mem

Phys Mem

QUIZ:
FILL IN PAGE TABLE

Page Tables:
WHERE ARE PAGETABLES STORED?

How big is a typical page table?
- assume 32-bit address space
- assume 4 KB pages
- assume 4 byte entries

Final answer: \(2^\left(32 - \log(4KB)\right) \times 4 = 4 \text{ MB}\)
- Page table size = Num entries \times size of each entry
- Num entries = num virtual pages = \(2^{\log(4KB)} = 32 - 12 = 20\)
- Bits for vpn = 32 – number of bits for page offset
- Num entries = \(2^{20} = 1 \text{ MB}\)
- Page table size = Num entries \times 4 bytes = 4 MB

Implication: Store each page table in memory
- Hardware finds page table base with register (e.g., CR3 on x86)

What happens on a context-switch?
- Change contents of page table base register to newly scheduled process
- Save old page table base register in PCB of descheduled process

OTHER PT INFO

What other info is in pagetable entries besides translation?
- valid bit
- protection bits
- present bit (needed later)
- reference bit (needed later)
- dirty bit (needed later)

Pagetable entries are just bits stored in memory
- Agreement between hw and OS about interpretation
MEMORY ACCESSES WITH PAGES

Assume PT is at phys addr 0x5000
Assume PTE’s are 4 bytes
Assume 4KB pages

How many bits for offset? 12

Old: How many mem refs with segmentation?
5 (3 instrs, 2 movl)

Physical Memory Accesses with Paging?
1) Fetch instruction at logical addr 0x0010; vpn?
   • Access page table to get ppn for vpn 0
   • Mem ref 1: 0x5000
   • Learn vpn 0 is at pnn 2
   • Fetch instruction at 0x2010 (Mem ref 2)
2) Exec, load from logical addr 0x1100; vpn?
   • Access page table to get pnn for vpn 1
   • Mem ref 3: 0x5004
   • Learn vpn 1 is at pnn 0
   • Movl from 0x0100 into reg (Mem ref 4)

Pagetable is slow!!! Doubles memory references

ADVANTAGES OF PAGING

No external fragmentation
• Any page can be placed in any frame in physical memory

Fast to allocate and free
• Alloc: No searching for suitable free space
• Free: Doesn’t have to coalesce with adjacent free space
• Just use bitmap to show free/allocated page frames

Simple to swap-out portions of memory to disk (later lecture)
• Page size matches disk block size
• Can run process when some pages are on disk
• Add “present” bit to PTE
DISADVANTAGES OF PAGING

Internal fragmentation: Page size may not match size needed by process
  • Wasted memory grows with larger pages
  • Tension?

Additional memory reference to page table --> Very inefficient
  • Page table must be stored in memory
  • MMU stores only base address of page table
  • Solution: Add TLBs (future lecture)

Storage for page tables may be substantial
  • Simple page table: Requires PTE for all pages in address space
    • Entry needed even if page not allocated
  • Problematic with dynamic stack and heap within address space
  • Page tables must be allocated contiguously in memory
  • Solution: Combine paging and segmentation (future lecture)

---

HOMEWORK EXERCISES

- Look at relocation.py
  - Base+bounds dynamic relocation

- Look at page-linear-translate.py
  - Basic page tables