Memory Management: Day 1

SWE3015

Sung-hun Kim
Memory Management

• Goals
  – To provide a convenient abstraction for programming
  – To allocate scarce memory resources among competing processes to maximize performance with minimal overhead
  – To provide isolation between processes
Background

• Modern computer systems
  – Modern computer architecture (CA)s conduct on shared memory architecture
    • Multi-core (UMA), NUMA, etc...
  – OSes should support multi-programming environments
    • Applications should share restricted one or several memory unit(s)
    • How do we support multi-program environment?

• Solutions
  – Virtual memory
  – Demand paging
Virtual Memory Overview

Applications always see its own virtual memory

OSes must provide:
1) Abstraction
2) Protection
3) Sharing

HW support:
1) MMU
2) MPU

1) Shared resource
2) Restricted size

On demand

SWE3015: Operating System Project
Memory Management Unit

- Translating virtual address to physical address
  - Dividing physical memory and allocating to virtual address
  - Segmented MMU
  - Paged MMU

CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer
**Segmentation**

- Variable allocated
  - Must use non-trivial placement
  - External fragmentation
  - No longer used in mainstream

Local Descriptor Table (LDT)

<table>
<thead>
<tr>
<th>Linear base address (BASE)</th>
<th>Segment size (LIMIT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x21430</td>
<td>0x0C000</td>
</tr>
<tr>
<td>0x0CEF0</td>
<td>0xA300</td>
</tr>
<tr>
<td>0x28C00</td>
<td>0xFC00</td>
</tr>
</tbody>
</table>

Main memory

- Segment 3
  - Segment selector 0x000F
- Segment 2
  - Segment selector: 0x0027
- Segment 1
  - Segment selector 0x0017
• Coder’s first calamity: segmentation fault

```
int main()
{
    unsigned long *x = 0;
    *x = 1234;
}
```

• Linux maintains segmentation to protect regions
  – LOGICALLY!
  – It can be provided also by paging
Linux provides segmentation as unit of virtual memory area (VMA)

- VMA includes address range and access privileges
- Other details will be reviewed later

1) int *p;
   p = 0x0012;
   *p = 30;

2) int *p;
   p = 0x0044;
   *p = 30;
• Memory is divided as fixed-size page
  – Virtual memory page is mapped to physical memory page
    • Default page size is 4KB (can be enlarged by system configuration)
    • Page table maintains the mapping
• Paged address translation architecture
• **Page table entries (PTEs)**

<table>
<thead>
<tr>
<th>V</th>
<th>R</th>
<th>M</th>
<th>Prot</th>
<th>PFN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>

- **Valid bit (V)** says whether or not the PTE can be used.
  - It is checked each time a virtual address is used.
- **Reference bit (R)** says whether the page has been accessed.
  - It is set when a read or write to the page occurs.
- **Modify bit (M)** says whether or not the page is dirty.
  - It is set when a write to the page occurs.
- **Protection bits (Prot)** control which operations are allowed on the page.
  - Read, Write, Execute, etc.
- **Page frame number (PFN)** determines physical page.
• x86 pte
• x86 pte
• **Space overhead**
  - In 32bit machine w/ 4KB page, 1,048,576 entries needed
    • Each process has 4 GiB of address space
    • $32 \times 1,048,576 = 4$ MiB (big overhead)
Multi-level Page Tables

- Reducing average page table size
  - Don’t have to allocate empty page tables
  - Fitting each page table in one page
Problem w/ Multi-level Page Table

- Time overhead
  - Page walk on multi-level (n) page table (PT)
    - At least, n+1 memory access needed for one memory access
    - Ex) 2-level PT: master PT(1) + second PT(1) + actual access (1)
  - A solution: translation lookaside buffer (TLB)
    - Hardware based solution
    - Simply cache virtual-to-physical mapping in buffer
    - Leverage locality
Translation Lookaside Buffer

- A cache to improve address translation speed
  - Managed by hardware MMU
  - Querying TLBs before a page walk
Translation Lookaside Buffer

- A cache to improve address translation speed
  - Managed by hardware MMU
  - Querying TLBs before the page walk