IA32/Linux Virtual Memory Architecture
Basic Execution Environment

Application Programming Registers

<table>
<thead>
<tr>
<th>General-purpose registers</th>
<th>Segment registers</th>
<th>Control registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>AH, AL</td>
<td>CS</td>
</tr>
<tr>
<td>EBX</td>
<td>BH, BL</td>
<td>DS</td>
</tr>
<tr>
<td>ECX</td>
<td>CH, CL</td>
<td>SS</td>
</tr>
<tr>
<td>EDX</td>
<td>DH, DL</td>
<td>ES</td>
</tr>
<tr>
<td>EBP</td>
<td>BP</td>
<td>FS</td>
</tr>
<tr>
<td>ESI</td>
<td>SI</td>
<td>GS</td>
</tr>
<tr>
<td>EDI</td>
<td>DI</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td>SP</td>
<td></td>
</tr>
</tbody>
</table>

System Table Registers

<table>
<thead>
<tr>
<th>System Segment Registers</th>
<th>System Table Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDTR</td>
<td>linear base address</td>
</tr>
<tr>
<td>IDTR</td>
<td>linear base address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Segment Registers</th>
<th>System Table Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TR</td>
<td>seg. selector</td>
</tr>
<tr>
<td>LDTR</td>
<td>seg. selector</td>
</tr>
</tbody>
</table>
IA32 VM Architecture (1)

- **Segmented memory model**
  - Memory appears to a program as a group of independent address space called segments.
  - A program must issue a logical address, which consists of a segment selector and an offset.
  - Up to 16,383 segments of different sizes and types
    - Each segment can be as large as $2^{32}$ bytes.
  - No way to disable segmentation.
  - The use of paging is optional.
IA32 VM Architecture (2)

- **Logical address (far pointer)**
  - User’s view, segmented

  - Segment selector (16bits)
  - Offset (32bits)

- **Linear address**
  - 32-bit, flat

- **Physical address**
  - 32-bit, flat
  - Pentium Pro and later processors support an extension of the physical address space to $2^{36}$ bytes.
  - Invoked with the physical address extension (PAE) flag located in CR4 register.
IA32 VM Architecture (3)

Logical Address (or Far Pointer)

Segment Selector

Offset

Linear Address Space

Segment

Lin. Addr.

Global Descriptor Table (GDT)

Segment Descriptor

Segment Base Address

Segmentation

Paging

Linear Address

Dir Table Offset

Page Directory

Entry

Page Table

Entry

Page

Phy. Addr.

Physical Address Space
Segmentation (1)

- Basic flat model
  - The OS and applications have access to a continuous, unsegmented address space.
  - All segment descriptors have the same base address value of 0 and the same segment limit of 4GB.
Segmentation (2)

- **Protected flat model**
  - Segment limits are set to include only the range of addresses for which physical memory actually exists.
  - May have multiple segments, but all overlay each other and start at address 0 in the linear address space.
**Segmentation (3)**

- **Multisegment model**
  - Each program (or task) is given its own table of segment descriptors and its own segments.
  - The segments can be completely private to their assigned programs or shared among programs.
Segmentation (4)

- **Segment registers**
  - Hold 16-bit segment selectors.
    - A segment selector is a special pointer that identifies a segment in memory
    - To access a particular segment, the segment selector for that segment must be present in the appropriate segment register.
  - Use of segment registers
    - CS: for code segment
    - DS, ES, FS, and GS: for data segments (up to 4 segments simultaneously)
    - SS: for stack segment
  - FS and GS registers were introduced with the 80386 family of processors.
Segmentation (5)

- Logical to linear address
  - Examine the segment descriptor in GDT or LDT to check the access rights and the offset is within the limits.
  - Adds the segment base address from the segment descriptor to the offset to form a linear address.
Segmentation (6)

- Segment selector

<table>
<thead>
<tr>
<th>15</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>T</td>
<td>I</td>
<td>RPL</td>
<td></td>
</tr>
</tbody>
</table>

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

- Segment registers

<table>
<thead>
<tr>
<th>Visible Part</th>
<th>Hidden Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Segment Selector</td>
<td>Base Address, Limit, Access Information</td>
</tr>
<tr>
<td>CS</td>
<td>SS</td>
</tr>
<tr>
<td>DS</td>
<td>ES</td>
</tr>
<tr>
<td>FS</td>
<td>GS</td>
</tr>
</tbody>
</table>
Segmentation (7)

- **Segment descriptor tables**
  - Each system must have one GDT (Global Descriptor Table), which may be used for all programs and tasks.
  - Optionally, one or more LDTs (Local Descriptor Tables) can be defined in a system segment.
  - GDT is not a segment, but a data structure in the linear address space pointed to by the GDTR register.
  - GDT must contain a segment descriptor for the LDT segment.
  - The first descriptor in GDT is not used.
  - The LDTR register caches the segment descriptor of the current LDT segment.
Segmentation (8)

- Global and local descriptor tables
Segmentation (9)

- Segment descriptor

<table>
<thead>
<tr>
<th>Field</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base 31:24</td>
<td>Segment base address</td>
</tr>
<tr>
<td>G D/B AVL</td>
<td>Segment type (G = granular, D/B = 32/16 bit default size, AVL = available)</td>
</tr>
<tr>
<td>P DPL S</td>
<td>Permissions and privilege level</td>
</tr>
<tr>
<td>Type</td>
<td>Segment type (0 = system, 1 = code or data)</td>
</tr>
<tr>
<td>Base 23:16</td>
<td>Segment limit</td>
</tr>
<tr>
<td>L</td>
<td>64-bit code segment (IA-32e mode only)</td>
</tr>
<tr>
<td>AVL</td>
<td>Available for use by system software</td>
</tr>
<tr>
<td>BASE</td>
<td>Segment base address</td>
</tr>
<tr>
<td>D/B</td>
<td>Default operation size (0 = 16-bit segment; 1 = 32-bit segment)</td>
</tr>
<tr>
<td>DPL</td>
<td>Descriptor privilege level</td>
</tr>
<tr>
<td>G</td>
<td>Granularity</td>
</tr>
<tr>
<td>LIMIT</td>
<td>Segment limit</td>
</tr>
<tr>
<td>P</td>
<td>Segment present</td>
</tr>
<tr>
<td>S</td>
<td>Descriptor type (0 = system; 1 = code or data)</td>
</tr>
<tr>
<td>TYPE</td>
<td>Segment type</td>
</tr>
</tbody>
</table>
### Paging (1)

#### Paging support in IA-32
- Optional: enabled by PG flag of CR0 register
- Default page size: 4KB
  - PSE (page size extension) flag of CR4 enables 4MB page size
    (From Pentium)

#### 36-bit physical addressing
- Pentium Pro and later processors support an extension of the physical address space to $2^{36}$ bytes.
  - Enabled by PAE (physical address extension) flag of CR4
  - With PAE enabled, 2MB page size is supported
- Pentium III introduced PSE-36 mechanism
  - Available when PSE-36 CPUID feature flag is set
  - Map up to 1024 4MB pages into 64GB physical address space
Paging (2)

- Linear to physical address (4KB)
  - The *physical* address of the current page directory is stored in the CR3 register (a.k.a. page directory base register or PDBR).

*32 bits aligned onto a 4-KByte boundary.*
Paging (3)

- Page tables and directories
  - Page directory
    - An array of 32-bit page-directory entries (PDEs) contained in a 4KB page (1024 PDEs/page).
  - Page table
    - An array of 32-bit page-table entries (PTEs) contained in a 4KB page (1024 PTEs/page).
    - Page tables are not used for 2MB or 4MB pages.
  - Page
    - Supports page sizes of 4KB, 2MB, and 4MB.
  - Page-directory-pointer table
    - An array of four 64-bit entries pointing to a page directory.
    - Only used when the physical address extension is enabled.
Paging (4)

- **Linear to physical address (4MB, PSE enabled)**
  - Both 4MB pages and page tables for 4KB pages can be accessed from the same page directory
  - Place OS kernel in 4MB pages to reduce TLB misses

![Diagram of Linear Addressing and Page Mapping]

*32 bits aligned onto a 4-KByte boundary.*
Paging (5)

- Linear to physical address (4KB, PAE enabled)

![Diagram showing paging structure with linear and physical addresses]

4 PDPTPE * 512 PDE * 512 PTE = 2^{20} Pages

*32 bits aligned onto a 32-byte boundary
## Paging (6)

### Page directory entry (PDE)

<table>
<thead>
<tr>
<th>Page-Directory Entry (4-KByte Page Table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 12 11 9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Page-Table Base Address</td>
</tr>
</tbody>
</table>

- Available for system programmer’s use
- Global page (Ignored)
- Page size (0 indicates 4 KBytes)
- Available
- Accessed
- Cache disabled
- Write-through
- User/Supervisor
- Read/Write
- Present
### Paging (7)

- **Page table entry (PTE)**

**Page-Table Entry (4-KByte Page)**

<table>
<thead>
<tr>
<th>31</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Available for system programmer’s use
- Global Page
- Page Table Attribute Index
- Dirty
- Accessed
- Cache Disabled
- Write-Through
- User/Supervisor
- Read/Write
- Present

---

CSE3008: Operating Systems | Fall 2012 | Jin-Soo Kim (jinsookim@skku.edu)
Paging (8)

- **TLBs**
  - The P6 family and Pentium processors have separate TLBs for the data and instruction. (DTLB & ITLB)
  - Separate TLBs for 4KB and 4MB page sizes
  - All TLBs are automatically invalidated if the PDBR register is loaded.
    - by explicit MOV instruction
    - implicitly by executing a task switch
  - A specific page-table entry in the TLB can be invalidated using INVLPG instruction.
  - The page global enable (PGE) flag in CR4 and the global (G) flag of a PDE or PTE can be used to prevent frequently used pages from being automatically invalidated.
IA32 References

- **For more information, see**
    - Volume 1: Basic Architecture
    - Volume 2: Instruction Set Reference
    - Volume 3: System Programming Guide
  - Available at Intel’s web site:
Linux VM Architecture (1)

Virtual memory

0x00000000

3GB

PAGE_OFFSET = 0xC0000000

0x00000000

1GB

0xFFFF0000

1GB

Physical memory

Available Page Frames

Kernel code
Kernel data
Page tables
Freelists, etc.
**Linux VM Architecture (2)**

- **Segmentation: Minimal approach**
  - For better portability across machines

<table>
<thead>
<tr>
<th>Segment selector</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>__KERNEL_CS</td>
<td>0x10</td>
</tr>
<tr>
<td>__KERNEL_DS</td>
<td>0x18</td>
</tr>
<tr>
<td>__USER_CS</td>
<td>0x23</td>
</tr>
<tr>
<td>__USER_DS</td>
<td>0x2b</td>
</tr>
</tbody>
</table>

**GDT**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NULL</td>
</tr>
<tr>
<td>0x08</td>
<td>(not used)</td>
</tr>
<tr>
<td>0x10</td>
<td>Kernel text from 0 (4GB)</td>
</tr>
<tr>
<td>0x18</td>
<td>Kernel data from 0 (4GB)</td>
</tr>
<tr>
<td>0x20</td>
<td>User text from 0 (4GB)</td>
</tr>
<tr>
<td>0x28</td>
<td>User data from 0 (4GB)</td>
</tr>
<tr>
<td>0xa0</td>
<td>(not used)</td>
</tr>
<tr>
<td>0xa0</td>
<td>(not used)</td>
</tr>
</tbody>
</table>

- Used for APM (4 entries)
- Used for PNPBIOS (8 entries)
- 4 entries per CPU For TSS’s & LDT’s
Paging: Three-level address translation

- In i386, the size of Page Middle Directory (PMD) is 1, if the physical address extension (PAE) flag is disabled.
Virtual memory areas (VMA)

- Nonoverlapping regions, each region representing a continuous, page-aligned subset of the virtual address space.
- Described by a single `vm_area_struct`
- VMAs are linked into a balanced binary tree to allow fast lookup of the region corresponding to any virtual address.
  - VMAs form a red-black tree.
Linux VM Architecture (5)

- task_struct
  - mm
  - mm_struct
    - map_count
    - pgd
    - mmap
    - mm_rb
  - page directory
  - PFN

- mm_struct
  - mm

- vm_area_struct
  - vm_start
  - vm_end
  - vm_mm
  - vm_rb
  - vm_ops
  - vm_next

Virtual address space

VM Area 1

VM Area 2
Linux VM Architecture (6)

- VMA example

```
[root@oz0 jinsoo]# cat /proc/1/maps
08048000-0804e000 r-xp 00000000 03:03 716858 /sbin/init
0804e000-0804f000 rw-p 00006000 03:03 716858 /sbin/init
0804f000-08053000 rwxp 00000000 00:00 0
40000000-40013000 r-xp 00000000 03:03 244332 /lib/ld-2.2.5.so
40013000-40014000 rw-p 00013000 03:03 244332 /lib/ld-2.2.5.so
40031000-40032000 rw-p 00000000 00:00 0
42000000-4212c000 r-xp 00000000 03:03 915244 /lib/i686/libc-2.2.5.so
4212c000-42131000 rw-p 0012c000 03:03 915244 /lib/i686/libc-2.2.5.so
42131000-42135000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp 00000000 00:00 0
[root@oz0 jinsoo]#
```