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, EBX, ECX, EDX, EBP, ESI, EDI, ESP</td>
<td>CS, DS, SS, ES, FS, GS</td>
<td>CR0, CR1, CR2, CR3, CR4</td>
</tr>
<tr>
<td>[31]</td>
<td>[15]</td>
<td>[31]</td>
</tr>
<tr>
<td>AH, BH, CH, DH, BL, DH, DL, BP, SI, DI, SP</td>
<td>seg. selector</td>
<td></td>
</tr>
</tbody>
</table>

## System Table Registers

<table>
<thead>
<tr>
<th>System Segment Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>system segment registers</td>
</tr>
<tr>
<td>TR, LDTR, GDTR, IDTR</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

- 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.
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**

  - Index
  - 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></td>
</tr>
<tr>
<td>SS</td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td></td>
</tr>
<tr>
<td>FS</td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td></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>Base 31:24</th>
<th>D</th>
<th>L</th>
<th>AVL</th>
<th>Seg. Limit 19:16</th>
<th>P</th>
<th>DPL</th>
<th>S</th>
<th>Type</th>
<th>Base 23:16</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24 23 22 21 20 19</td>
<td>16 15 14 13 12 11</td>
<td>8 7</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>16 15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base Address 15:00</td>
<td>Segment Limit 15:00</td>
</tr>
</tbody>
</table>

L — 64-bit code segment (IA-32e mode only)
AVL — Available for use by system software
BASE — Segment base address
D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
DPL — Descriptor privilege level
G — Granularity
LIMIT — Segment Limit
P — Segment present
S — Descriptor type (0 = system; 1 = code or data)
TYPE — Segment type
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).

![Diagram of paging structure]

*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

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

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

```
<table>
<thead>
<tr>
<th>Linear Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 21 20 12 11 0</td>
</tr>
</tbody>
</table>

Directory Pointer

- Directory
- Table
- Offset

Page Directory

- Directory Entry

Page-Directory-Pointer Table

- Dir. Pointer Entry

CR3 (PDPTR)

*32 bits aligned onto a 32-byte boundary

4 PDPTE * 512 PDE * 512 PTE = 2^{20} Pages
Paging (6)

- Page directory entry (PDE)

<table>
<thead>
<tr>
<th>Page-Directory Entry (4-KByte Page Table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</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
### Page table entry (PTE)

<table>
<thead>
<tr>
<th>Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Available for system programmer’s use</td>
</tr>
<tr>
<td>30-25</td>
<td>Global Page</td>
</tr>
<tr>
<td>24-19</td>
<td>Page Table Attribute Index</td>
</tr>
<tr>
<td>18</td>
<td>Dirty</td>
</tr>
<tr>
<td>17-12</td>
<td>Accessed</td>
</tr>
<tr>
<td>11-6</td>
<td>Cache Disabled</td>
</tr>
<tr>
<td>5</td>
<td>Write-Through</td>
</tr>
<tr>
<td>4-0</td>
<td>User/Supervisor</td>
</tr>
<tr>
<td></td>
<td>Present</td>
</tr>
</tbody>
</table>

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

```
<table>
<thead>
<tr>
<th></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></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>
<tr>
<td>31</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>
```

- **Avail**: Available
- **G**: Global
- **P**: Present
- **A**: Accessed
- **T**: Dirty
- **D**: Cache Disabled
- **P**: Write-Through
- **C**: User/Supervisor
- **W**: Read/Write
- **T**: Present
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

PAGE_OFFSET = 0xC0000000

3GB

0x00000000

0x3FFFFFFF

Physical memory

1GB

1GB

Available Page Frames

Kernel code

Kernel data

Page tables

Freelists, etc.
### Linux VM Architecture (2)

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

#### GDT

<table>
<thead>
<tr>
<th>Offset</th>
<th>Segment Type</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>0x2c</td>
<td>(not used)</td>
</tr>
<tr>
<td>0x30</td>
<td>(not used)</td>
</tr>
</tbody>
</table>

- Used for APM (4 entries)
- Used for PNPBIOS (8 entries)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Segment Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xa0</td>
<td>__USER_CS</td>
</tr>
<tr>
<td>0x10</td>
<td>__KERNEL_CS</td>
</tr>
<tr>
<td>0x18</td>
<td>__KERNEL_DS</td>
</tr>
<tr>
<td>0x23</td>
<td>__USER_CS</td>
</tr>
<tr>
<td>0x2b</td>
<td>__USER_DS</td>
</tr>
</tbody>
</table>

4 entries per CPU
- For TSS’s & LDT’s
Linux VM Architecture (3)

- 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.
Linux VM Architecture (4)

- 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

- **mm_struct**
  - mm
  - map_count
  - pgd
  - mmap
  - mm_rb

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

- **Virtual address space**
  - VM Area 1
  - VM Area 2

- **PFN**
  - page directory
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
```

VMA  | permission | offset  | device | i-node | mapped file