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</td>
<td>AL</td>
</tr>
<tr>
<td>EBX</td>
<td>BH</td>
<td>BL</td>
</tr>
<tr>
<td>ECX</td>
<td>CH</td>
<td>CL</td>
</tr>
<tr>
<td>EDX</td>
<td>DH</td>
<td>DL</td>
</tr>
<tr>
<td>EBP</td>
<td>BP</td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td>SI</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td>DI</td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td>SP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Table Registers</th>
<th>System Segment 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>

SSE3044: Operating Systems | Fall 2014 | Jin-Soo Kim (jinsookim@skku.edu)
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)
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.

![Diagram showing segment registers, segment descriptors, and linear address space.](image)
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

![Segment Selector Diagram]

- 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>Bit #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Base 31:24</td>
</tr>
<tr>
<td>24-22</td>
<td>G (Global)</td>
</tr>
<tr>
<td>21-20</td>
<td>D/B (Default/Big)</td>
</tr>
<tr>
<td>19</td>
<td>AVL (Available)</td>
</tr>
<tr>
<td>16-15</td>
<td>Seg. Limit 19:16</td>
</tr>
<tr>
<td>14-13</td>
<td>P (Present)</td>
</tr>
<tr>
<td>12-11</td>
<td>DPL (Descriptor privilege level)</td>
</tr>
<tr>
<td>8-7</td>
<td>S (Segment type)</td>
</tr>
<tr>
<td>0</td>
<td>Type</td>
</tr>
</tbody>
</table>

- Base Address 15:00
- Segment Limit 15:00

| 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 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](image-url)

*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)

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

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

- **IA-32e paging mode in Intel64**
  - 48-bit virtual address $\rightarrow$ 52-bit physical address (4KB)
### Paging (7)

- **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>12 11</td>
</tr>
<tr>
<td>9 8 7 6 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Page-Table Base Address</td>
</tr>
<tr>
<td>Avail G P S A V L A P C D P W T U R S W P</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 (8)

## Page table entry (PTE)

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

- 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

• IA-32 Intel Architecture Software Developer’s Manual
  – 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

0xC0000000

0xFFFFFFFF

Physical memory

0x00000000

1GB

0x00000000

0xFFFFFFFF

0xFFFFFFFF

Available Page Frames

0xC0000000

Kernel code

Kernel data

Page tables

Freelists, etc.

PAGE_OFFSET = 0xC0000000

0x00000000

0x3FFFFFFF
Virtual Address Space

- BIOS
  - kernel text
  - kernel data
- kmalloc etc.
- vmalloc
- pkmap
- fixmap

Physical Address Space

- BIOS
  - kernel text
  - kernel data
- low memory
  - high_memory
    - VMALLOC_START
    - PKMAP_BASE
    - FIXADDR_START
  - ~ 890MB
- high_memory
  - 1GB
  - 2GB

Use large pages where possible

Kernel Space

- PAGE_OFFSET

Linear Mapping

Non-contiguous Mapping

3GB

4GB

0

Low Memory

High Memory
### Linux VM Architecture (3)

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

#### GDT

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NULL (not used)</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></td>
<td>(not used)</td>
</tr>
<tr>
<td></td>
<td>(not used)</td>
</tr>
<tr>
<td></td>
<td>Used for APM (4 entries)</td>
</tr>
<tr>
<td></td>
<td>Used for PNPBIOS (8 entries)</td>
</tr>
<tr>
<td>0xa0</td>
<td>4 entries per CPU For TSS’s &amp; LDT’s</td>
</tr>
</tbody>
</table>

#### Segment selectors

<table>
<thead>
<tr>
<th>Selector</th>
<th>Offset</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>

Table Indicator:
- 0 = GDT
- 1 = LDT

Requested Privilege Level (RPL)
Paging: Four-level address translation

- The size of PUD and 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 (6)

- task_struct
  - mm
    - 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_start
    - vm_end
    - vm_mm
    - vm_rb
    - vm_ops
    - vm_next
  - VM Area 2
## Linux VM Architecture (7)

- **VMA example**

<table>
<thead>
<tr>
<th>VMA</th>
<th>permission</th>
<th>offset</th>
<th>device</th>
<th>i-node</th>
<th>mapped file</th>
</tr>
</thead>
<tbody>
<tr>
<td>08048000-0804e000</td>
<td>r-xp</td>
<td>00000000</td>
<td>03:03</td>
<td>716858</td>
<td>/sbin/init</td>
</tr>
<tr>
<td>0804e000-0804f000</td>
<td>rw-p</td>
<td>00006000</td>
<td>03:03</td>
<td>716858</td>
<td>/sbin/init</td>
</tr>
<tr>
<td>0804f000-08053000</td>
<td>rwxp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>/lib/ld-2.2.5.so</td>
</tr>
<tr>
<td>40000000-40130000</td>
<td>r-xp</td>
<td>00000000</td>
<td>03:03</td>
<td>244332</td>
<td>/lib/ld-2.2.5.so</td>
</tr>
<tr>
<td>40013000-40014000</td>
<td>rw-p</td>
<td>00013000</td>
<td>03:03</td>
<td>244332</td>
<td>/lib/ld-2.2.5.so</td>
</tr>
<tr>
<td>40031000-40032000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>/lib/i686/libc-2.2.5.so</td>
</tr>
<tr>
<td>42000000-4212c000</td>
<td>r-xp</td>
<td>00000000</td>
<td>03:03</td>
<td>915244</td>
<td>/lib/i686/libc-2.2.5.so</td>
</tr>
<tr>
<td>4212c000-42131000</td>
<td>rw-p</td>
<td>0012c000</td>
<td>03:03</td>
<td>915244</td>
<td>/lib/i686/libc-2.2.5.so</td>
</tr>
<tr>
<td>42131000-42135000</td>
<td>rw-p</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>/lib/i686/libc-2.2.5.so</td>
</tr>
<tr>
<td>bffff000-c0000000</td>
<td>rwxp</td>
<td>00000000</td>
<td>00:00</td>
<td>0</td>
<td>/lib/i686/libc-2.2.5.so</td>
</tr>
</tbody>
</table>
Linux VM Architecture (8)

- Linux page replacement
  - LRU-based (approximation)
  - Second chance
    - Similar to FIFO
    - But give second chance when a tail page is referenced
    - Not all architectures provide recency information
  - 2Q-based LRU
    - Filter data which is used only once
  - Separate LRU lists for file and anonymous pages
  - Anonymous pages are not scanned when there is no swap
Linux VM Architecture (9)

- Page replacement: File pages

![Diagram of page replacement algorithm]

- New page
  - INACTIVE LIST
    - Not freeable
    - Referenced once or never
  - ACTIVE LIST
    - 2x Referenced
    - Not referenced
    - Referenced
  - Evicted

\[|\text{Active list}| / |\text{Inactive list}| = 1\]
Linux VM Architecture (10)

- Page replacement: Anonymous pages

\[ \frac{|\text{Active list}|}{|\text{Inactive list}|} = 1, \quad \text{if mem < 1GB} \]
\[ = \sqrt{10 \times \text{gb}}, \quad \text{otherwise} \]