Memory management in i386

Addresses:

- Logical addresses: part of machine language instruction that specifies address of operand / instruction
  - 48-bit 2-tuple (\langle\text{segment selector}(16), \text{offset}(32)\rangle)
- Linear addresses: 32-bit unsigned integer
- Physical addresses: bit-pattern put on address lines on the bus
Segmentation in i386

- Logical address space is divided into 2 partitions (up to 8K private segments, up to 8K shared segments)

- Segment descriptor tables:
  - Segmentation information stored in memory in Local Descriptor Table (LDT), Global Descriptor Table (GDT)
  - Base address of descriptor tables contained in `gdtr`, `ldtr` registers
Segment selector

- 13-bit index to identify a segment descriptor entry
- 1-bit table indicator to specify the table to be used (LDT/GDT)
- 2-bit privilege level to hold the current privilege level of CPU
Segment descriptor tables

- **base**(32) – linear address of the first byte of segment
- **limit**(20) – segment length (upto 4 GB)
- **granularity**(1) – whether segment size is expressed in bytes or in 4KB units
- **descriptor privilege level**(2) – minimal CPU privilege level required for accessing the segment
- **segment-present flag**(1) – whether segment is currently in main memory (always 1 in Linux)
- **type**(4) – code, data/stack, etc.
Segmentation registers

- 6 segment registers to hold segment selectors
  - cs, ds, ss: code, data, stack segment
  - es, fs, gs: general purpose (may point to arbitrary data segments)
  - process can address up to 6 segments at one time

- 6 non-programmable 8-byte registers to hold corresponding descriptors from LDT/GDT
  - speeds up translation of logical addresses into linear addresses
  - when a Segment Selector is loaded into segmentation register, corresponding Segment Descriptor is loaded from memory into the matching 8-byte register
Linux (+ other variants of UNIX) “ignores” segmentation
(∵ RISC architectures have limited support for segmentation)
⇒ processes see a flat address space

All segment descriptors are stored in GDT (LDTs not used)

<table>
<thead>
<tr>
<th>Segment</th>
<th>Base</th>
<th>Limit</th>
<th>DPL</th>
<th>Perm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel code</td>
<td>0x00000000</td>
<td>0xffffffff</td>
<td>0 (kernel mode)</td>
<td>r-x</td>
</tr>
<tr>
<td>Kernel data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User code</td>
<td></td>
<td></td>
<td>3 (user mode)</td>
<td>r-x</td>
</tr>
<tr>
<td>User data</td>
<td></td>
<td></td>
<td></td>
<td>r-w</td>
</tr>
</tbody>
</table>

Operating Systems: Memory Management in Linux – p. 7/10
Paging in i386

- Page size: 4K
- 2-level paging is used

Offset: \( p_1 (10) \) \( p_2 (10) \) \( d (12) \)

- \( p_1 \): Page Directory pointer
- \( p_2 \): Page Table pointer

Physical address of page directory is stored in \( \text{cr3} \)
- \( \text{cr3} \) is changed on a context switch
- change of \( \text{cr3} \) automatically flushes TLB

Page table pages:
- allocated RAM only when required
- can be swapped to disk

**NOTE:** Paging is enabled by setting the PG flag of \( \text{cr0} \)

\( (\text{PG} = 0 \Rightarrow \text{linear addresses are interpreted as physical addresses}) \)
Page directory/page table entries:

- **Present flag**: indicates whether page/PT is contained in memory / disk
  - remaining bits can be used to specify location on disk
- **Frame address**: 20 most significant bits of a page frame physical address
- **Reference bit**: set whenever corresponding page frame is addressed*
- **Dirty bit** (only for PT entries): set whenever page frame is written*
- **Permissions**: access rights (read/write or read)
- **User/Supervisor flag**: privilege level required to access the page or Page Table

(*): has to be cleared by OS (not by hardware)
Process address space

- Linear address space of a process is divided into two parts:
  - 3GB (0x00000000 to 0xbfffffff)
    - can be addressed in user / kernel mode
    - covered by first 768 entries of PGD
  - 1GB (0xc0000000 to 0xffffffff)
    - can be addressed only in kernel mode
    - covered by remaining entries of PGD
    - identical to corresponding entries of the kernel master PGD (same for all processes)
- simplifies user → kernel transitions