Segmentation Review

Typically, the kernel uses only a few GDT entries: some for the hardware's data structures (e.g., Task State Segment), and others for the OS to use. For example, the kernel may use only two segment descriptors, one for the user's address space (USEGMENT), and another for the kernel's address space (KSEGMENT). The contents of the KSEGMENT descriptor remain constant throughout the execution of the system, as the kernel does not change its own location. The contents of the USEGMENT descriptor are overwritten on every context switch, and loaded with the base and limit values of the current process.

Caching of Segment Descriptors

A segment descriptor is cached on the CPU (on chip) each time a segment register is loaded. Thus the logic to translate a virtual address (VA) through segmentation does not require a memory access.

Address translation and sharing using page tables

Why do we care about x86 address translation?

- It can simplify s/w structure: addresses in one process not constrained by what other processes might be running.
- It can implement tricks like demand paging and copy-on-write.
- It can isolate programs to contain bugs or increase security.
- It can provide efficient sharing between processes.

Why aren't protected-mode segments enough?

- Why did the 386 add translation using page tables as well?
- Isn't it enough to give each process its own segments?
- Programming model, fragmentation
- In practice, segments are little-used

Translation using page tables (on x86):

- segmentation hardware first computes the linear address
- in practice, most segments (e.g. in `pintos`, Linux) have base 0 and max limit, making the segmentation step a no-op.
- paging hardware then maps linear address (la) to physical address (pa)
- (we will often interchange "linear" and "virtual")
- when paging is enabled, every instruction that accesses memory is subject to translation by paging

- paging idea: break up memory into 4096-byte chunks called pages
- independently control mapping for each page of linear address space
- compare with segmentation (single base + limit): many more degrees of freedom

- 4096-byte pages means there are 2^20 = 1,048,576 pages in 2^32 bytes
- conceptual model: array of 2^20 entries, called a page table, specifying the mapping for each linear page number
- table[20-bit linear page #] => 20-bit phys page #
- PTE entries: bottom of handout
- 20-bit phys page number, present, read/write, user/supervisor, etc
- puzzle: can supervisor read/write user pages?

- can use paging hardware for many purposes
    - (seen some of this two lectures ago)
    - flat memory
    - segment-like protection: contiguous mappings
    - solve fragmentation problems when allocating more memory (xv6-like process memory layout)
    - demand-paging (%cr2 stores faulting address)
    - copy-on-write
    - sharing, direct access to devices (e.g. /dev/fb on linux)
    - switching between processes

- where is this table stored? back in memory.
- in our conceptual model, CPU holds the physical address of the base of this table.
- %cr3 serves this purpose on the x86 (with one more detail below)
- for each memory access, access memory again to look up in table

- why not just have a big array with each page #'s translation?
- same problems that we were trying to solve with paging! (demand-paging, fragmentation)
- so, apply the same trick

    - we broke up our 2^32-byte memory into 4096-byte chunks and represented them in a 2^22-byte (2^20-entry) table
    - now break up the 2^22-byte table into 4096-byte chunks too, and represent them in another 2^12-byte (2^10-entry) table
    - just another level of indirection
    - now all data structures are page-sized
- 386 uses 2-level mapping structure
- one page directory page, with 1024 page directory entries (PDEs)
- up to 1024 page table pages, each with 1024 page table entries (PTEs)
- so la has 10 bits of directory index, 10 bits table index, 12 bits offset
- %cr3 register holds physical address of current page directory
- puzzle: what do PDE read/write and user/supervisor flags mean?