CS353 Linux Kernel

Chentao Wu吴晨涛
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

6A. Memory Management -- Addressing

Chentao Wu
Associate Professor
Dept. of CSE, SJTU
wuct@cs.sjtu.edu.cn

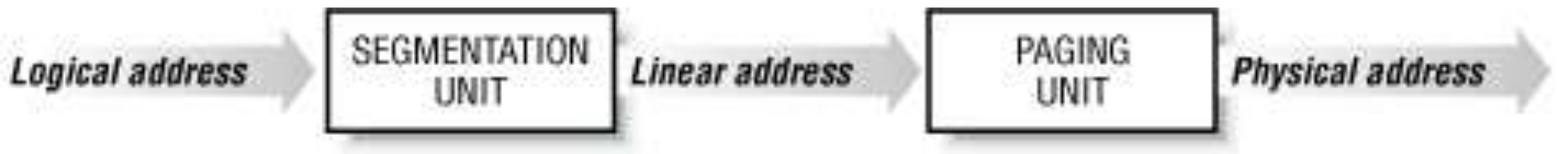
上海交通大學

Outline

- Memory Addresses
- Segmentation in Hardware
- Segmentation in Linux
- Paging in Hardware
- Paging in Linux

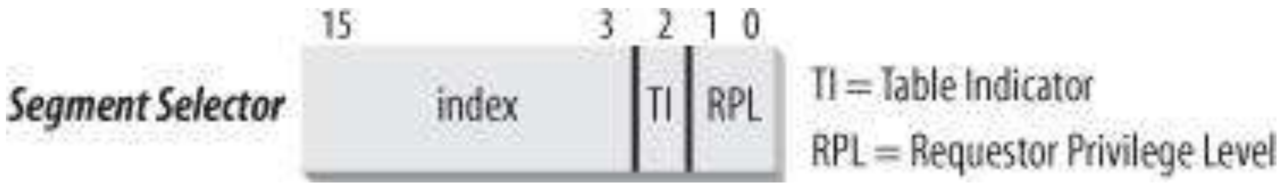
Memory Addresses

- 3 kinds of addresses in 80x86 microprocessors
  - *Logical address*
    - Included in the machine language instructions
  - *Linear address* (*virtual address*)
    - A single 32-bit unsigned integer that can be used to address up to 4GB
  - *Physical address* (32-bit unsigned integers)
    - Used to address memory cells in memory chips



Segmentation in Hardware (1)

- Logical address
  - Segment id: 16-bit (*Segment Selector*)
  - Offset: 32-bit



- Segment selector
  - Index: of the segment descriptor
  - TI (Table Indicator): GDT or LDT
  - RPL (Requestor Privilege Level)
- Segmentation registers
  - To hold segment selectors
  - Special purpose
    - cs: code segment, ss: stack segment, ds: data segment
  - General purpose: es, fs, gs

Segmentation in Hardware (2)

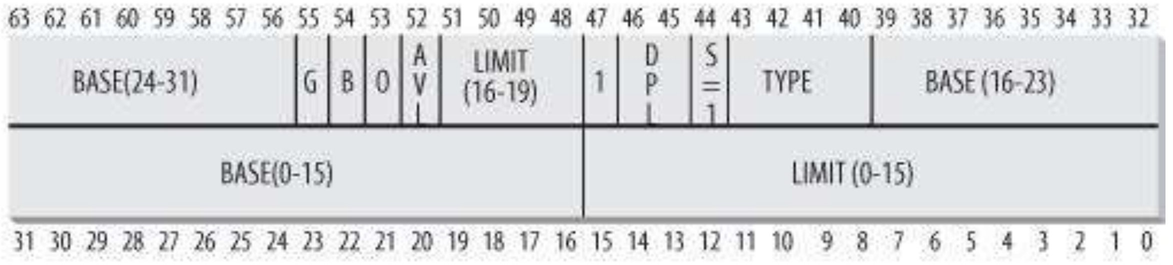
- Segment descriptors: 8-byte
  - Stored either in GDT (global descriptor table) or in LDT (local descriptor table)
  - Address/size of segment descriptor contained in processor control registers: gdtr, ldtr
- Fields in segment descriptor
  - Base: 32-bit linear address
  - G *granularity* flag: 1-bit (segment size in bytes or 4KB)
  - Limit: 20-bit offset
  - S system flag: 1-bit (system segment or not)
  - Type: 4-bit
    - Code, Data, Task Sate (TSSD), Local Descriptor Table (LDTD)

Segmentation in Hardware (3)

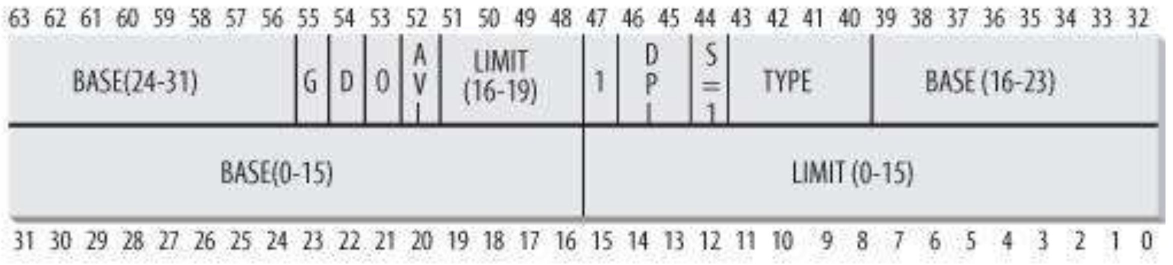
- DPL (descriptor privilege level): 2-bit
  - To restrict access to the segment
- Segment-present flag: 1-bit (in memory or not)
- D or B flag: 1-bit (depending on code or data)
- Reserved bit (bit 53): 0
- AVL flag: 1-bit (ignored by Linux)

Segment Descriptor Format

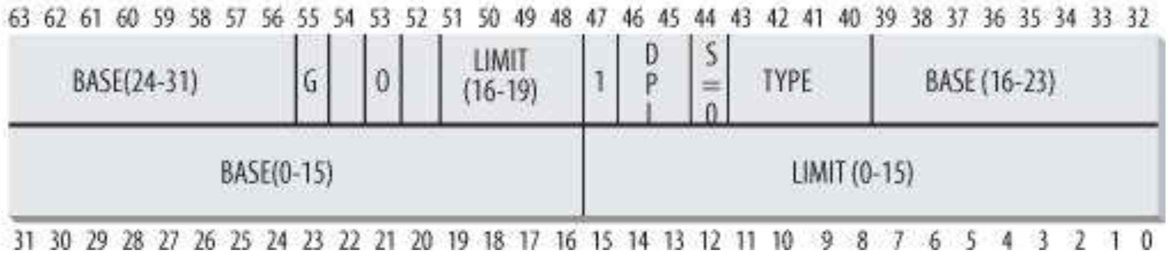
Data Segment Descriptor



Code Segment Descriptor



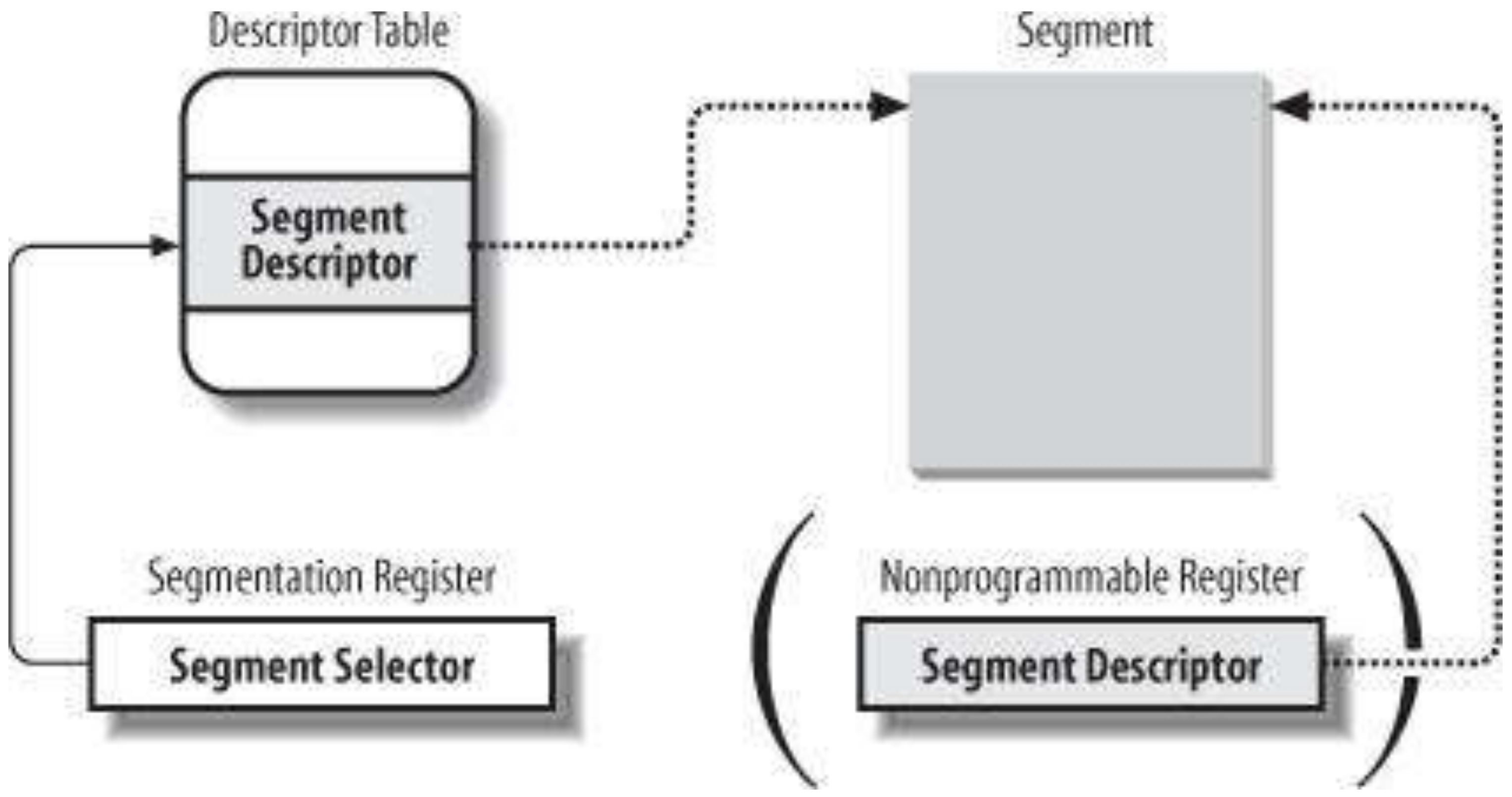
System Segment Descriptor



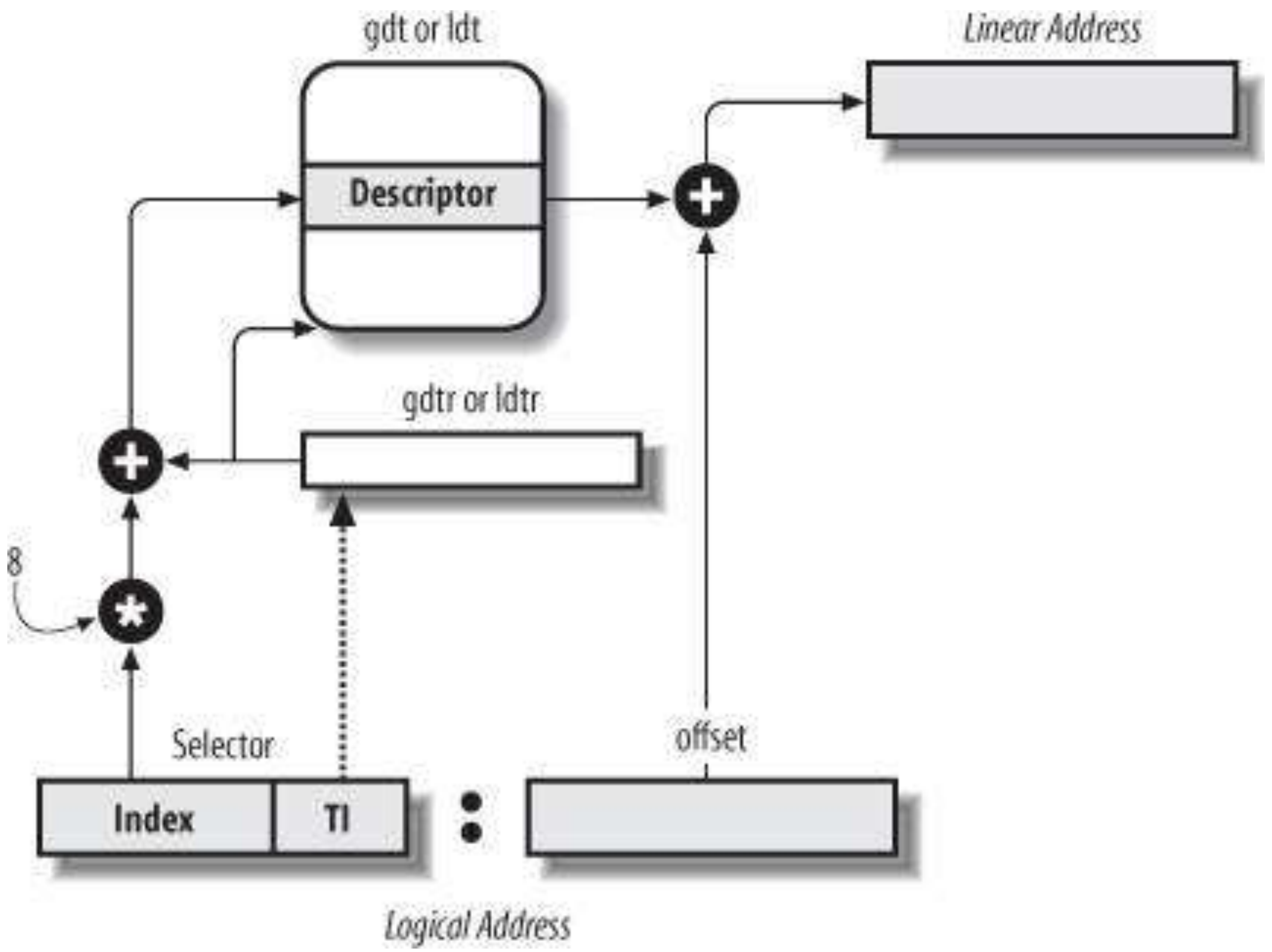
Fast Access to Segment Descriptors

- For each of the six programmable segmentation registers, 80x86 provides an additional nonprogrammable register, which is loaded every time a segment selector is loaded in a segment register
  - Without accessing the GDT or LDT in memory

Segment Selector and Segment Descriptor



Segmentation Unit



Segmentation in Linux (1)

- Used in a limited way
- Linux prefers paging to segmentation
  - Memory management is simpler
  - Portability to wide range of architectures such as RISC
- -> Linux 2.6 uses segmentation only when required

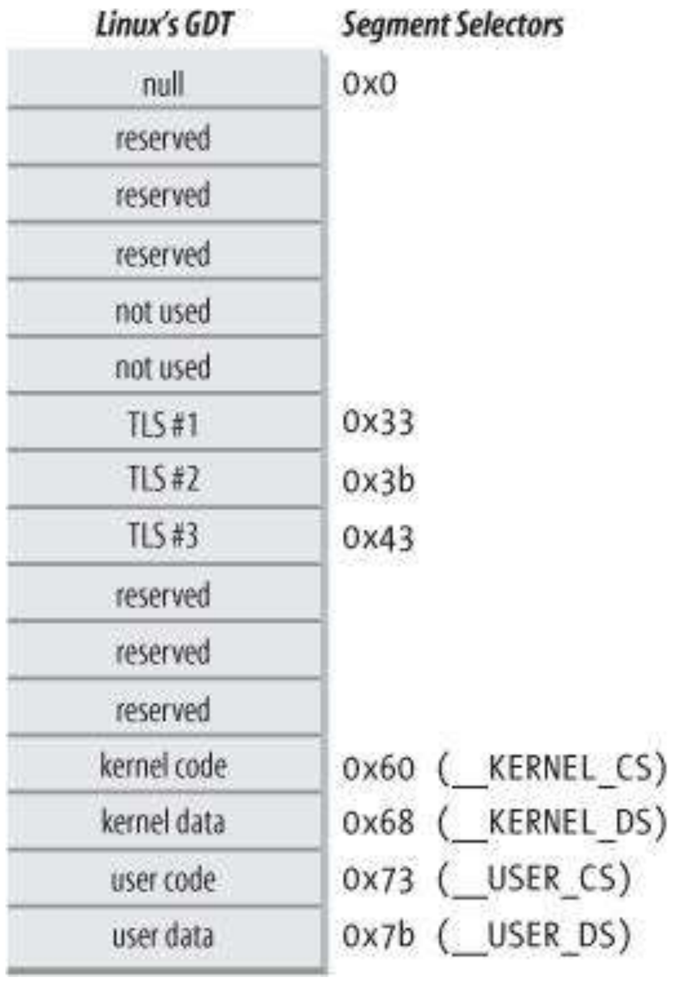
Segmentation in Linux (2)

- 4 main segments used by Linux
  - Kernel code segment: __KERNEL_CS macro
  - Kernel data segment: __KERNEL_DS macro
  - User code segment: __USER_CS macro
  - User data segment: __USER_DS macro

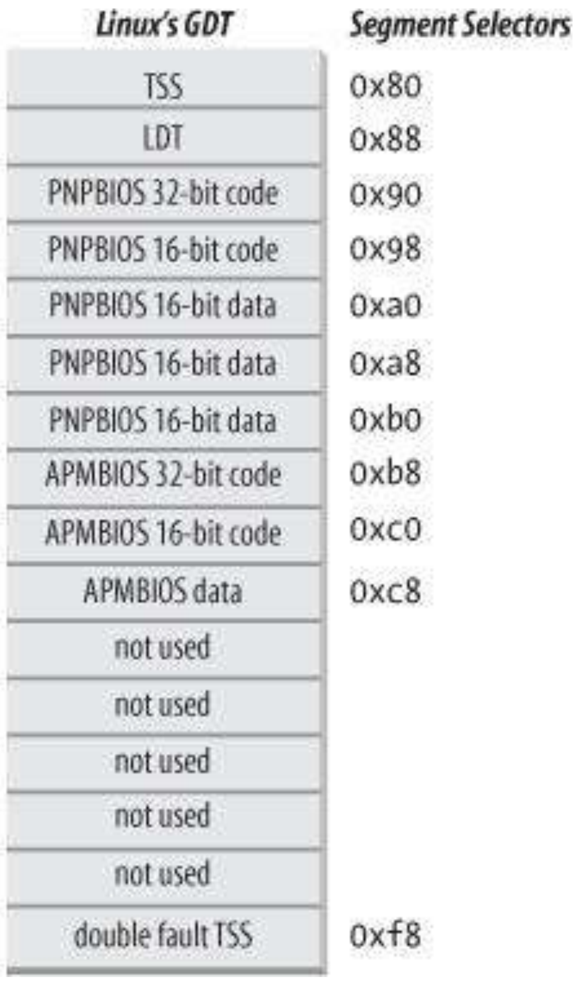
Linux GDTs (1)

- Linux GDTs
  - One GDT per CPU
    - Stored in cpu_gdt_table array
    - Addresses/sizes stored in cpu_gdt_descr array
- 18 segment descriptors
  - 4 user/kernel code/data segments
  - Task state segment (TSS): init_tss array
  - A default LDT: default_ldt
  - 3 Thread-Local-Storage (TLS) segments
  - 3 segments related to APM (Advanced Power Management)
  - 5 segments related to PnP (Plug and Play)
  - A special TSS segment to handle “double fault” exceptions

Linux GDTs (2)



Linux's GDT	Segment Selectors
null	0x0
reserved	
reserved	
reserved	
not used	
not used	
TLS #1	0x33
TLS #2	0x3b
TLS #3	0x43
reserved	
reserved	
reserved	
kernel code	0x60 (__KERNEL_CS)
kernel data	0x68 (__KERNEL_DS)
user code	0x73 (__USER_CS)
user data	0x7b (__USER_DS)



Linux's GDT	Segment Selectors
TSS	0x80
LDT	0x88
PNPBIOS 32-bit code	0x90
PNPBIOS 16-bit code	0x98
PNPBIOS 16-bit data	0xa0
PNPBIOS 16-bit data	0xa8
PNPBIOS 16-bit data	0xb0
APMBIOS 32-bit code	0xb8
APMBIOS 16-bit code	0xc0
APMBIOS data	0xc8
not used	
not used	
not used	
not used	
not used	
double fault TSS	0xf8

Linux LDTs (3)

- Default LDT: stored in default_ldt array
  - 5 entries included, but only two are effectively used by the kernel
    - A call gate for iBCS executables
    - A call gate for Solaris/x86 executables
    - *Call gates*: mechanism provided by 80x86 microprocessors to change the privilege level of the CPU

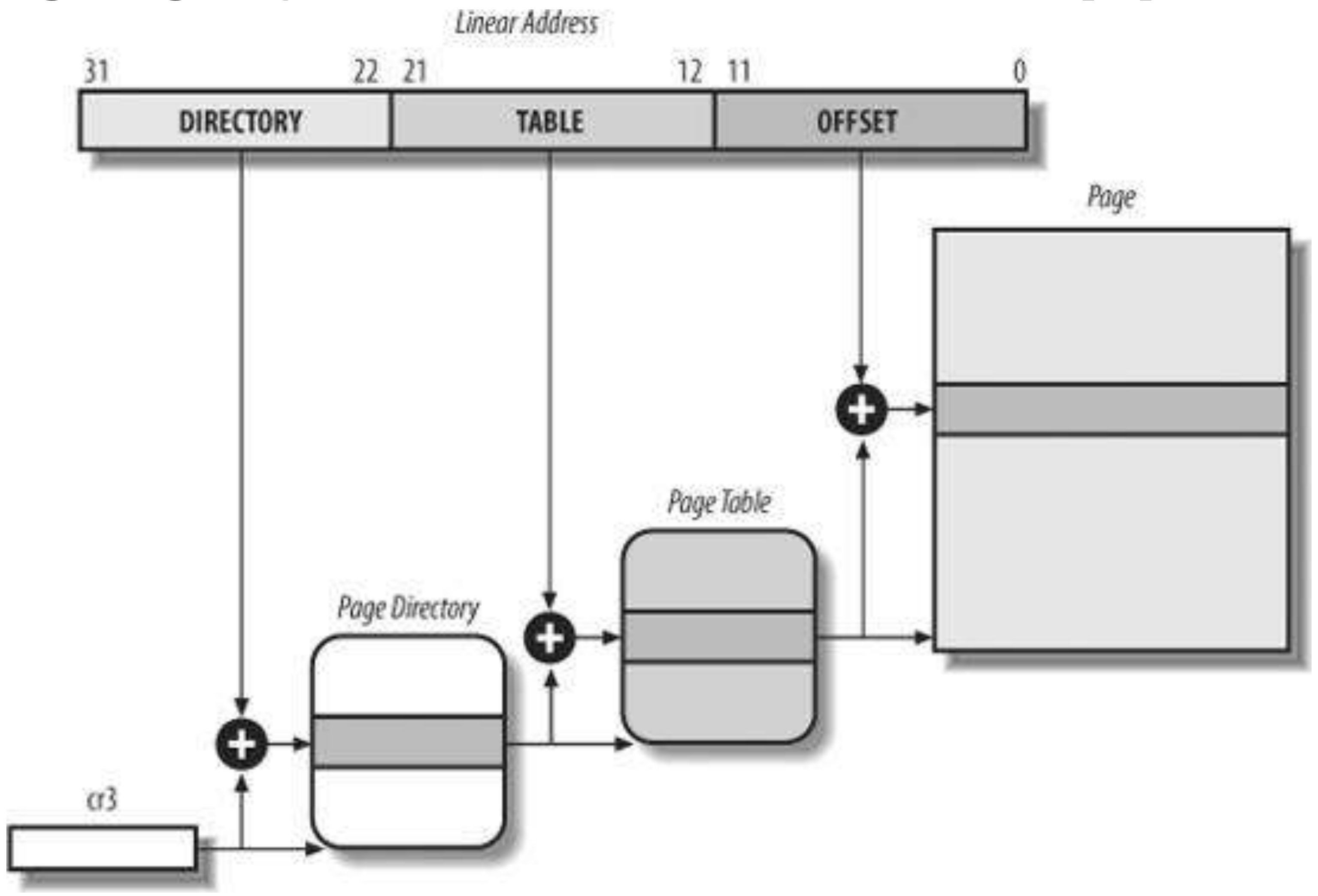
Paging in Hardware

- Pages: linear addresses grouped in fixed-length intervals
- Page frames (physical pages): RAM partitioned into fixed-length blocks
- In 80x86 processors, paging is enabled by setting the PG flag of control register cr0
  - 4KM pages

Regular Paging

- 4KB pages
  - Linear address: 32-bit
    - Directory: 10 bits
    - Table: 10 bits
    - Offset: 12 bits
  - Two-step translation
    - Page directory: physical address stored in cr3 register
    - Page table

Paging by 80x86 Processors (1)

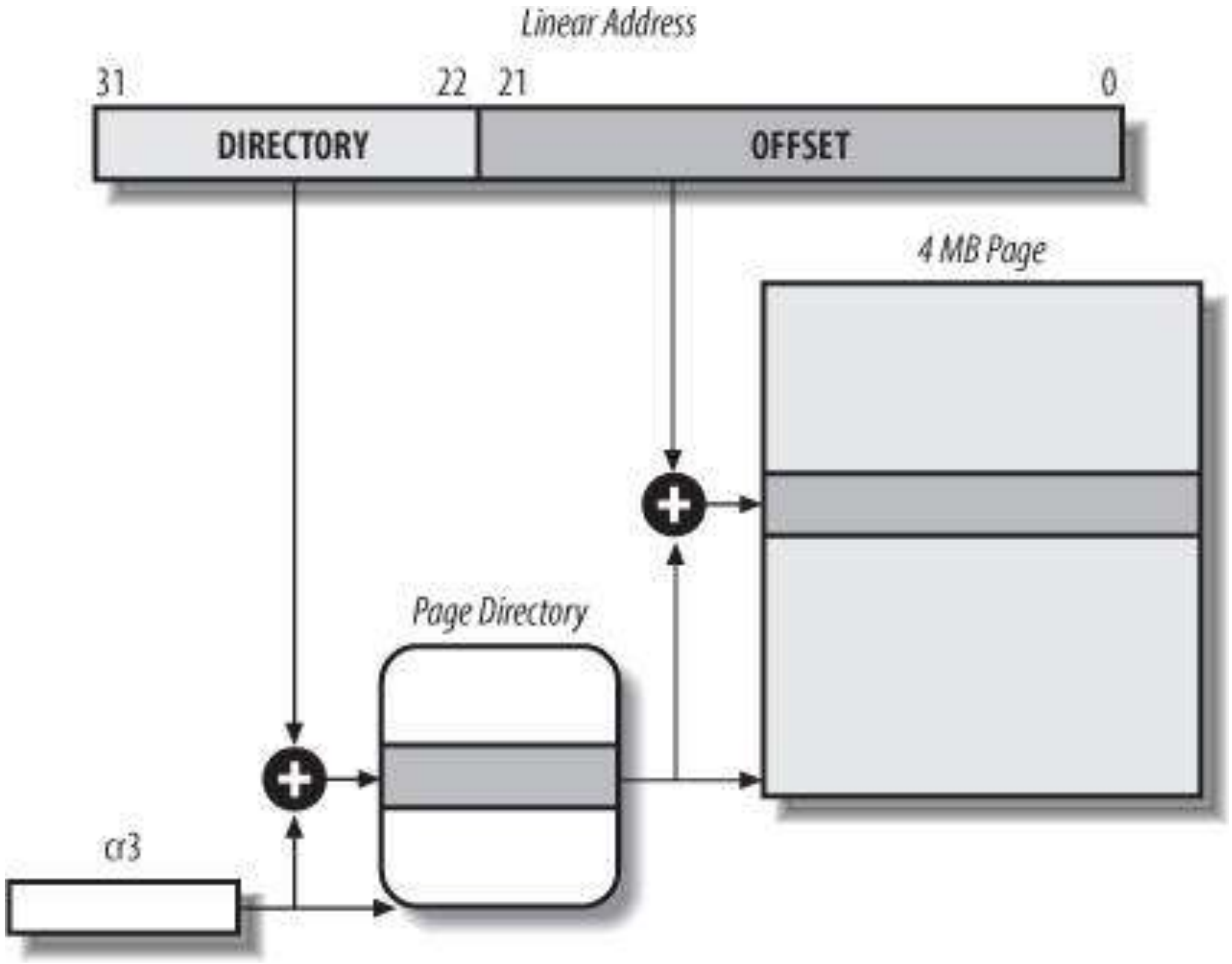


上海交通大學

Paging by 80x86 Processors (2)

- Same structure for the entries in page directories and page tables
  - Present flag: in memory or not
  - 20-MSB of a page physical address
  - Accessed flag: used to select pages to be swapped out
  - Dirty flag: applies only to page table entries
  - Read/write flag: access right of the page
  - User/supervisor flag: privilege level
  - PCD and PWT flag: hardware cache
  - Page size flag: applies only to page directory entries (2MB or 4MB)
  - Global flag: applies only to page table entries (to prevent frequently used pages from being flushed from the TLB cache)

Extended Paging (1)



Extended Paging (2)

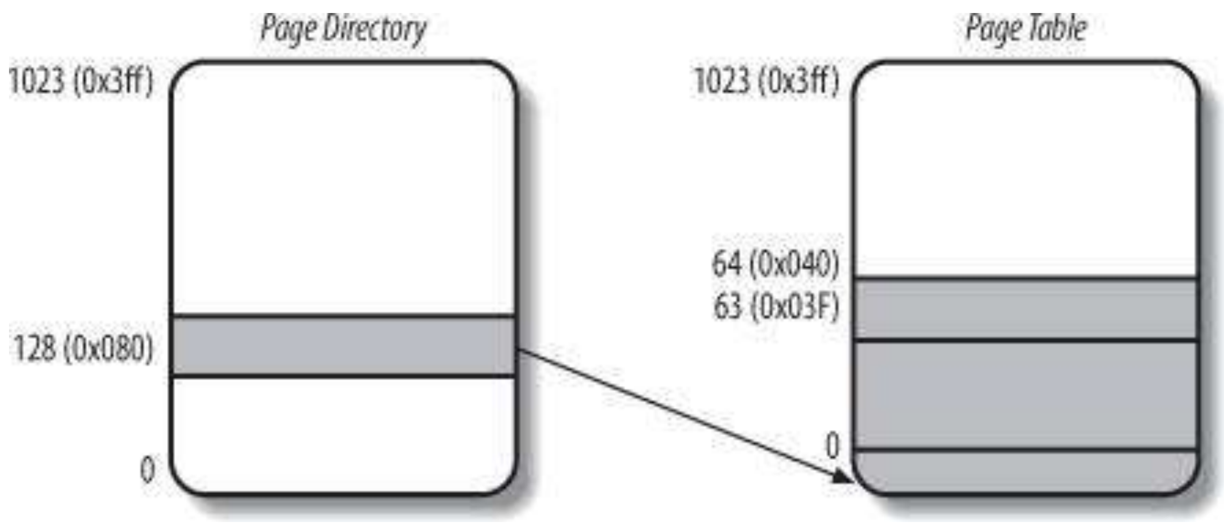
- Used to translate large contiguous linear address ranges
  - Page size: 4MB
- Linear address: 32 bits
  - Directory: 10 bits
  - Offset: 22 bits
- Page directory entries for extended paging are the same as for regular paging, except that:
  - The Page Size flag must be set
  - Only the 10 most significant bits of the 20-bit physical address are significant

Hardware Protection scheme for Paging

- Hardware protection scheme
  - Only two privilege levels associated with pages
    - by user/supervisor flag
  - Only two types of access rights associated with pages
    - by read/write flag

An Example of Regular Paging

- Ex: 0x20000000-0x2003ffff



Zero

Valid

Physical Address Extension (PAE) Paging Mechanism (1)

- Starting with Pentium Pro, the number of address pins are increased to 36
  - Up to 64GB RAM
  - PAE is activated by setting the PAE flag in cr4 control register

Physical Address Extension (PAE) Paging Mechanism (2)

- Paging mechanism changes
  - 2^{24} page frames, physical address field of page table entries expanded from 20 to 24 bits
  - A new level of page table called PDPT (Page Directory Pointer Table)
    - Consists of 4 64-bit entries
    - Cr3 register contains 27-bit PDPT base address field

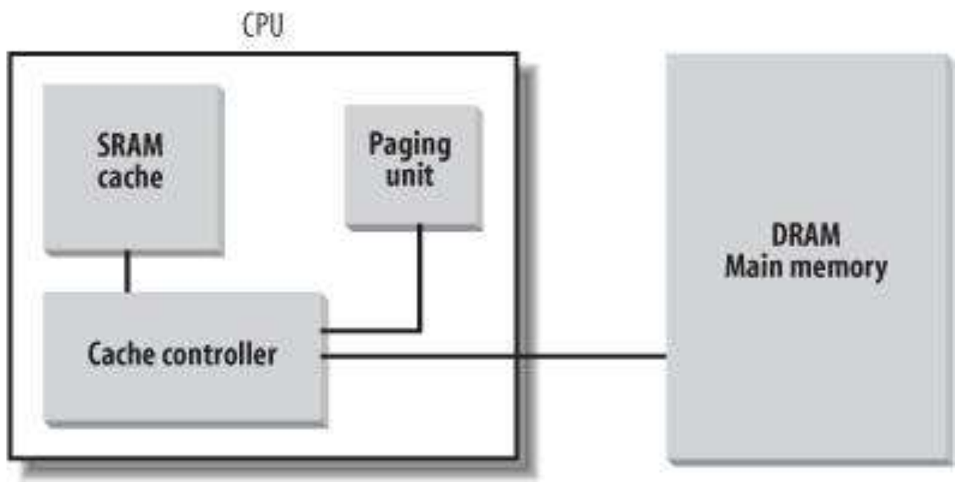
Physical Address Extension (PAE) Paging Mechanism (3)

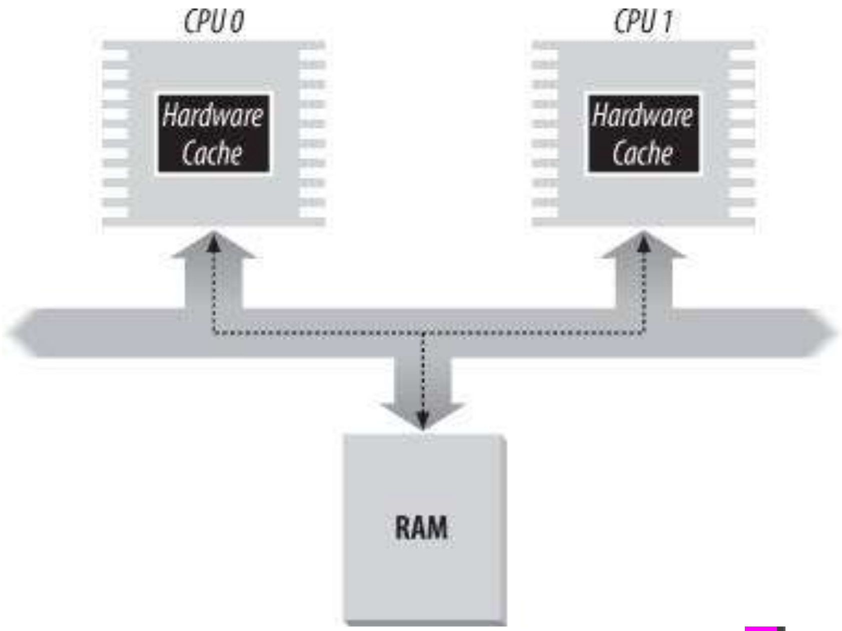
- When mapping 4KB pages
  - Bits 31-30: points to entries in PDPT
  - Bits 29-21: points to entries in page directory
  - Bits 20-12: points to entries in page table
  - Bits 11-0: offset
- When mapping 2MB pages
  - Bits 31-30: points to entries in PDPT
  - Bits 29-21: points to entries in page directory
  - Bits 20-0: offset

Paging for 64-bit Architectures

- Two-level paging is not suitable
  - The number of levels depends on the type of processor
    - 3-level: alpha, ia64, ppc64, sh64
    - 4-level: x86_64

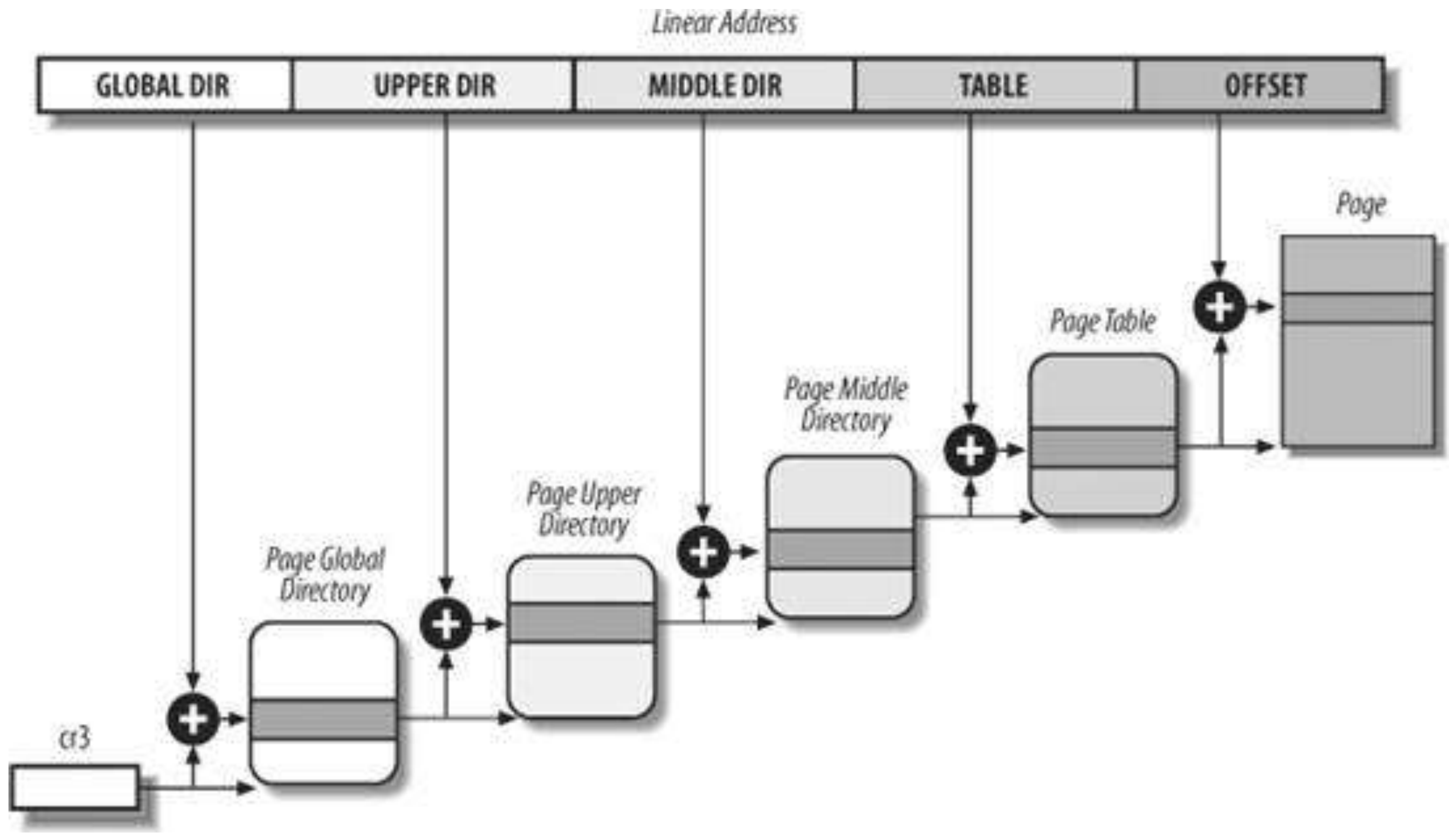
Hardware Cache





- TLB: Translation Lookaside Buffers

Paging in Linux



上海交通大學

Paging in Linux

- A common paging model for both 32-bit and 64-bit architectures
  - Up to Linux version 2.6.10, 3-level paging
  - Starting with 2.6.11, 4-level paging
- With no PAE, 2-level paging is enough
  - Linux essentially eliminates the Page Upper Directory and Page Middle Directory fields
- With PAE, 2-level paging is used
  - Page Global Directory: x86's PDPT
  - Page Upper Directory: (eliminated)
  - Page Middle Directory: x86's Page Directory
  - Page Table: x86's Page Table

Page Table Handling Functions/Macros (1)

- Macros for simplifying page table handling:
  - PAGE_SHIFT, PMD_SHIFT, PUD_SHIFT, PGDIR_SHIFT
  - PTRS_PER_PTE, PTRS_PER_PMD, PTRS_PER_PUD, PTRS_PER_PGD
- Data structures for page table handling:
  - pte_t, pmd_t, pud_t, pgd_t
  - pgprot_t
- Macros for page table type conversions:
  - Macros: __pte, __pmd, __pud, __pgd, __pgprot
  - Macros: pte_val, pmd_val, pud_val, pgd_val, pgprot_val

Page Table Handling Functions/Macros (2)

- Macros and functions to read or modify page table entries:
  - pte_none, pmd_none, pud_none, pgd_none
  - pte_clear, pmd_clear, pud_clear, pgd_clear
  - set_pte, set_pmd, set_pud, set_pgd
  - pte_same(a,b), pmd_large(e)
  - pte_present, pmd_present, pud_present, pgd_present
- Macros to check page table entries
  - pmd_bad, pud_bad, pgd_bad
- Functions to query the current value of any flag in a page table entry:
  - pte_user(), pte_read(), pte_write(), pte_exec(), pte_dirty(), pte_young(), pte_file()
- Functions to set the value of flags in a page table entry:
  - mk_pte_huge(), pte_wrprotect(), pte_rdprotect(), pte_exprotect(), pte_mkwrite(), pte_mkread(), pte_mkexec(), pte_mkdirty(), pte_mkclean(), pte_mkyound(), pte_mkold(), pte_modify(p,v),
  - ptep_set_wrprotect(), ptep_set_access_flags(), ptep_mkdirty(), ptep_test_and_clear_dirty(), ptep_test_and_clear_young()

上海交通大學

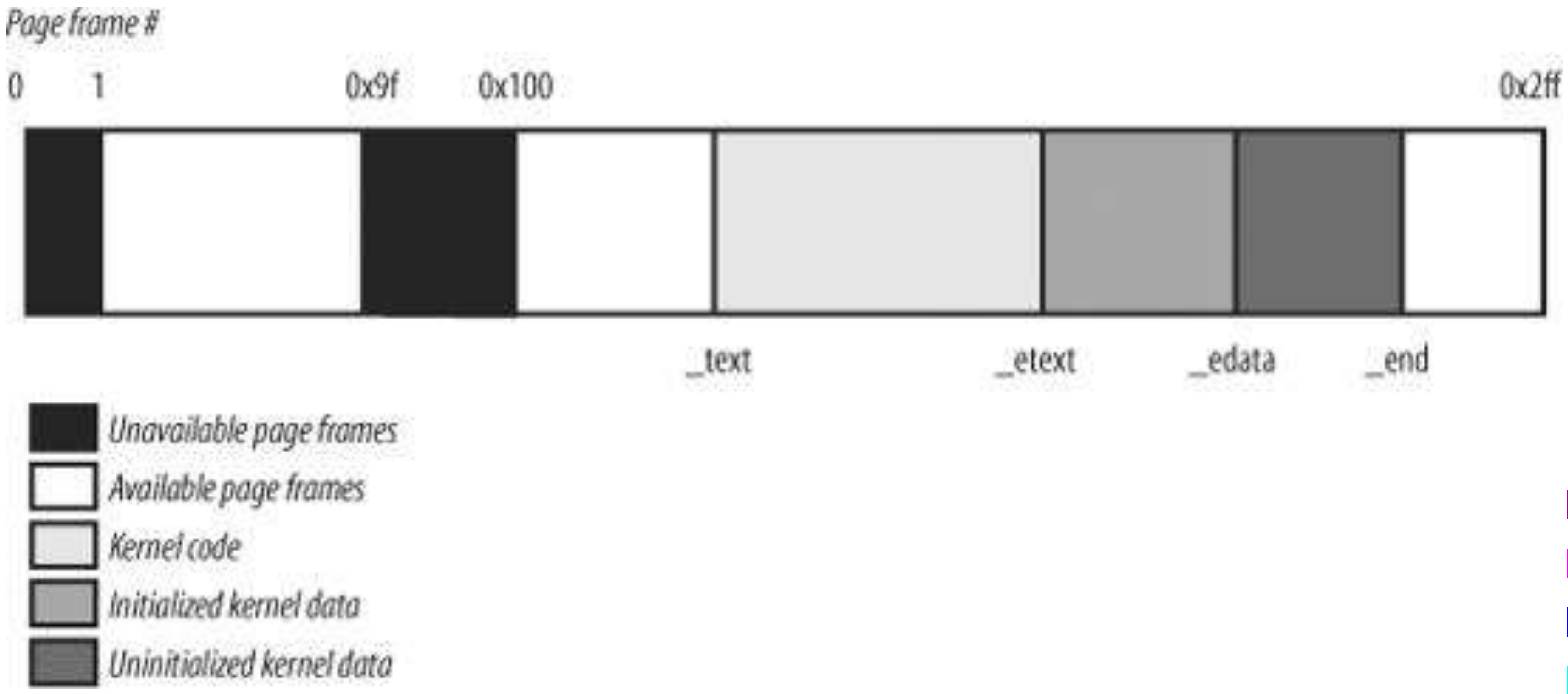
Page Table Handling Functions/Macros (3)

- Macros to combine/extract page address into/from a page entry
  - mk_pte, mk_pte_phys, pte_page(), pmd_page(), pgd_offset(p,a), pmd_offset(p,a), pte_offset(p,a)
- Functions to create and delete page table entries:
  - pgd_alloc(m), pud_alloc(m, p, a), pmd_alloc(m,p,a), pte_alloc(m,p,a)
  - pte_free(p), pmd_free(x), pud_free(x), pgd_free(p), free_one_pmd(), free_one_pgd(), clear_page_tables()

Physical Address Layout

- In general, Linux kernel is installed in RAM starting from the second megabyte (0x00100000)
    - Page frame 0: used by BIOS (to store system configuration detected during POST
    - 0x000a0000 to 0x000fffff: reserved to BIOS (to map ISA cards)
    - Some page frames within the first megabytes may be reserved by specific computer models

The First 768 Page Frames (3MB) in Linux 2.6



Project 3:
Memory Management

Memory management homework

- Write a module that is called mtest
- When module loaded, module will create a proc fs entry /proc/mtest
- /proc/mtest will accept 3 kind of input
  - **"listvma" will print all vma of current process in the format of**

    start-addr end-addr permission

    e.g

    0x10000   0x20000 rwx

    0x30000   0x40000 r—
  - **"findpage addr" will find va->pa translation of address in current process's mm context and print it. If there is not va->pa translation, prink "tranlsation not found"**
  - **"writeval addr val" will change an unsigned long size content in current process's virtual address into val. Note module should write to identity mapping address of addr and verify it from userspace address addr.**
- All the print can be done with printk and check result with dmesg.