ECE391
Computer System Engineering

Lecture 15

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

Spring 2021

Lecture Topics

- x86 support for VM
  - protection model
  - segmentation
  - paging

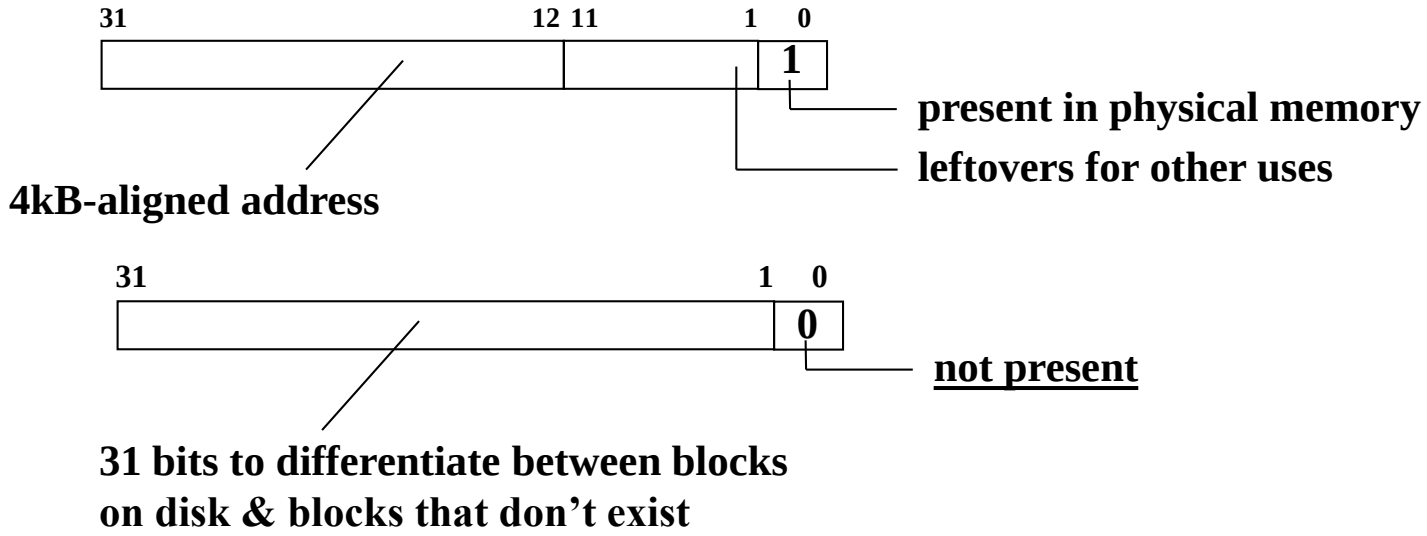
x86 Paging

- Paging is the second level of indirection in x86

- Each page of a virtual address space is in one of three states
  - doesn't exist
  - exists and is in physical memory
  - exists, but is now on the disk rather than in memory

x86 Paging

- We can encode these possibilities as follows using 4B

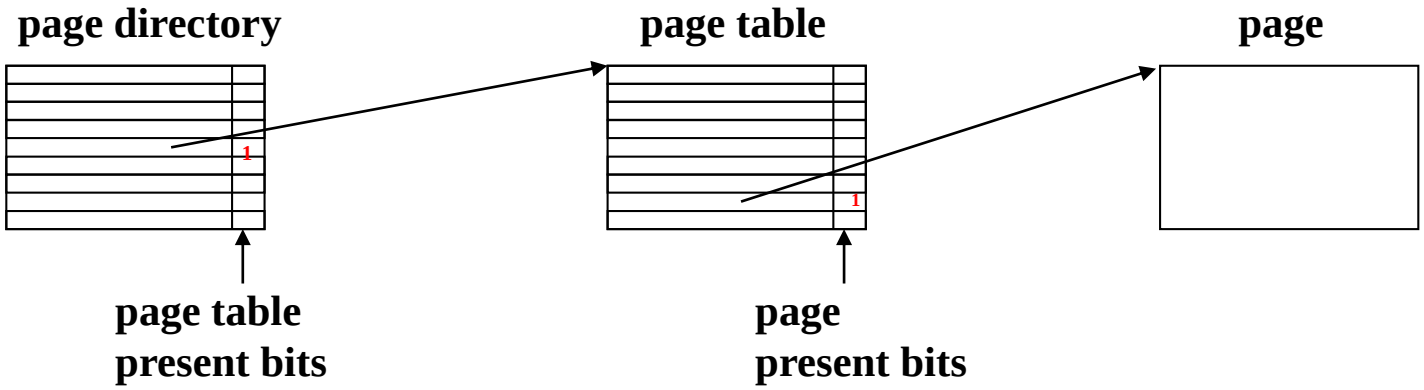


- These 4B are called a page table entry (PTE); a group of them is a page table
- Question: if we use 4B for every 4kB, how big is the page table for a single virtual address space?

$$(4 / 4096) \times 2^{32} = 4\text{MB} \quad \text{too big…}$$

x86 Paging

- Solution?
  - page the page table
  - i.e., use a hierarchical structure

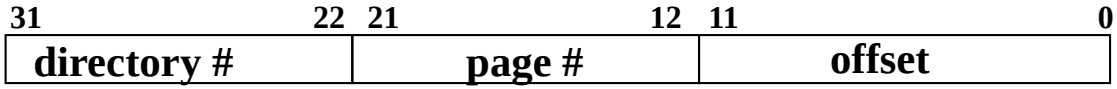


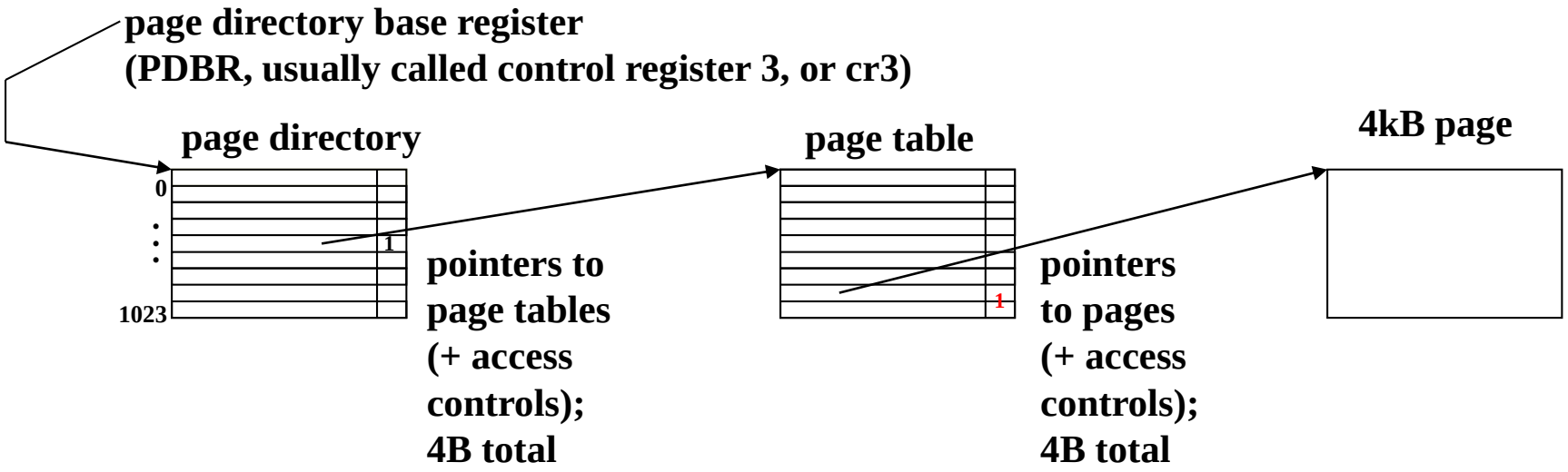
- The page table is just another 4kB page
  - it holds 4096 / 4 = 1024 PTEs

x86 Paging

- What about the page directory?
  - given
    - 2^{32} bytes total (32-bit address space)
    - 2^{10} PTEs per table
    - 2^{12} bytes per page
  - the page directory needs $2^{32}/(2^{10} \times 2^{12}) = 2^{10}$ entries total
  - which also fits in one page
    (and could be paged out to disk, although Linux does not)

x86 Paging


31
22
21
12
11
0
directory #
page #
offset


page directory base register
(PDBR, usually called control register 3, or cr3)
page directory
0
1023
1
pointers to
page tables
(+ access
controls);
4B total
page table
1
pointers
to pages
(+ access
controls);
4B total
4kB page

x86 Paging

- To translate a virtual address into a physical address
  - start with the PDBR (cr3)
  - look up page directory entry (PDE) using the 10 MSb of virtual address
  - extract page table address from PDE
  - look up page table entry (PTE) using next 10 bits of virtual address
  - extract page address from PTE
  - use last 12 bits of virtual address as offset into page

x86 Paging

- Way too slow to do on every memory access!

- Hence the translation lookaside buffers (TLBs)
  - keep translations of first 20 bits around and reuse them
  - only walk tables when necessary (in x86, OS manages tables, but hardware walks them)
  - TLBs flushed when cr3 is reloaded

x86 Paging

- Remember the 11 free bits in the PTEs?
- What should we use them to do?
  - protect
  - optimize to improve performance

- Protect
  - User/Supervisor (U/S) page or page table
    - User means accessible to anyone (any privilege level)
    - Supervisor requires PL < 3 (i.e., MAX (CPL,RPL) < 3)
  - Read-Only or Read/Write

x86 Paging

- Optimize
  - TLBs must be fast, so you can't use many (~32 or 64)
  - nice if
    - some translations are the same for all programs
    - bigger translations could be used when possible
      (e.g., use one translation for 4MB rather than 1024 translations)

x86 Paging

- x86 supports both
  - **G flag—global**
    - TLB not flushed when changing to new program or address space (i.e., when cr3 changes)
    - used for kernel pages (in Linux)
  - **4MB pages**
    - skip the second level of translation
    - indicated by PS (page size) bit in PDE
    - PS=1 means that the PDE points directly to a 4MB page
    - remaining 22 bits of virtual address used as offset
    - x86 provides separate TLBs for 4kB & 4MB translations

Important Bit Settings to Enable/Use Paging

- Paging is optional and must be enabled for use
  - PG (page enable): bit 31 in CR0 (control register 0)
- Allow mixed page sizes (i.e., pages of size 4kB and 4MB)
  - PSE (page size extension) bit 4 in CR4 (control register 4)
- Allow 4MB pages
  - PS (page size), bit 7 in PDE (page directory entry) must be set to 1

MP3: System Initialization (example)

x86_desc.S - Set up x86 segment descriptors, descriptor tables

```
#define ASM 1
#include "x86_desc.h“
.....
 tss_size:
.long tss_bottom - tss - 1

ldt_size:
.long ldt_bottom - ldt - 1

ldt_desc:
.word KERNEL_LDT
.long ldt

gdt_desc:
// <gdt size>
// <gdt address>
.
tss:
_tss:
.rept 104
.byte 0
.endr
tss_bottom:
```

gdt:

```
_gdt:
# First GDT entry cannot be used
.quad 0
# NULL entry
.quad 0
.....
. gdt_bottom:

ldt:
.rept 4
.quad 0
.endr
ldt_bottom:
.....
idt_desc_ptr:
.word idt_bottom - idt - 1
.long idt
.align 16

idt:
_idt:
.rept NUM_VEC
.quad 0
.endr
idt_bottom:
```

MP3: System Initialization (example)

boot.S - start point for the kernel after GRUB gives us control

#define ASM 1#include "multiboot.h“
#include "x86_desc.h“

……

 # Load the GDT

lgdt (????)

……

.

.