Lecture 5:

Intro to Paging

Last Time: [Fall '11] [VM: paging intro]

Virtualizing Memory:
each process gets its own private memory (illusion)

Key Mechanism: Address Translation (V→P) (efficient, control)

Early approaches: → Base/Bounds (Dyn. Rel)
→ segmentation (gen. b/b)
[review]

H/W: involved on every mem reference (fast)

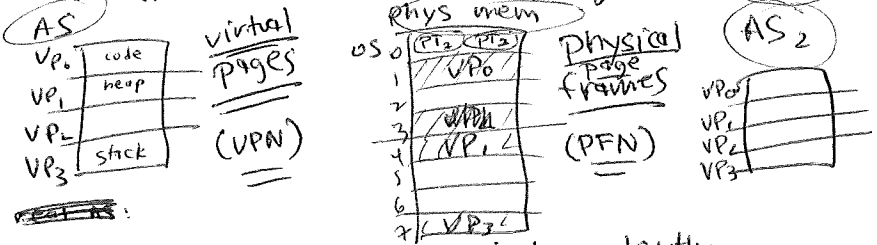
OS: when involved? what does it do?

Problems w/ segmentation?
⇒ still not super flexible (e.g. large but mostly unused heap)
⇒ external frag (OS issue)

Paging: also age old technique, used in virt. all modern systems

concept: divide all of memory into
⇒ fixed-sized units ("pages")

typical: 4KB



OS: free list: of frames that are unused

⇒ relocate each page independently in memory

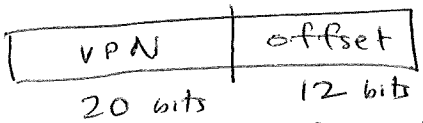
OS/HW: must have a lot of addr. translation info stored in memory (of OS) in [page table]

Example: 64-byte AS, w/ 16 byte pages
⇒ show all VAs for VP0
⇒ show page table, PTE
Now, do one whole translation

} see next pages

Bad
Problems: Two: size, time

Linear Page Table: (just an array)
32-bit virtual AS w/ 4KB pages



how big is table? 2^{20} * sizeof(PTE) ⇒ BIG

how many tables are there? (1/process)

Speed: per (logical) memory access how mem. accesses? ⇒ SLOW

2 (for linear)

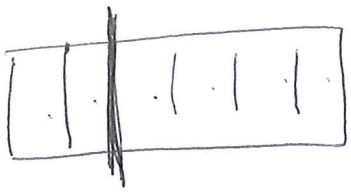
h/w involved: on every mem reference

OS involved: when?
new proc: find pages for init code, stack
grow proc: add new heap page
proc exits: free pages (put on free list)

ctxt switch: [what to do?]

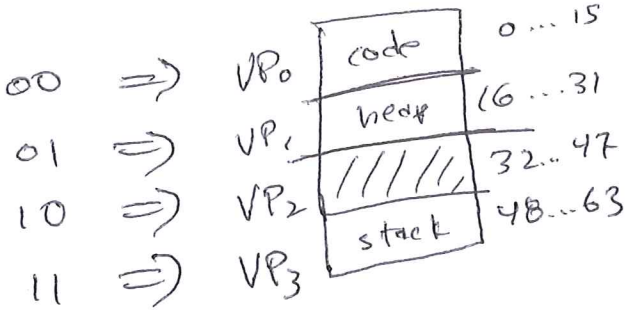
Good ⇐ compare to segmentation
→ ext frag?
→ general sparse AS?

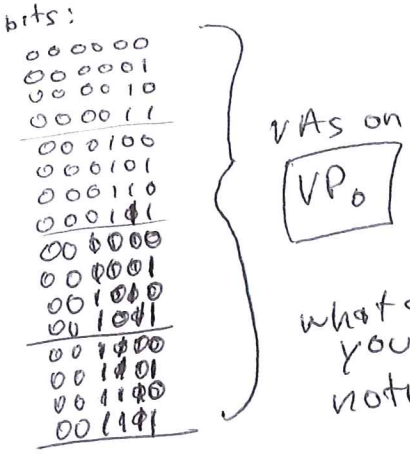
Example (1)



2^6 ⇒ 64 bytes AS

16 byte pages





what do you notice?

⇒ top 2 bits are always [00]

Page Table:
for each virtual page:
which physical ~~address~~ frame is it located in?

logically:

(VPN=0) ⇒ PFN = 1 (001)

VPN=1 ⇒ PFN = 4 (100)

VPN=2 (not valid) ← ah!

VPN=3 ⇒ PFN = 7 (111)

Page Table Entry

V	PFN

in

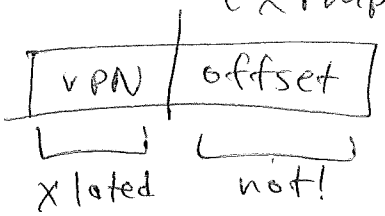
↳ (valid bit)

⇒ [load 17, R1]

Example (2)

VA: 17 ⇒ 01|0001

split into ⇒ VPN | offset



VPN: use to look up PTE in page table

offset: concat to PFN to form final physical address

Problem: hardware must do translation · but where is page table for this process?

PTBR: holds physical address ~~of~~ of page table for this process

problem: h/w must know exact structure of PT

PTE: | V | PFN | 4 bits: unused prot | RWX M OS

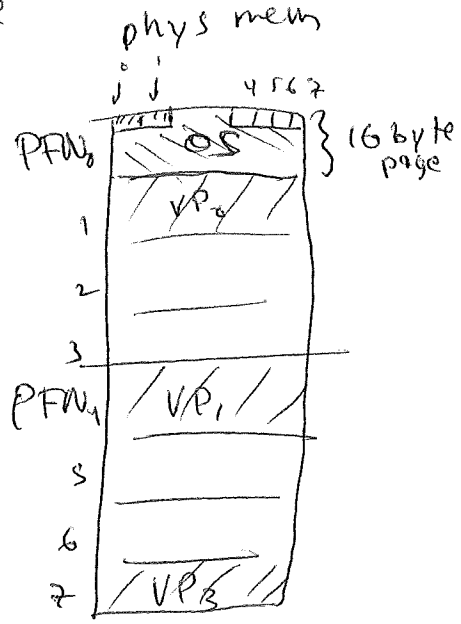
how many bits? ⇒ 3

PTE size: 4 bits

Entire Page Table: How Big?

	V	PFN
VPN0	V	"
VPN1	V	"
VPN2	V	"
VPN3	V	"

4 bits × 4 ⇒ ~~32~~ bits ⇒ ~~4~~ bytes, 1 per PTE



PTBR: 4

VPN | offset

01 | 0001

(address formulation)

index into PT : PTBR + (VPN * sizeof(PTE))

ALL IN H/W

FETCH PTE for VPN1 → PFN, offset

if (VALID == 0)

Fault()

PA = (PFN << PAGESHIFT) | offset;

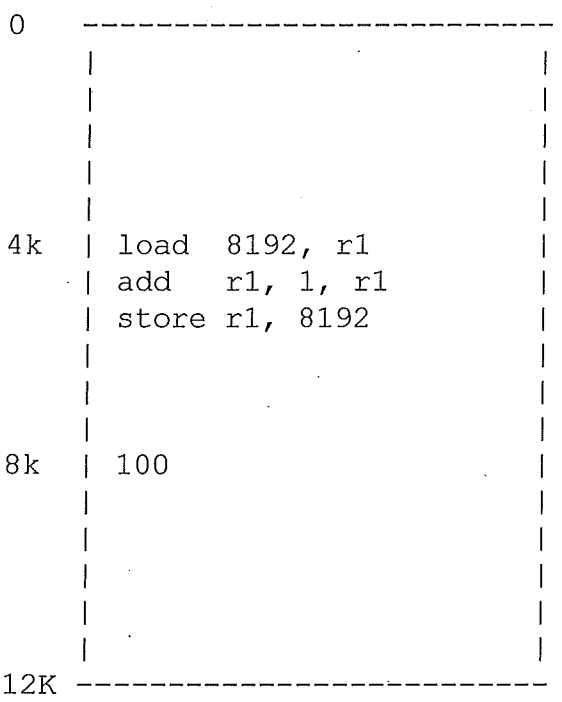
FETCH PA ⇒ R1

Here is some assembly code:

```
load  8192 r1      # load value at memory location 8192 -> r1
add   r1, 1, r2    # put r1 + 1 -> r2
store r2, 8192     # stores r2 -> memory location 8192
```

Assume each instruction takes up 4 bytes in memory.

Assume the program counter (PC) is set to 4096 (4k) when running the first instruction of this sequence. The virtual address space of this process looks like this (not to scale):



The total size of this virtual address space is 12 KB.

Assume this is a system with "base and bounds" registers used for memory relocation and protection.

In this example, assume that the process's address space is loaded into physical memory at physical address 64 KB (this is the base). The bounds is set to the size of the address space: 12 KB.

List all the physical memory locations that are referenced during the execution of this three-instruction sequence.