2013CS50286

Anket Hirulkar

Homework:3

09

10 • Assignment: paging

VA = 0X80100000 has 32 bits

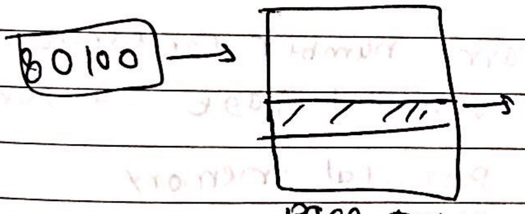
11 Divide into $\boxed{20, 12}$ bits .

12 | 80100 | 000 |

virtual page number

01 page offset (Tells where we are in the page)

02



03 frame number corresponding to virtual page number in physical memory

04 Page Table lookup

05 physical address PA = frame number . $\underbrace{000}_{\text{Top 12 bits}}$

06

• Page table reload

07

★ Output of print/X kpgdir[0]

= $1 = 0x0

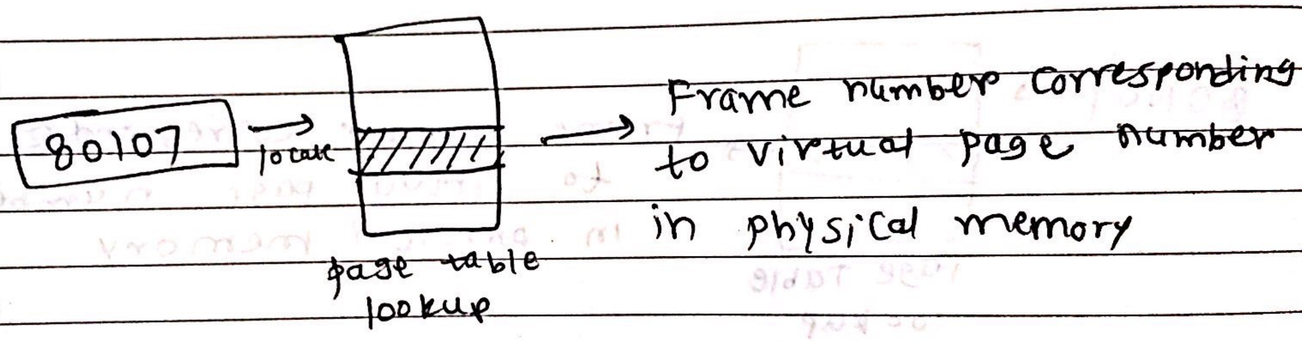
Reason: XV6 arranges for each process' memory to be contiguous and start at virtual address 0. Therefore the first address in page directory is seen as 0x0.

★ (gdb) x/i kvmalloc

09 Out: 0x80107beb : push %ebp

10 ⇒ conversion to physical address

80107	beb
20 bits	12 bits
Virtual page number	page offset (Tells where we are in the page)



Frame number corresponding to virtual page number in physical memory

05 Physical address $PA = \text{Frame number}.\underbrace{beb}_{\text{Top 12 bits}}$

07★ Print/x kpgdir[0x200]
$ 0x114007

⇒ ~~this~~ We had obtain 0x200 by taking 10 highest bits of address 0x80107beb (through operation 0x80107beb >> 22 right shift operation)

$2^{10 \text{ bit}} = 1024 = 1$ MB memory.

In XV6, there are memory mapped devices at physical address < 1 MB.

(Continued on next page)

09 bootmain loads kernel starting at 0X100000. Stored in DRAM.

10

So the kernel process we started is at location
11 0X 114000 in physical memory's DRAM.

12 ✱ What is the PPN?

The paging hardware translates a linear address
01 by using its top 20 bits to index into the page table to find a PTE, and replacing those bits
02 with the PPN in the PTE.

03 PPN is 20 bit physical page number

04 ∴ in 0X114007 → top 20 bits

$\boxed{= 0X11400 \text{ is PPN}}$

05

06 ✱ What was 7 here in 0X114007?

07 Answer: 7 = 0111 in binary.
4 2 1
Thus 3 ending bits are 1.

This conveys the following information:

8 ① PTE-P=1. Tells us PTE is valid. So the reference to page will not cause fault.

② PTE_W=1. Tells whether instructions are allowed to issue writes to the page.

③ PTE-U=1. Tells us ~~that not just kernel but~~ user programs are allowed to use the page.

★ Addressing

If we modified bootmain() to add 0x100000 to the va/pa of each ELF section:

kernel might not be able to find itself after it starts.

To fix the problem, we should be changing the boot address in the ELF header.

Also if a process tries to access 0x100000 to 0x200000 address between this, it will run into error (seg fault)

★ Traps:

A process can have three sets of saved registers in its stack if:

The fork() child process of a parent process switched to kernel mode.

① Is it possible to have two "context" structures and one "trapframe" structure on ~~the~~ kstack?

→ Since the trapframe can contain information saved about multiple processes, yes, it is possible

② Is it possible to have two trapframe structures & one context structure on kstack?

→ No. Since when returning or resuming the old process, it must be able to locate the trapframe where information about process is saved. It is not possible to have two trapframes for a single process.

☆ Context switching

09

A process that is running in the kernel
10 calls sched(), which ends up jumping into scheduler().

11

Q: Where is the stack that sched() executes on?

12 ⇒ In the context structure

Q: 01 Where is the stack that scheduler() executes on?

02 ⇒ In the trapframe structure.

03

Q: When sched() calls swtch(), does that call to
04 swtch() ever return? If so when?

⇒

05 The scheduler checks for RUNNABLE processes. ~~If there~~ are no RUNNABLE process or a process
06 is done executing, the swtch returns

07

☆ What is the four character pattern?

⇒ The four character patter is "acbd"

☆ ~~why are very~~ first character "ac"?

⇒ When sched calls its cprintf, it takes time for the characters to get printed on screen as it is called from process /user level. scheduler runs in kernel mode ans printf

/k	M	T	W	T	F	S	S
4			1	2	3	4	5
5	6	7	8	9	10	11	12
6	13	14	15	16	17	18	19
7	20	21	22	23	24	25	26
8	27	28	29	30			

"a" quickly. Then "c" is printed.

2

1

2

03

04

05

06

07