GAJERA DEV 2016EE10431

COL331 Homework 3

Bunk Pages

⇒ Assignment : Paging.

va : 0x80100000 ⟶ pa : 0x00100000

↓

top 10 bits : 1000000000 = 512

next 10 bits : 0100000000 = 256

512th entry of page directory will point to a page table, 256th entry of that page table will point to physical address 0x00100000. Permissions : PTE_P (PTE_W is not set to make it read only)

⇒ Assignment : Pagetable reload

(gdb) print/x kpgdir[0]

→ Why is this zero ?

↳ At this point, kpgdir is setup for the kernel part of the page table by setupkvm() function. It has mapped pages above KERNBASE address, and 0 to KERNBASE is still empty and available for user processes. That's why kpgdir[0] is zero.

→ How would we translate 0x80107beb to a physical address ?

↳ (for xv6) V2P_WO() function can translate virtual address to physical address. We can do it by subtracting 0x80000000 (KERNBASE).

0x80107beb - 0x80000000

= 0x107beb

(gdb) print/x 0x80107beb >> 22

$4 = 0x200

(gdb) print/x kpgdir[0x200]

$6 = 0x114007

→ what is this ?

↳ 2nd level page table address.

→ what is PPN ?

↳ 0x200

→ what does the 7 mean ?

↳ last three bits are '1' they correspends to flags present, writable and user. Here, page is present, writable and can be accessed by all.

(gdb) print/x (0x80107beb >> 12) & 0xfff

$6 = 0x107

(gdb) print/x ((int*) 0x114000)[0x107]

$12 = 0x107001

→ what is this ?

↳ This is starting address of the page containing memory location we want to access.

→ why 1 in the low bits ?

↳ It indicates that the page is present.

→ Why did physical address work in gdb ?

↳ because paging is not enabled yet. (flag in cr3 register.)

After switchkvm loads kpgdir into %cr3

(gdb) x/i 0x107beb

0x107beb : can not access memory at address 0x107beb

→ Why ?

↳ switchkvm enables paging by setting the flag in cr3 register. Now, all addresses ~~need~~ core virtual and can be accessed via paging only.

⇒ Assignment : Addressing

Suppose you wanted bootmain() to load the kernel at 0x80200000 instead of 0x80100000, and you did it by modifying bootmain() to add 0x100000 to va/pa of each ELF section. Something would go wrong. What ?

↳ bootmain() calls entryfunction which maps 0 to 4MB and 2GB to 2GB+4MB of va space to 0 to 4MB of pa space. After this pa > 4 MB can't be accessed. If kernel is loaded at 2GB+2MB instead of 2GB+1M there might be loss of information.

⇒ Assignment : Traps

→ Is it possible to have two "context" structures and one "trapframe" structure on the kstack ? If so, when ? If not, why not ?

↳ NO. kstack can not have two "context" structures. Because, after pushing context on a kstack, you either pop the same context or ~~switch~~ pop context from some other process' kstack. ~~Therefore~~, ~~there~~ Interrupts are also disabled during this operation. ~~Therefore,~~

→ Is it possible to have two trapframe structures and one context structure on the stack. If so, when ? If not, why not ?

↳ YES ; When there is an interrupt in user process, it starts running interrupt handler in kernel mode after storing trapframe on kstack. While running in kernel mode, if a timer interrupt occurs then kernel values will be stored and switch to another process. In this case, there are two trapframes (user trapframe and kernel trapframe) and a context structure on the kstack.

→ Is it possible to have more than three sets of saved registers in the stack? If so, when? If not, why not?

↳ NO. for xv6, kstack can contain maximum two trapframes only. It is ensured by using locks and disabling external interrupts, such that kernel instructions are not interrupted in the middle.

⇒ Assignment: Context switching

→ where is the stack that sched() executes on?

↳ sched() calls swtch, it executes on kstack of currently running process.

→ where is the stack that scheduler executes on?

↳ scheduler doesn't have its own address space or stack. It runs on the same stack on which main() function is running.

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

↳ Yes, swtch() will return, but not immediately. It will return when the process is resumed next time.

→ Could switch do less work and still be correct? Could we reduce the size of a struct context? Provide concrete examples if yes, or argue why not.

↳ NO. To ensure correctness, callee-save registers need to be stored to maintain gcc calling conventions. Therefore, size of struct context can't be ~~d~~ reduced and switch can't do less work.

→ What is the four-character pattern?

↳ Output: acbadcbadcbad...

repeating pattern: "cbad"

→ The very first characters are ac. Why does this happen?

↳ While setting up processor, scheduler is ~~initialized~~ initialized for the first time. therefore 'a' is first character.