MAHAGURU Page No.
Date

OS- HW3

2016EE10442

Paging

Q. On x86 using 4KB pages only, map VA 0x80100000 to PA 0x 00100000 with read only permissions.

→ Page Directory offset: 512 (0x 200)
PTE_P = 1 PTE_W = 0 PTE_U = 0
It will point to entry in a page table.

Page Table Offset: 256 (0x 100)
PTE_P = 1, PTE_W = 0, PTE_U = 0
It will point 0x 00100000 physical address

Page Table Reload:

Q: (gdb) print/x kpgdir[0]
Why is this zero?

→ The setupkvm() function has mapped the virtual address above KERN_BASE into kpgdir[]. This results in the mapped entries beginning from 512th [0x200] index. All entries before are unmapped & hence kpgdir[0] is zero.

MAHAGURU Page No.
Date / /

Q. How would we translate 0x80107bcb to a physical address?

→ Subtract KERN-BASE (0x80000000) from the virtual address.

Q. (gdb) print/x kpgdir[0x200]
$6 = 0x114007

Q: What is this?

→ This the page directory entry corresponding to the page which contains the physical address mapped to the virtual address 0x 80107bcb

Q. What is the PPN?

→ 0x114 is the PPN

Q. What does the 7 mean

→ It means the page is present [PTE_P = 1], page is writable [PTE_W = 1], and page is accessed in ~~supervisor mode~~ ring 3 [PTE_U = 1]

(gdb) print/x ((int*) 0x114000) [0x107]
$12 = 0x 107 001

Q. What is this?

→ It is the page table entry & indicates the location of starting address of the page containing VA 0x8010...

Q.

Q: Why 1 in the low bits?
→ It indicates that the page is present (PTE_P = 1)

Q: Why did the physical address work in the gdb?
→ SetupKvm() is used to set up the page tables but paging is not yet enabled. Hence, there is an identity mapping from VA to PA. Thus, physical address can be accessed directly via gdb.

Q: (gdb) x/i 0x107beb
0x107beb: cannot access memory at address 0x107b
Why?
→ SwitchKvm() enables paging by loading the kpgdir into %cr3. After this point, all address will be treated as virtual & be decoded using the paging mechanism which maps address above KERNBASE. Hence, we cannot access 0x107beb directly.

MAHAGURU Page No.
Date

Traps

Q. Is it possible to have two "context" structures & one "trapframe" structure of the kstack?

→ No, two context structures are not possible on the same kstack, as by the time "context" is pushed, switch would have already occurred. Moreover, interrupts are disabled while pushing context.

Q. Is it possible to have two trapframe structures & one context structure on the kstack?

→ Yes, it is possible when a user process is interrupted & it starts running instructions from the kstack. While running in the kernel mode, if a timer interrupt occurs, kernel will store context & switch to another process. Here, two trapframes exist, one for user system call & another one for kernel mode interrupt, but only a single context structure is there.

Q. Is it possible to have more than 3 sets of saved registers in the kstack? If so when?

→ No, in xv6 there can be a maximum of two trapframes possible & hence only 2 sets of saved registers. It is ensured using locks & disabling external interrupts. However, in theory, ~~two~~ three sets of registers are possible in an OS where external interrupts are not disabled.

MAHAGURU Page No.
Date

Context Switching

Q. Where is the stack that sched() executes on?
→ The kstack of the running process

Q. Where is the stack that scheduler() executes on?
→ The same stack as that of main() function in main.c.

Q. When sched() calls switch(), does that call to switch ever return? If so, when.

→ Yes, when the scheduler schedules the process to execute again, the control returns back to sched().

Q. Could switch() do less work & still be correct? Could we reduce the size of struct context?

→ No, the size of struct context can't be reduce & neither could switch do less work as they are both required to correctly ~~save~~ store the callee-saved registers & ~~[illegible]~~ ensure gcc calling conventions are maintained correctly.

Q. What is the 4 character pattern?
→ cbad
Q. The very first characters are ac. Why?
→ Initially mpmain() calls the scheduler the first time where 'a' is printed & the first process begins to execute. when it sleeps, it goes to sched() for the 1st time & 'c' is outp