Question 1, HW-3 Paging

SALIL CHANDRA
2016EE30246

Ans. For page directory:

① Offset = 0x200 = 512th entry

② Entry value = 0x00200005 → 0101

Can be any Physical address of any allocated page

0 → User
1 → Read only
0 → Present
1 → Present

For page table (2nd level):

① Offset = 0x100 = 256th entry

② Entry value = 0x00100005 → 0101

PTE_U: 0 → User
PTE_W: 1 → Read only
PTE_P: 0 → Write back
PTE_P: 1 → Present

Question 2. Page Table Reload

① (gdb) print /x kpgdir[0] · Why is this zero?

Ans · All entries in kpgdir are initialised to zero in setupkvm() (line 1825: memset(pgdir, 0, PGSIZE);) Then mappings are made only starting from KERNBASE. Therefore entries below KERNBASE are still 0 (unmapped) and are to be used for user address space mapping later.

② · How would we translate 0x80107beb to a physical address?

Ans. First 10 bits are used to offset into PD, next 10 bits offset into PT. Final 12 bits are offset into the physical mapped page in the physical address space.

Since setupkvm not called yet, PA = VA - KERNBASE → 0x80000000

③ (gdb) print/x 0x80107beb >> 22
$4 = 0x200
(gdb) print/x kpgdir[0x200]
$6 = 0x114007

(i) What is this?

Ans. $4 = 0x200 are the first 10 bits of the virtual address that index into the page directory.

$6 = 0x114007 is the page directory entry at offset 0x200 (ie, 512th entry)

(ii) What is the PPN?

PPN for page table (2nd level page table page) is 0x114000.

(iii) What does 7 mean?

Ans. 7 (0111) is the last 4 bits of the PDE, which implies the page is Write back, User accessible, Writable and Present.

(4) (gdb) print /x (0x80107beb >> 12) & 0x3ff
$6 = 0x107
(gdb) print /x ((int*) 0x114000) [0x107]
$12 = 0x107001

Ans $6 = 0x107 is the offset into the 2nd level page table (263rd entry).

$12 = 0x107001 is the 2nd level page table entry at offset 0x107. (0x107000 is the PPN, 001 are flags)

(ii) Why 1 in the low bits?

Ans 1 in low bit indicates that the page is present. (0th bit is the PTE_P flag)

(5) Why did the physical address work in gdb?

Ans. kpgdir is still not being used because switchkvm() has not been called yet. Since presently, the virtual addresses from (0 to 4MB) and (Kernbase to Kernbase + 4MB) are identically mapped to physical addresses 0 to 4MB (due to entrypgdir); we can still access the physical address directly.

(6). Cannot access memory at address 0x107beb.
why?.

thus After setupkvm() is called, the paging hardware starts to use the kpgdir. In kpgdir, only addresses above kernbase are mapped. Since 0x107beb is an address below kernbase, it cannot be accessed.

Question 3 Addressing.

Ans. The addresses may overflow above 4MB. Since entryhgdir only maps VA from (0 to 4MB) and (KBase to KBase+4MB) to (0 to 4MB) in the physical address space, (assuming XV6 fits inside 4MB space), if we offset all addresses by 1MB, they may overflow the 4MB cap, and therefore will not be valid while using entryhgdir.

Question 4. Traps

① Explain a situation when a suspended process will have 3 sets of saved regs in its k-stack.

Ans. Suppose a process makes a system call (Therefore User registers are saved in trapframe → ①). When trap() is returning, an external timer interrupt occurs and the kernel side registers for the process get saved in another trapframe later on the same stack. ~~Sup~~ Now the timer interrupt will cause context s/w, therefore kernel registers will get ~~to~~ saved again in 'struct context'

② Is it possible to have 2 'context' structs and one 'trapframe struct' on k-stack.

Ans No, there can't be 2 context structures on one same k-stack because the ~~same~~ interrupts are disabled while switching, therefore no timer interrupt (which causes struct context to be pushed) doesn't appear.

③ Is it possible to have 2 trapframes and one context structure on the same k-stack.

Ans Yes. Exactly same scenario as in ①.

i.e, the chronology of system call → Timer interrupt (while returning from sys call) → context switch.

④ · Is it possible to have ~~not~~ greater than 3 sets of saved regs on the K-stack?

Ans No. Since there can be a maximum of 2 trapframes at a time on the stack (system call followed by external interrupt). There cannot be a 3rd trapframe because interrupts are disabled while handling the external interrupt.

Also, a maximum of 1 context structure can appear on the K stack at a time (as in ②).

Therefore >3 sets of saved regs are not possible.

Ques 5. Context Switching

(1). Where is the stack that sched() executes on?

Ans. sched() executes on the currently running process's k-stack.

(2). Where is the stack that scheduler() executes on?

Ans. scheduler() executes on the main stack (where the main() function is executing).
However in certain instances of Process model, the scheduler can have a separate stack, but not in xV6.

(3) When sched() calls switch(), does that call to switch() ever returns? If so, when?

Ans. Yes, switch() function returns when the process is again 'context switched in', which may not ~~not~~ happen ~~at a too~~ immediately, but at a later context switch.

(4). Could switch() do less work & still be correct? Could we reduce size of struct context? Provide examples

Ans. No, ~~size of~~ switch cannot do ~~too~~ less work and we cannot reduce the size of struct context.
This is because switch() only saves the callee-save registers (according to gcc calling convention), which is the bare minimum to ensure correctness.
~~Struct context must store all registers.~~

(5). What is the four char pattern?

Ans. cbad
(O/P is "a cbad cbad ...")

(6). The very first characters are 'ac'. Why?

Ans.