HW3 COL331

Shivanshu Verma.
2015CH70186.

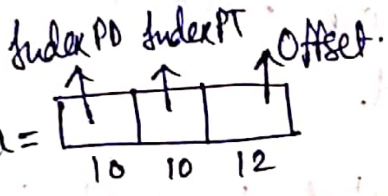
Assignment: Paging (2 marks)

Vaddr = ~~0x80100000~~ 0x80100000 , paddr = 0x00100000

Vaddr = $8 \times 16^7 + 16^5$ = $2^{20} + 2^{31}$ = $1(0)_{10}1(0)_{20}$ in binary.

{ $(0)_{10}$ = ten zeroes i.e. 0000000000 }

and va =



⇒. Page Directory offset = first 10 bits of vaddr
= $1(0)_9$

and Page Table Index = next 10 bits of vaddr
= $01(0)_8$.

~~and~~ Page Directory offset = $1(0)_9 = 2^9$ = 512th entry.
and this will contain the physical address of page table. (PPN + flags)

Page Table offset = $01(0)_8$ = 2^8 = 256th entry in page table

this will contain the physical address of page ~~[illegible]~~ ~~paddr~~ ~~PPN~~ which is equal to paddr here

Page offset = $(0)_{12}$ = 0.

permissions → read only, present,

⇒. 0x00100001
↳ present flag

Assignment : Page Table 'Reload'

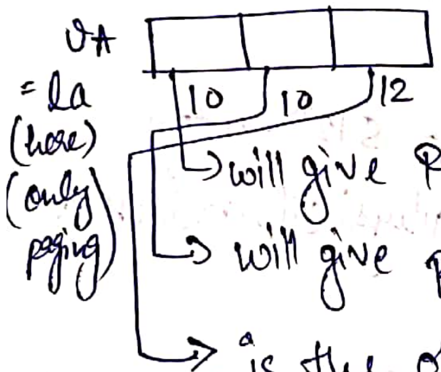
Why is kpgdir[0] zero?

Ans: kpgdir = setupkvm() → this func^n only maps the space above KERNBASE (2GB) not the user space.

That is why, not only 0^th index but all the entries from 0 - 511 (first 512 entries) →(till 2GB) were zero and from 512 index onwards these are mapped.

How would you translate 0x80107beb to a physical address?

Ans: we'll use page table and page directory to get the suitable physical address.



→ will give page directory entry which is pointed by CR3

→ will give page table entry where page table is pointed by PDE

→ is the offset at page which was pointed by PTE.

```
(gdb) print/x kpgdir[0x200]
$6 = 0x3fe007
```

Q What is this?   (PPN + flags)

Ans. Physical address of page table plus some flags. (PDE)

Q What is PPN?

first 20 bits of PDE i.e. 0x3fe007 >> 12
= 0x3fe.

Th . reduce th . . [illegible] . . .

Q What does 7 mean?

Ans → $(7)_{16} \rightarrow (0111)_2$

$$\begin{tikzcd} & (0111)_2 \arrow[dl] \arrow[d] \arrow[dr] & \\ \text{uses flag} & \text{writable flag} & \text{Present flag} \end{tikzcd}$$

(gdb) print /x ((int *)0x3fe000)[0x106]
$12 = 0x106001

Q What is this?

Ans → ~~Physical address of the page + flags~~
(PPN + flags) → of page to which offset will be added.

Q Why 1 in the low bits?

1 is the present flag, showing that the page is avaible.

Q Why did the physical address work in gdb?

B/C we are still in entrypgdir and did not load kpgdir in cr3 yet.

Q Why it won't work after calling switchkvm()?

switchkvm() will load the kpgdir into cr3 or switch to kpgdir from entrypgdir and kpgdir does not have maping to its own address.

Assignment : Addressing.

Ans: By adding 0x100000 to va/pa of each elf section, we are only changing the elf sections. But as functions and variables are named by addresses, the already compiled code may have internal reference to pointers having predefined locations and shifting addresses may point to wrong locations which will cause errors.

Assignment : Traps.

Q Situation in which a suspended process will have three set of saved registers in its kstack.

Ans If a timer interrupt occured in the middle of kernel execution. Still have 2 trapframes (for kernel & user) and 1 context. set of registers.

Q 2 context and 1 trapframe

No, not possible. Context structure ~~stores~~ is used when we switch and given 2 context, where to start execution from. No, not possible.

Q 2 trapframes and 1 context.

Yes, in the situation described above where the ~~kernel~~ interrupt occurs in the middle of kernel execution.

(Q More than 3 set of saved registers
No, to prevent kstack overflow, we disable interrupts while executing a handler ensuring atmost 2 trapframes in kernel and therefore, (2 trapframe, 1 context), we can't have more than 3 sets of saved registers.

Assignment: Context Switching

Q Where is the stack that sched() executes on?
→ Sched( ) executes on process's kstack which is located from kernel heap's space..

Q: Where is the stack that scheduler() executes on?
Scheduler() executes on its own kstack or say CPU's kstack which is initialized during kernel initialization and this does not belong to any process. It doesn't have a pgdir associated to it.

Q When sched() calls switch(), does that call to switch() ever return? when?
Sched() executes on process's kstack and when it calls switch(), that will switch to scheduler kstack. (say P1)
Now, whenever scheduler (after some process switches) picks that process (P1) again, it will return.
(and switches to P1)

Scanned with CamScanner

[illegible]

Q What is the four character pattern?

→ acbad cbad cbad cbad ......

Q Why ac as very first characters?

When the kernel intializes itself, mpmain() function calls the schedular() and it enters schedular() at first, therefore, a in the very begining.

Now after a,

1. cprintf('a');

we switch → 2. Swtch(&cpu->schedular, &proc->context);
to process kstack. 3. cprintf("b");

Then some process calls shed(), therefore, we see "c" as second character on screen.

so we switch 4. cprintf("c");
to schedular → 5. Swtch(~~&cpu->schedular, &pro~~ &proc->context, &cpu->schedular);
again. 6. cprintf("d");

↳ In schedular, we'll continue from where we left, i.e. line 3: cprintf("b"), and as there's a infinite for loop in schedular(), we again come to line 1: cprintf("a"), switch from here, now whenever sched() is called, we continue from line 6, i.e. cprintf("d") and this continues

⇒ pattern is a cbad cbad cbad ...... so on.

Q Could switch do less work and still be correct?
Can we reduce the size of struct context?

In general, as per the gcc calling conventions, switch requires to save the callee saved registers and doing less work, i.e. not saving these registers can only work in case the register values are not changed, which can't be true for all the function calls.

We can ~~definitely~~ make the context to point to the last register value i.e. edi and make struct contain only this value but save the rest of registers value in kstack. This way, we reduced context struct size but followed gcc calling conventions too.

Scanned with CamScanner