HW-3

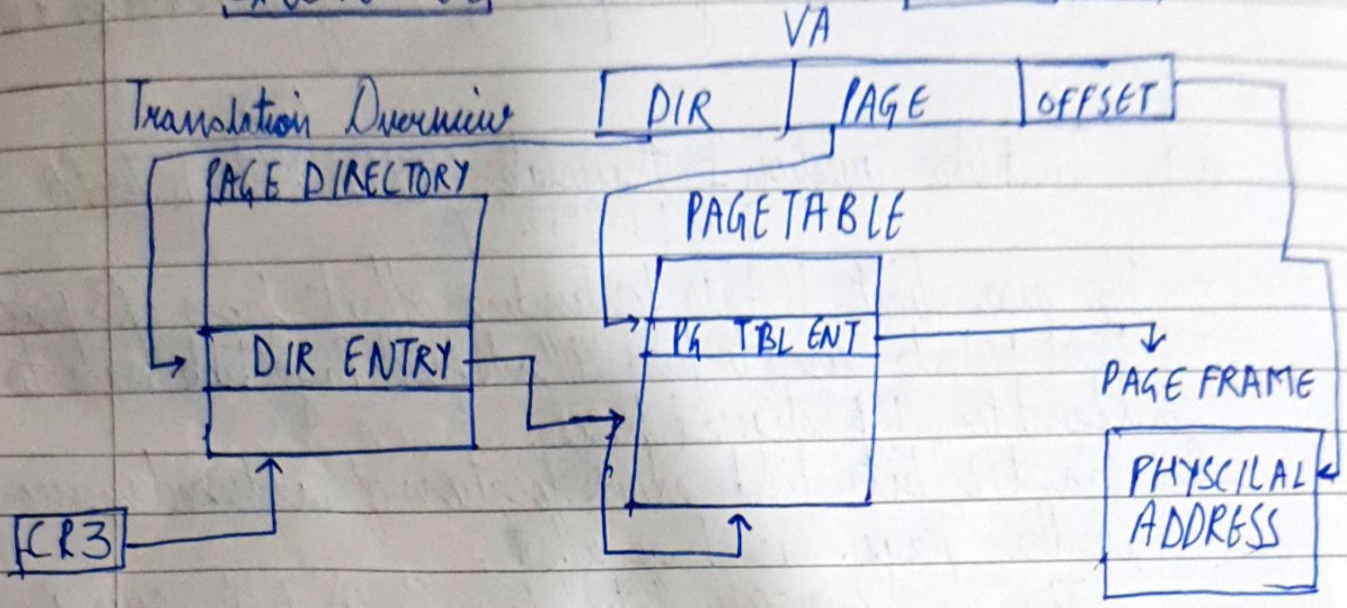
PARTH CHOPRA, 2016MEC1021.

OPERATING SYSTEM

4) x86 4KB pages (Paging)

VA → PA

0x80700000 → 0x00100000



0x80100000

→ 0x200 PAGE DIR OFFSET

0x200 PAGETABLE OFFSET

0x000 Physical Page offset.

ENTRY at in PAGE DIRECTORY (CR3 + 0x200 * sizeof(PTE))

4 bytes.

MODIFIED TO. not specified 12 FLAG BITS.

0x00384 | 0 0 1

Read write bit | Present Bit

Entry at PAGETABLE 0x00384 MODIFIED TO.

0x00384 + 0x200 * 4 bytes to

0x00100 | 001

A2 PAGE TABLE RELOAD.

(i) Virtual Address starts from 0 to $2^{31}-1$ (Incase of 32 bit machine) when we addressing 1st instruction we access it by. kpdir[0] This gives returns 0

(ii) Just like question to translate 0x80107beb. To PA

[10 | 10 | 12.]

- We use first 10 bits for finding offset from CR3 points
- Next 10 bits used to find offset from Page table base to find point to 4kB aligned page.
- Last 12 bits find exact location of addressed memory in the page, ie page offset,

This we can translate from P VA to PA once Paging is enabled and pages being 4kB.

(iii) 0x114007.

- This is page table directory entry which is used to get the address of the page table. where ~~for~~ this 0x80107beb =

(iv) 0x114. is PPN.

(v) 7. in the end corresponds to the flags for the page table as a whole.

7 = 1 1 1

- 1 → presence bit
- 1 → Read/writable bit
- 1 → user accessible bit

Here all three bits are set. which means page table present it readable/writable. user can access the page However all this is provided, the specific page also has all these bits set.

(vii) This is page table entry which is used to get to the page containing the data associated with VA. First 20 bits tell us about the address of page. as all pages are 4kb aligned (4kb in size so in continuous memory, so 4kb apart) last 12 bits used as flags associated with the page.

(viii) Here 1 is set in lowest bits indicates the flags look like 0 0 1, which indicates that this page is present. Moreover is read only mode. It takes kernel rig to access the page. User can't access the page.

(ix) Physical Address worked in ~~qdb~~. because x86 uses that structure to adress to PA and VA hence. PA also worked. ~~hence and so we can use~~ here.

(x) However, once kmalloc ~~data~~. calls setupkvm to create page table to start using it switch kvm loads ~~page~~ Pgdir into CR3.

Now 0x107beb can't be accessed in user mode because the flag is set such that only kernel will be able to access it as we are using paging to ~~read~~ the address.

A3. Addressing

This will be wrong because once the kernel executing starts it will set up paging hardware to map virtual address 0x80100000 to physical address starting at 0x00100000; kernel assumes that there is physical memory at lower address. At this stage boot process, paging is not enabled. Instead Kernel.ld specifies that ELF starts at 0x00100000 which will cause bootloader to write at bit values hence when paging hardware will eventually point to these crashing the system.

A4 Traps.

(i) It is ~~not possible~~ for a suspended process k-stack to have two "context" structure because once a system is context switched out ~~only~~ it returns to running state only ~~when~~ using the context structure. Moreover, a process can be context switched out once. Additionaly, context structure is last entry on the process k stack.

(ii) Yes, it is possible to ~~two~~ have two trap frames structure along with context structure on k stack, this can happen if for a process the first trap occurs due to software interrupt / exceptions/ signal and second due to external interrupt received while executing in the kernel. While handling this it can be context switched out. resulting in two trap structure and one context structure.

(iii) No. In a process k stack can have atmax 1 context structure saved register and '2' trap structure saved registers. So can't have more than 3

A5

(i) process k-stack sched() executes on.

(ii) Each CPU scheduler has its own stack.

(iii) When sched() calls switch(), this switch function does the job of switching between two contexts. The switch() returns to a different process. At a later stage again switch would be called it might return to thisSched() and then this process would be resumed again caller doesn't get to see returns.

(iv) Yes, switch could do less work and even then be correct. when timer interrupt occurs process yield() which calls sched() which inturn calls sched switch(), assuming GCC conventions, already on the k-stack there are useful values of registers saved, so really it can work with that. no need to save and pop values can just switch(esp)(esp)

(v) a c d b (c a b d) c a b d — In one core this will remain consistent like this.

(vi) This happens because first process created doesn't actually context switches out, it doesn't call switch() the k-stack is initial made as if the init process was ready to run without it really ever being contexted switched out Hence, very first time 'iret' appear we start a Single CPU system.