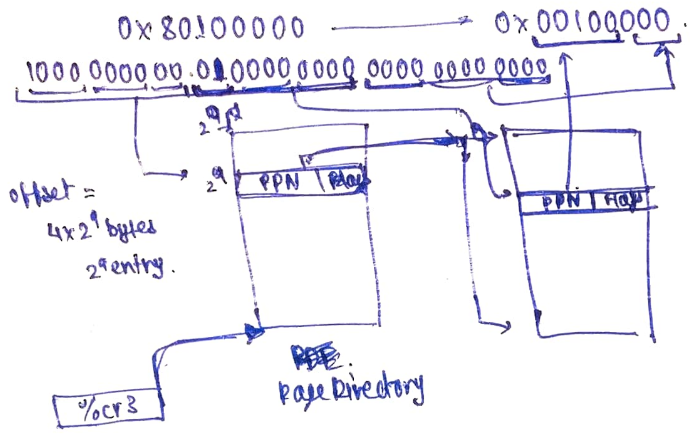
COL331 - OPERATING SYSTEMS

HOMEWORK-3

SHIKHAR GOEL
2016CH10089

Assignment: Paging.



Page directory offset = 4×2^9 bytes,

Page table offset = 4×2^8 bytes.

page directory entry value = PPN of page table.
first 20 bits
then 12 bits → flags.
Page table readable

page table entry value = 0x00100
first-20 bits.
then 12 bits – flags.

Assignment: Page Table Reload.

(gdb) break kvmalloc

(gdb) continue

(gdb) next

(gdb) next   # execution reaches the start of "switchkvm()" line.

(gdb) print/x kpgdir[0].

→ This is 0 as the page directory entry points to 0x0000 in physical memory. ~~page directory~~ the data in memory starts and has where the ~~page directory~~ to be paged

(gdb) x/i kvmalloc

→ we can translate 0x80107beb to a physical address by using & operator with 0x0ffffff. this will give PA = VA – 2GB, we'll get 0x00107beb

(gdb) x/i $eip
0x80107b/b: call 0x80107c02

(gdb) print/x 0x80107beb>>22

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

→ Page directory at 200, first 10 bits of address, gives us the page table starting point

→ PPN is 0x114, and 7 means page is present, read/writable, user accessible

(gdb) print/x (0x80107beb >>12)&0xfff. $6 = 0x107

(gdb) print/x ((int*)0x114000)[0x107], $12 = 0x107001

value of entry at page table, 0x107000 is this the physical page PPN and 1 means the page is present

(gdb) print/x 0x107000 + 0xbeb., $13 = 0x107beb.

physical address works in GDB as paging is not yet enabled by hardware, and the lower memory's are filled, and mapped.

, after switchkvm loads kpgdir into cr3 register, paging is enabled & 0x107beb is not valid now.

Assignment: Addressing

If we load the kernel at 0x80200000 instead of 0x80100000, then our kernel code and data would start from after 2 mb in RAM and the space 1MB to 2MB would be empty.

If the kernel code and data is ~~.~~ more than 2MB then it will throw an error, there will be no space for heap to define page directory.

Assignment: Traps

~~...~~ If a process while running transfer the control to kernel and then an external interrupt occurs which ~~gives~~ ~~contro~~ ~~transfer control to handler~~ ~~is~~ then handled by handler function. ~~..~~ If the handler functions ~~...~~ gives control to scheduler function and changes ~~from~~ to different process then three sets of registers are saved. ∴ 2 trapframe – 1 for ~~p~~ user process. 1 for kernel, and 1 ~~..~~ context structure.

- It is not possible to have 'context' structures on the k-stack as after the context switch happens the control is given to ~~..~~ different process by scheduler. context-switch and interrupt handler cannot happen switching context
- Yes it is possible, a user process gives control to kernel. ~~in~~ then interrupt occurs and scheduler ~~gives control to~~ switches context to different process.
- It is not possible to have more than 3 set of saved register as after second trapframe ~~..~~, ~~the~~ it is handled by ~~..~~ ~~..~~ interrupt handler function which temporarily disables interrupts. And two context switches cannot happen as the process ~~..~~ gives control to different process after context switch.

Assignment: context switching

→ sched() executes on the processe's ~~..~~ kernel stack in user space

→ scheduler() executes on the ~~.....~~ ~~& ..~~ main kernel stack in kernel space

→ switch() function eventually returns when the scheduler gives the control back to the process.

→ switch cannot do less work as it only saves the callee save registers which are bare minimum necessary for completion

→ four-character pattern is 'adcb'

→ the very first characters are 'ac' as the processes have to be initialized once before the context switch and the scheduler function ~~...~~ doesn't return until the new process is created.