Handling Traps

Transition from user to kernel mode:

1. System call
2. H/W interrupt
3. User prog doing illegal work(Exception)

Action needed during H/W interrupt :

- change to kernel mode and switch to kernel stack
- kernel to work with devices, if needed
- kernel need to understand devices

Actions needed on a trap :

- saves proc. Reg (context) for future use
- sets up system to run kernel code (kernel context) on kernel stack
- start kernel in appropriate place (sys. Call, Interr hanadler)
- kernel gets all info related to event by **H/W**
    - block I/O done
    - which sys call
    - which proc causeed exception
    - type
    - args to the system call

Privilege Level :
4 protection level (0 – 3)
0 – most privilege
3 – least privilege

In practice, many OS use 2 level
0 : kernel mode
3 : user mode

Current privilege level of x86 instr execution in %cs register in CPL field
CPL : Curent privilege level

Changes automatically on

- int instr
- H/W interr
- exception

Changes back when :**iret**

in xv6 INT 64 : actual system calls
in linux hexadecimal 80
INT 10 : makes 10th h/w interrupt
s/w interrupt can be used to create h/w interrupt

Interrupt Descriptor Table (IDT):
- **IDT** define interr handlers
- 256 entries
    - given each %cs and %eip to handle corres. interr
- **interrupt 0 – 31** are defined for s/w exceptions like **divide errors or attempt to access invalid memoery addr.**

- xv6 maps 32 h/w interrupt to range 32 -63
- intrr 64 is used for sys. Call

Entries in ICTR : called as Gates

when kernel has to set up **idt table** it has to set up gatedesc

gatedesc form data part of kernel . Loaded when kernel loaded in bootmain
Done in tvinit()

- loops 256 times (size of idt table)
- call macro SETGATE
    - addr of idt entry we want to set
    - istrap
    - set cs to 1
    - actual addr of function to be executed
    - dpl : 0 (kernel mode)

In second call,
T_SYSCALL = 64
entry is overwritten system call (64) is called,
istrap =1
set cs
set function
dpl_user (user mode)

in any interr, code jumos to entries in vectors.S

tvinit sets up idt array, then h/w does rest of job. Jumps to location specified in file

tvinit is called in main.c //kernel initializer function

alltraps:
    pushes all segments
    then calls trap

CPU during INT instr/interrupt:

1. Fetch nth descriptor from IDT, where n = arg of INT
2. Check if CPL in %cs <= DPL
    current privilege level <= descriptor privilege level
    We can't go to higher level
3. H/w automatically saves cur stack point %esp and stack segment %ss in internel reg (temporary)
    Only if going into privilege level which is lower (or same**)**
    when going from user to kernel we have to save them, so that we go back to user we have to use values of saved regs.

When we have to run kernel(from proc, h/w interrupt, interr), process gets kernel stack from **task segment descriptor.**
%ss, %esp from task segment descriptor.
## stack --> kernel stack
TS desc. is on GDT given by TR reg.

Check privilege level
1. push %ss (Optionally) only if privilege 3 to 0
2. push %esp (Optionally)(also changes es,esp using TSS) by **CPU**
    have to be used when rescheduled in future
3. Push %eflags
4. Push %cs
5. Push %eip
    All old context of proc. All done by single INT instrr. No h/w involved
6. Clear the IF (interrupt) in %eflag **but only on interrupt . Not from proc.**
7. Set up **new %cs and %eip to values in descriptor** (i.e. vector 1 to ....)

After INT job is done:
idt is set

jump to 64th (sys call)
    on kernel stack ss, esp, cs, eip are already pushed by h/w
    pushing 0 (**error number, for error code**), 64
    then run **alltraps from trapasm.S**

## Alltraps
now **%ds,%es,%fs , %gs , all gen. Purpose reg** are pushed

Now stack has:
    ss, esp, cs, eip,0 (**error number, for error code**), 64,**%ds,%es,%fs , %gs, eax,ecx,edx,ebx,oesp,ebp,esi,edi**
This is struct **Trapframe**
Kernel stack contains trapframe

Trapframe is a part of kernel stack

Now ds and es are loaded with SEG_KDATA

Pushes curr value of esp
Then call trap
esp is arg to trap. Trapframe points to last addr after push to stack
Now this trapframe points to all relevent info.

## trap()
args : trapframe
in alltraps:
    before “call trap” there was “push %esp”
    stack had trapframe

# when function is called, the stack contains args in reverse ord. (only 1 arg)

## Padding (2 byte) is added to that 2 byte values are pushed to 4 byte values in struct

in trap

if sys_call (T_SYSCALL), then syscall() is used.
 Switch based on trapno

Based on number, called interrupts

Trap has a switch (uses trapno which was pushed in trapframe by vectors)

Depending type of trap, calls interr handler
 Timer:wakeup,
 IDE – disk interrupt : ideintr,
 Keyboard: kbdintr,
 COM1: Uatrintr
 Timer : calls yield --> call sched
  If proc is killed --> calls exit()

When trap returns: (only not when exit)
- goes to alltraps
- popped the esp that was pushed
- run **trapret**
- **popal :** pops eax,edx,ecx,ebx,oesp,ebp,esi,edi
- **pop all seg :** gs,fs,es,ds
- add 8 to spe: removes 0 error code, 64
- iret : opp to int instr. Automatically pops 5 values which have pushed by int
  ss,esp,eflags,cs,eip
- stack switched to application , proc. Stack with old values of seg.

Interrupt only at end of instruction , NOT DURING.

When we are in assembly, no reg are not changed / touched.

For system call codes:
in usys.S there is macro
  it create global fork
  copy $SYS in eax
  then call INT $SYS_call
  in int we go to vector.s
  we go to vector64
  we push 0 , 64
  then go to alltraps
  push ....., pushal (in pushal eax is pushed)
  call trap
  if syscall then **call syscall**