Lecture 9 —— xv6 hands-on. CS744

Lecture 10 — system calls & memory mgmt.

* Lab 3 available
  xv6-based
  system calls,
  process abstraction
  memory abstraction

* xv6-book reference

System call mechanism.

- ✓ how to invoke ?
- ✓ how to switch mode of execution ?
- how to handle arguments & return values ?
- how to identify which system call ?
- ✓ how to handle context of execution ?

(i) Hw assisted mechanism for invocation is an instruction(s) of an ISA.

x86: `int 0x80` ~ explicit s/w interrupt

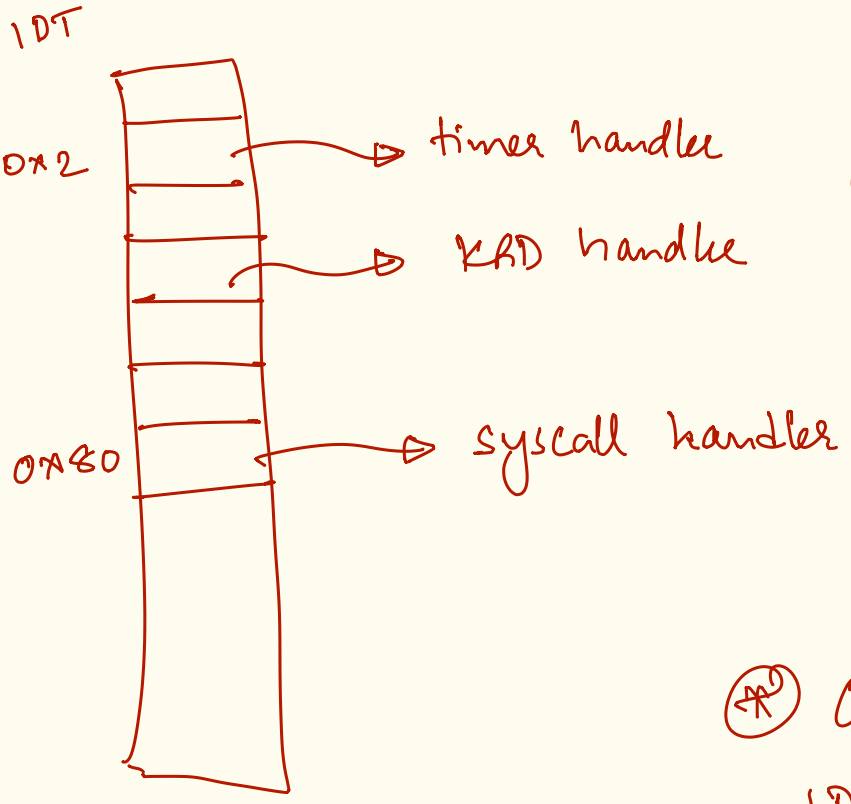
`int` — instruction; `0x80` — argument - interrupt vector

- s/w privilege level to ring 0
- (*) save context of user process on kernel stack
- s/w to using kernel stack
- jmp to interrupt handler. ~ depends on IDTR & interrupt vector

(ii) how to jump to the interrupt handler?

IDT — interrupt descriptor table

— array of function pointers (interrupt handling functions)



* how to get IDT?

IDTR

↳ register to store base addr of the IDT.

* OS maintains IDT & IDTR.

(iii) how to identify the system call and its arguments?

— via syscall number

— before ~~invoking~~ invoking system call setup syscall number & arguments in CPU regs.

eax ← syscall number

ebx, cx, dx, ... } arguments.

int 0x80

} user-space wrapper fn. sets up this invocation

Open:

```
mov eax, SYS_OPEN
{mov ebx
     ecx
     :
int 0x80
```

(iv) system call handler:

```
syscall_handler {
    fn ptr = syscall_table [ eax ];   // value of
                                      // ~~from~~ on kernel stack
    call fn ptr;
    iret;   // return from interrupt.
}
```

`iret` — instruction of the ISA.

return values are via register (eax).

(#) xv6 mechanism details.

traps.h — list of all interrupts

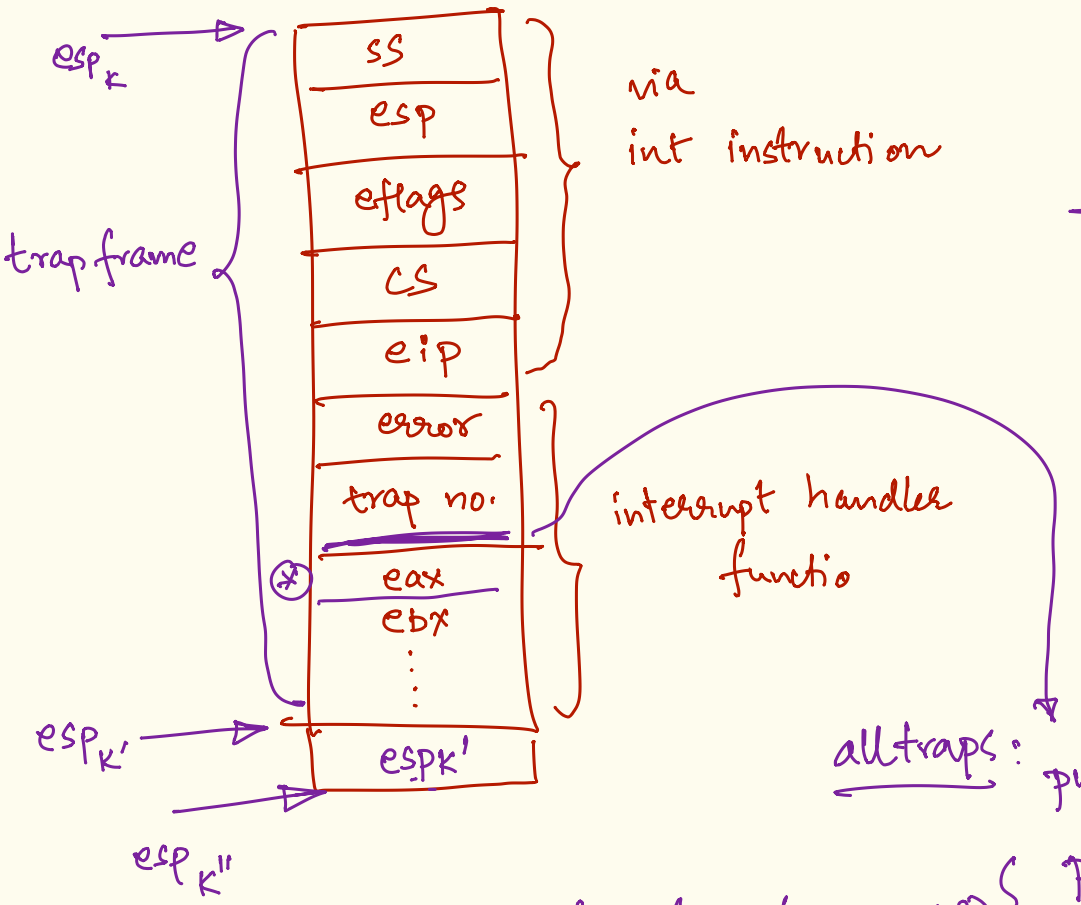
vectors.S — IDT entries for each interrupt vector

trapasm.S — alltraps ← label / address entry point for all interrupts

trap.C — generic IDT handler

syscall.c — syscall handler

⊛ xv6 (x86) syscall / interrupt invocation state



esp_K — Kernel stack ptr. info. is stored in task segment descriptor.

alltraps: push all regs.

trap (trapframe * tf) { push esp ⊛
call trap