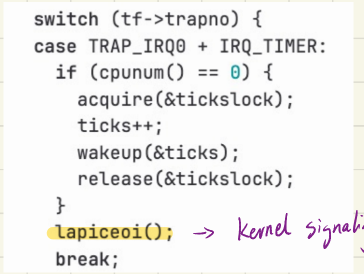
3/29/24

Mode Transfers



```
switch (tf->trapno) {
case TRAP_IRQ0 + IRQ_TIMER:
  if (cpunum() == 0) {
    acquire(&tickslock);
    ticks++;
    wakeup(&ticks);
    release(&tickslock);
  }
  lapiceoi();
  break;
```

→ kernel signaling end of interrupt

(kernel / trap.c)

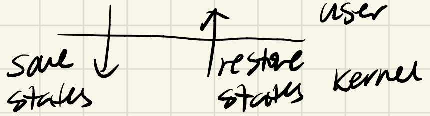
→ **interrupts** : issued by hw, high priority
- → also called external interrupts / hardware interrupts
- → serviced one at a time, kernel sends EOI when done handling
- → can preempt exception & syscall handlers

→ **exceptions** : problem caused by current instr.
- → exception behavior varies
- → what if exception occurs in an interrupt handler ?

→ **syscalls** : requested by user
- → software interrupt ("INT syscall#")

Mode Transfer Mechanisms

→ upon mode switch, hw overwrites %rip with kernel handler address

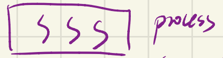


→ must save process's %rip before overwriting
→ must save user's regs. somewhere too
} process's states

→ kernel handler execution also needs a stack

→ can we save everything onto the user stack & use it for execution?

→ who has access to the user stack?

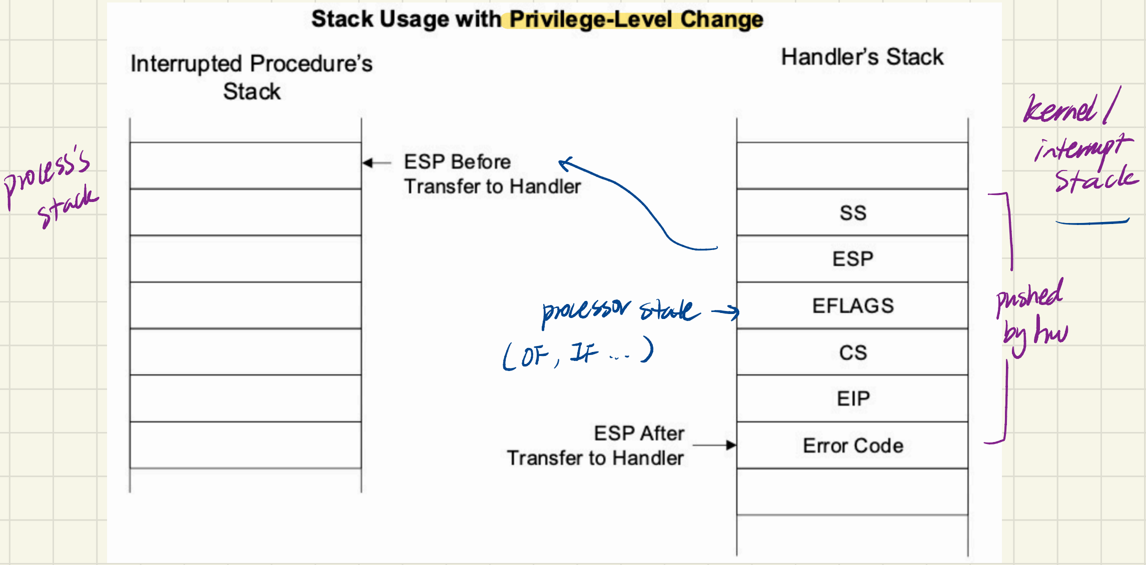


(all threads in the process have access to the stack)

→ kernel handler may push kernel data onto the stack, what else might be pushed?

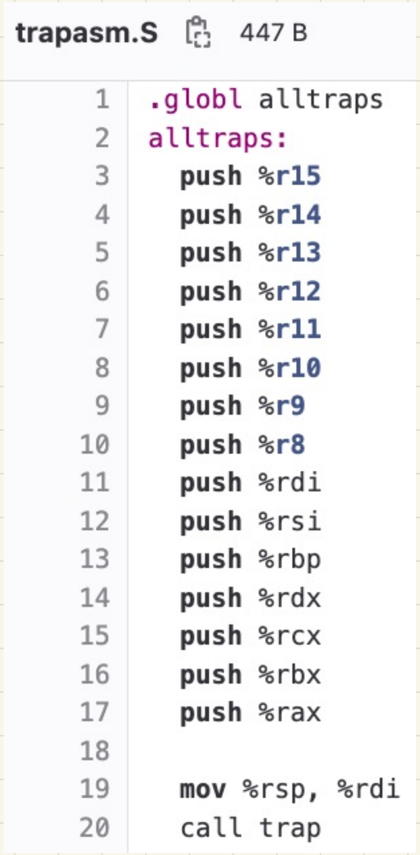
→ separate kernel stack
(allocated in kernel memory)

→ stack switch on
mode switch


Stack Usage with Privilege-Level Change
Interrupted Procedure's Stack
process's stack
ESP Before Transfer to Handler
Handler's Stack
kernel / interrupt stack
SS
ESP
EFLAGS
CS
EIP
Error Code
processor state
(OF, IF ...)
pushed by hw
ESP After Transfer to Handler

How many kernel stacks are there?

→ one per process

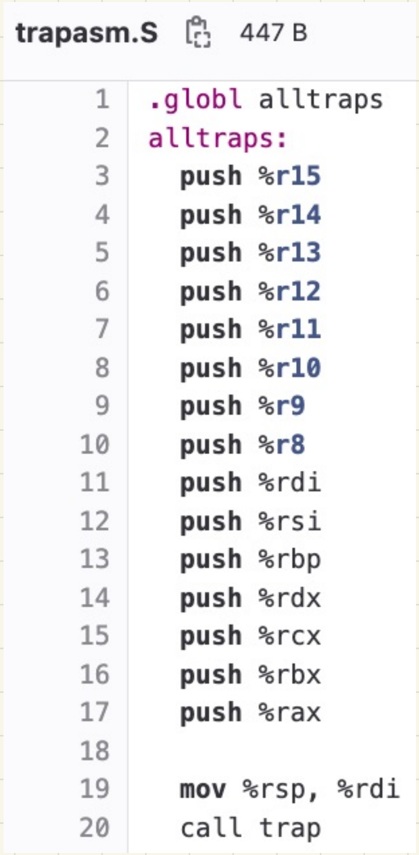


trapasm.S 447 B

```
.globl alltraps
alltraps:
  push %r15
  push %r14
  push %r13
  push %r12
  push %r11
  push %r10
  push %r9
  push %r8
  push %rdi
  push %rsi
  push %rbp
  push %rdx
  push %rcx
  push %rbx
  push %rax

  mov %rsp, %rdi
  call trap
```

Kernel handler pushes the rest of regs.

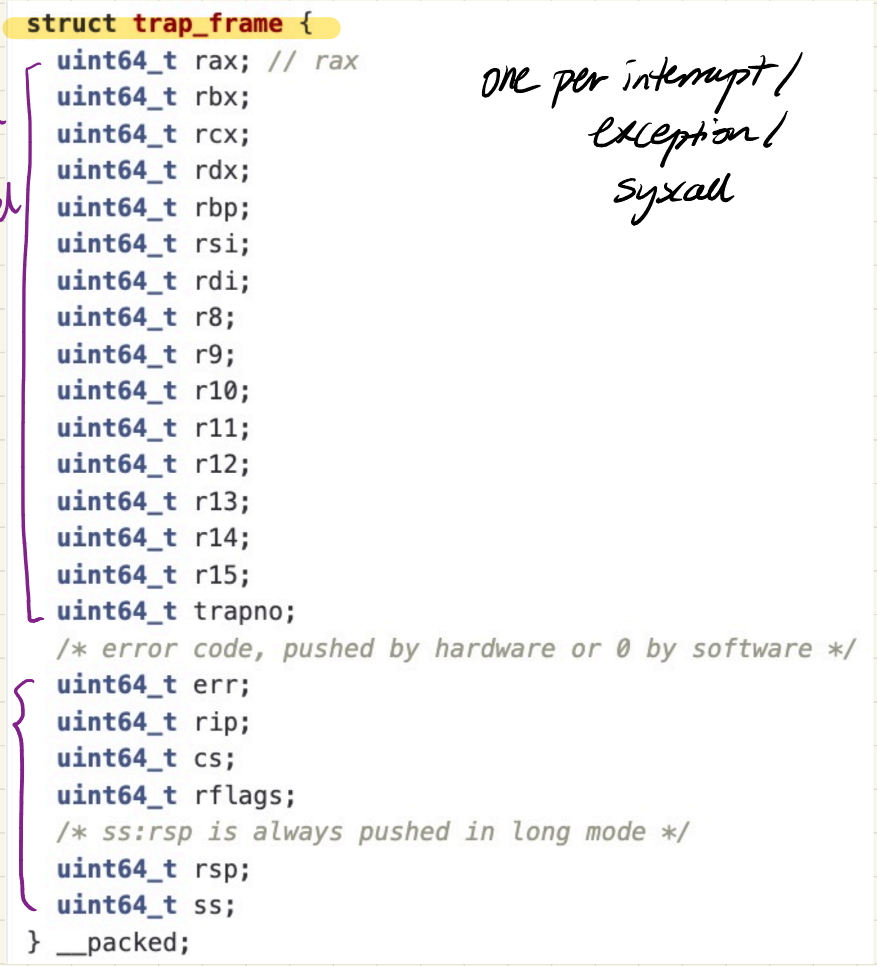


trapasm.S 447 B

```
.globl alltraps
alltraps:
  push %r15
  push %r14
  push %r13
  push %r12
  push %r11
  push %r10
  push %r9
  push %r8
  push %rdi
  push %rsi
  push %rbp
  push %rdx
  push %rcx
  push %rbx
  push %rax

  mov %rsp, %rdi
  call trap
```

Kernel handler pushes the rest of regs.



```
struct trap_frame {
  uint64_t rax; // rax
  uint64_t rbx;
  uint64_t rcx;
  uint64_t rdx;
  uint64_t rbp;
  uint64_t rsi;
  uint64_t rdi;
  uint64_t r8;
  uint64_t r9;
  uint64_t r10;
  uint64_t r11;
  uint64_t r12;
  uint64_t r13;
  uint64_t r14;
  uint64_t r15;
  uint64_t trapno;
  /* error code, pushed by hardware or 0 by software */
  uint64_t err;
  uint64_t rip;
  uint64_t cs;
  uint64_t rflags;
  /* ss:rsp is always pushed in long mode */
  uint64_t rsp;
  uint64_t ss;
} __packed;
```

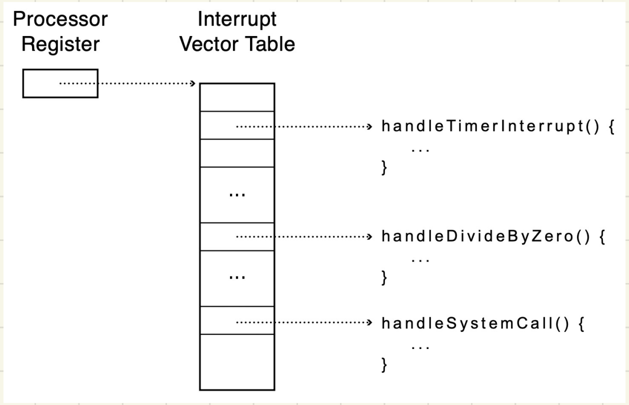
Saved by kernel

Saved by hw

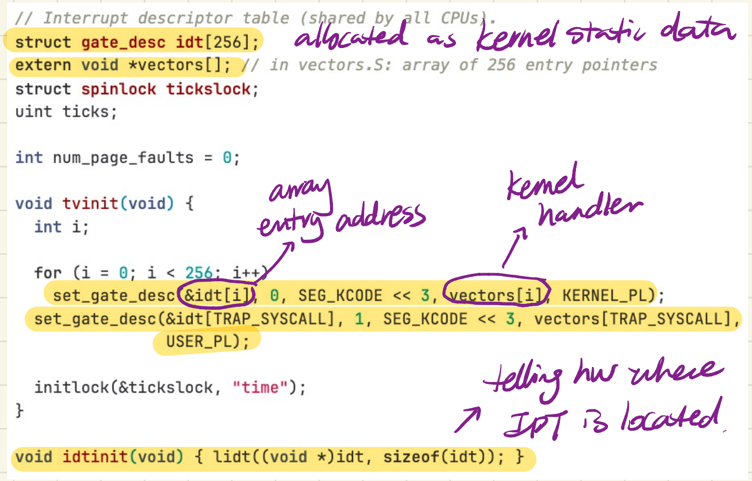
One per interrupt/ exception/ syscall

How does hw know which handler to load into %rip?

→ Interrupt Vector Table / Interrupt Descriptor Table (x86)



initialized by OS on startup



```
// Interrupt descriptor table (shared by all CPUs).
struct gate_desc idt[256];
extern void *vectors[]; // in vectors.S: array of 256 entry pointers
struct spinlock tickslock;
uint ticks;

int num_page_faults = 0;

void tvinit(void) {
  int i;

  for (i = 0; i < 256; i++)
    set_gate_desc(&idt[i], 0, SEG_KCODE << 3, vectors[i], KERNEL_PL);
  set_gate_desc(&idt[TRAP_SYSCALL], 1, SEG_KCODE << 3, vectors[TRAP_SYSCALL],
                USER_PL);

  initlock(&tickslock, "time");
}

void idtinit(void) { lidt((void *)idt, sizeof(idt)); }
```

allocated as kernel static data

array entry address

kernel handler

telling hw where IDT is located

→ x86 Interrupt Descriptor Table [architecture support]

→ Table of 256 entries

→ array index = interrupt #.
array entry = handler location