ECE391
Computer System Engineering

Lecture 13

Dr. Zbigniew Kalbarczyk

University of Illinois at Urbana- Champaign

Fall 2018

Lecture Topics

- Soft interrupts (tasklets) in Linux
- Summary of Linux interrupt system

Aministrivia

- **MP3 Teams**
  - **Due by end of Wednesday, October 10**

Soft Interrupts in Linux

linux/interrupt.h **and** *kernel/softirq.c*

- When are soft interrupts executed?
  - after a hard interrupt completes
  - periodically by a daemon in the kernel

- How?
  - seven or eight prioritized types, including high and low tasklet priorities
  - linked list for each tasklet priority
  - run on processor on which interrupt is scheduled
  - each handler atomic with respect to itself (only)

Soft Interrupts in Linux (cont.)

linux/interrupt.h and *kernel/softirq.c*

Declaring a handler

name for `tasklet_struct`

an unsigned long

```
DECLARE_TASKLET (name, func, data);
```

void (*func) (unsigned long);

next	linked list
state	TASKLET_STATE_SCHED, TASKLET_STATE_RUN
count	# of disables
func	pointer to the tasklet function
data	integer which can be used by the tasklet function

Tasklet Scheduling

- The following two calls schedule a tasklet for execution

```
void tasklet_schedule (struct tasklet_struct* t);
void tasklet_hi_schedule (struct tasklet_struct* t);
```

- First form
  - schedules tasklet at low priority
  - on the executing processor
- Second form schedules at high priority
- Enable and disable calls analogous to hard interrupts (including nesting)

Tasklet Execution

- **`do_softirq`** call
  - checks per-processor bit vector of pending priorities (high, low, etc.)
  - executes action for each priority [**`softirq_vec`**]
  - **`tasklet_action`** walks through linked list [**`tasklet_hi_action`** walks through high-priority list]
  - repeats up to 10 times or until no softirqs are raised

Tasklet Execution Atomicity

- Two bits in state changed atomically
  - TASKLET_STATE_SCHED – tasklet scheduled for execution
  - TASKLET_STATE_RUN – tasklet executing on some processor

- When scheduling
  - set TASKLET_STATE_SCHED atomically
  - if already set, schedule call does nothing

Tasklet Execution Atomicity (cont.)

When executing, for each tasklet in linked list (at either priority)

A. set TASKLET_STATE_RUN atomically (if already set, stop)

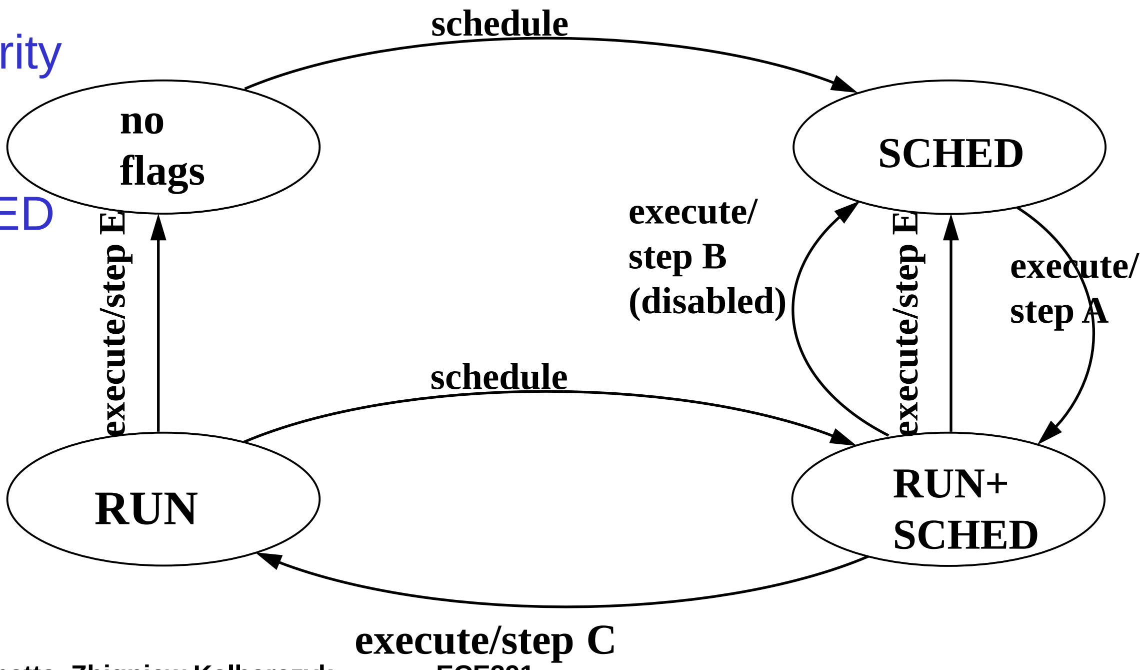
B. check if tasklet is software disabled (count field)

- if so, clear TASKLET_STATE_RUN
- leave the tasklet in the linked list for this priority
- set the pending bit for this priority
- daemon will try again later

C. clear TASKLET_STATE_SCHED

D. execute handler

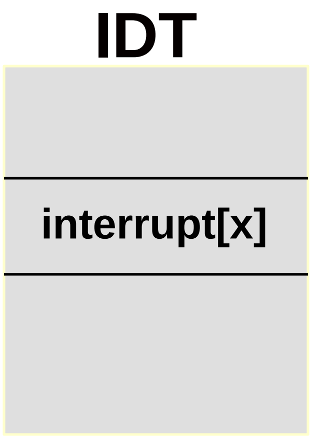
E. clear TASKLET_STATE_RUN



execution (execute/step D) occurs in lower two states (and execute/step A fails in these states)

Interrupt Descriptor Table (IDT)

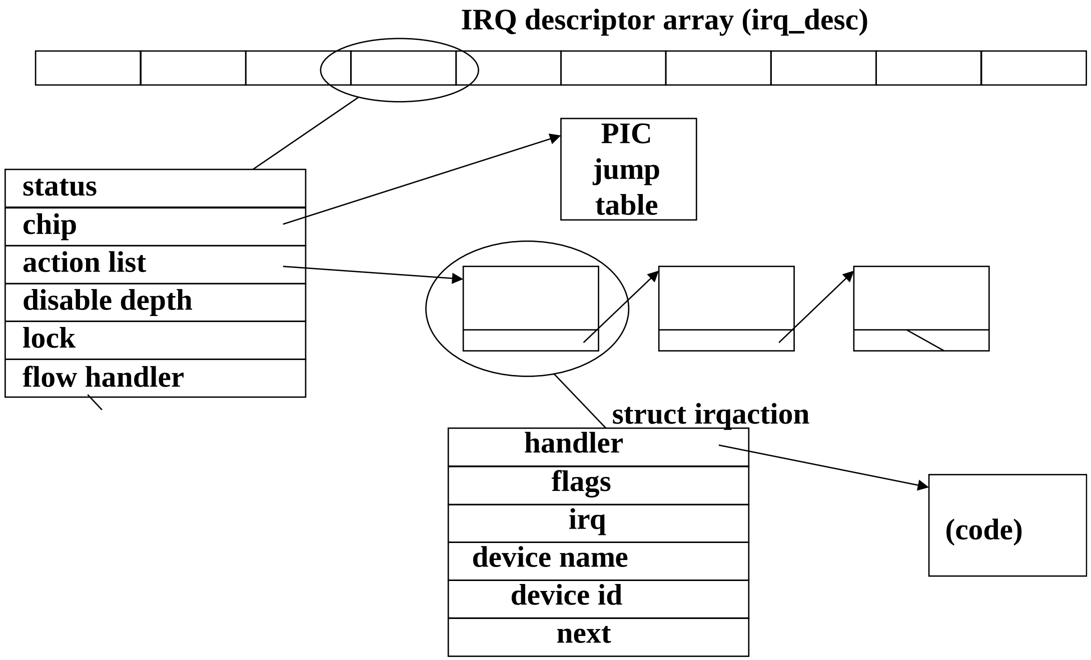
- Associates the interrupt line with the int. handler routine.
- 256 entries (each 8-bytes) or descriptors; each corresponds to an interrupt vector
  - hardware interrupts mapped into vectors 0x20 to 0x2F
- All Linux interrupt handlers are activated by so called: *interrupt gates (a descriptor type)*
- Whenever an interrupt gate is hit, interrupts are disabled automatically by the processor



- Before the kernel enables interrupts it must initialize the *idtr* register to point to the IDT table *(set by the kernel using* **`lidt`** *instruction)*

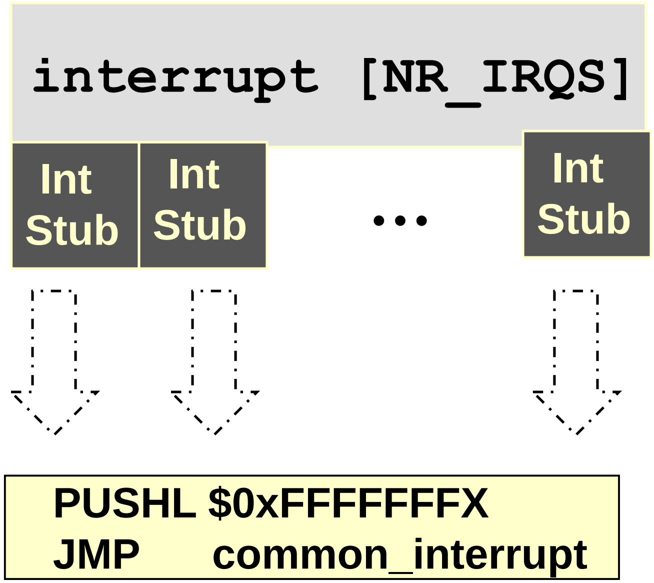
- **IDT Initialization**
- Kernel initialization (`setup_idt()`) fills all the 256 entries of IDT with the provisional (or null) handler

`irq_desc` *Array*



- Every interrupt vector has its own `irq_desc_t` descriptor
- Descriptors are grouped together in `irq_desc` array, a data structure supported by Linux
- When a device driver calls the `request_irq()` function a new structure to represent the handler is allocated and initialized

interrupt[NR_IRQS] Array



- Kernel maintains one global array of function pointers (`interrupt[NR_IRQS]`) in which it stores pointers to interrupt stubs (`NR_IRQS` is 16 if we use the PIC)

Initialization of Interrupt Gates

- During initialization `init_IRQ()` sets the status field of each IRQ descriptor to IRQ_DISABLED
- `init_IRQ()` updates the IDT by replacing the provisional interrupt gates with new ones

```
for (i = 0; i < NR_IRQS; i++)
     if (i+32 != 128)
          set_intr_gate(i+32, interrupt[i]);
```

- Interrupt gates are set to the addresses found in the `interrupt[NR_IRQS]` array