Operating System Concepts and Implementations

Betriebssystemkonzepte und -implementierungen

V. Interrupts (SoftIRQs)

November 10, 2026

Benedict Herzog

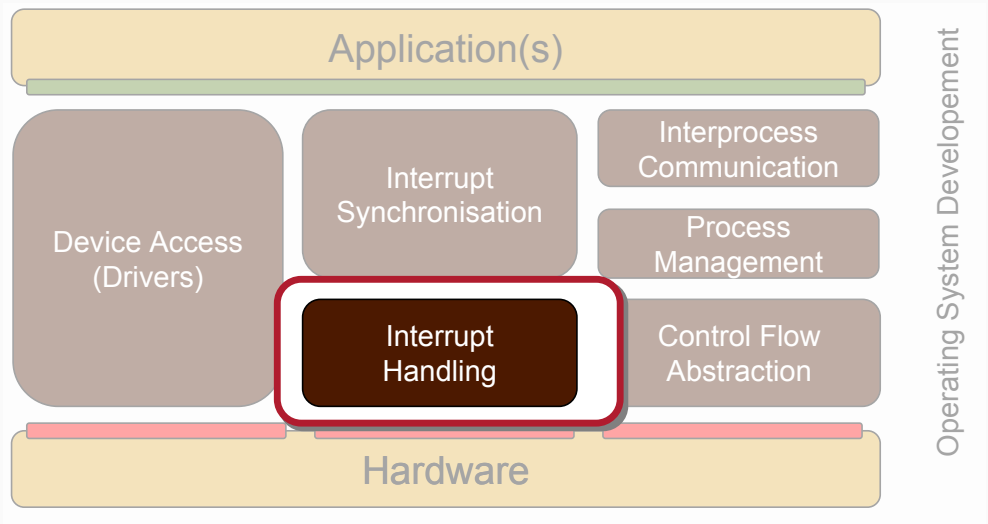
Material based on *Chair of Computer Science 4 (System Software)*
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

BOCHUM
OPERATING SYSTEMS
& SYSTEM SOFTWARE

RUHR
UNIVERSITÄT
BOCHUM

RUB

Overview: Today's Focus



Agenda



Context

Interrupt Handling Partitioning

Related Concepts

Summary

Agenda



Context

Interrupt Handling Partitioning

- Examples
- Concept
- Implementation
- SoftIRQs
- SoftIRQ Example

Related Concepts

Summary

Example: Console

What happens when a character is typed into the console?

1. The user presses a key.
2. The keyboard controller signals an interrupt.
3. The CPU reads the key code.
4. The CPU translates the key code into an ASCII character.
5. The ASCII character is stored in the input buffer.
6. If “ECHO” mode is enabled, the character is displayed on the screen
   → may include scrolling, cursor moving, setting/clearing pixels according to character set
7. If necessary, the CPU wakes up a waiting process.

Example: Network

What happens when an Ethernet TCP message is received?

1. A network interface card receives the Ethernet packet.
2. The network interface card signals an interrupt.
3. The CPU passes the packet, for example, to the IP layer.
4. The CPU verifies the IP checksum.
5. At the IP layer, multiple IP packets may be reassembled into a complete packet.
6. The CPU passes the packet, for example, to the TCP layer.
7. At the TCP layer, ACK messages are generated, for example.
8. The data is placed in the queue of the corresponding port.
9. If necessary, the CPU wakes up a waiting process.

Example: NFS Server



What happens when an NFS client requests data from the server?

1. IP message is received as described above.
2. The IP message is passed to the UDP layer.
3. According to the request in the packet, data is read from disk, for example.
4. The data and destination/recipient address are copied into a new UDP packet.
5. The UDP packet is passed to the IP layer.
6. If necessary, the IP packet is fragmented into several smaller IP packets.
7. A checksum is calculated for each packet.
8. All IP packets are passed to the Ethernet layer.
9. The network interface card transmits the Ethernet packets.

Problems:

- During the entire interrupt handling, all (lower-priority) interrupts are disabled.
  - Risk of high interrupt latency
  - Risk of data loss
- Interrupt handling cannot wait passively.
  - Interrupts are disabled.

Agenda



Context

Interrupt Handling Partitioning

- Examples
- Concept
- Implementation
- SoftIRQs
- SoftIRQ Example

Related Concepts

Summary

Example: Console

What happens when a character is typed into the console?

1. The user presses a key.
2. The keyboard controller signals an interrupt.
3. The CPU reads the key code.

⇐ **Hardware IRQ handled!**

4. The CPU translates the key code into an ASCII character.
5. ...

After step 3, IRQs could be enabled again!

Interrupt Handling Partitioning



Interrupt handling is divided into two parts:

1. Part that reads characters/packets/... from hardware and copies them into a buffer.
   - Interacts only minimally with the rest of the system.
   - Can (almost) always be executed.
2. Part that reads characters/packets/... from the buffer and processes them further.
   - Is largely hardware-independent.
   - Can (almost) always be interrupted.

Interrupt Handling Partitioning



Interrupt handling is divided into two parts:

The first part may be empty

- Example: timer/clock interrupt
- Typical for all *edge-triggered* interrupts

Interrupt Handling Partitioning

Multiprocessors:

- The first and second parts can be executed on different cores
  - Example: I/O interrupts; load balancing

  or
- The first and second parts must be executed on the same core
  - Example: scheduler interrupt

Agenda



Context

Interrupt Handling Partitioning

- Examples
- Concept
- Implementation
- SoftIRQs
- SoftIRQ Example

Related Concepts

Summary

Example / Idea

```
void keyboard_hard_irq() {
        uint8_t code = read_code_from_keyboard_controller();
        if (count < sizeof(buffer)) {
                buffer[head] = code;
                head = (head + 1) % sizeof(buffer);
                count++;
        }
}
void keyboard_soft_irq() {
        asm volatile ("cli\n");
        while (1 <= count) {
                uint8_t code = buffer[tail];
                tail = (tail + 1) % sizeof(buffer);
                count--;
                asm volatile ("sti\n");
                process_code(code);
                asm volatile ("cli\n");
        }
        asm volatile ("sti\n");
}
void keyboard_irq() {
        keyboard_hard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
}
```

Example / Idea

```
void keyboard_irq() {
        keyboard_hard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
}
```

Example / Idea

```
void keyboard_irq() {
        keyboard_hard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
}
```

This may cause a problem!

Example / Idea

```
void keyboard_irq() {
        keyboard_hard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
}
```

This may cause a problem!

- Problem: after `sti`, the stack may overflow!

Example / Idea

```
void keyboard_irq() {
        keyboard_hard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
}
```

This may cause a problem!

- Problem: after `sti`, the stack may overflow!
- Therefore (similar to Linux):

```
void keyboard_irq() {
        keyboard_hard_irq();
        apic_disable_keyboard_irq();
        asm volatile ("sti\n");
        keyboard_soft_irq();
        asm volatile ("cli\n");
        apic_enable_keyboard_irq();
}
```

```
apic_enable_keyboard_irq();
```

```
apic_disable_keyboard_irq();
```

May enable or disable all lower-priority interrupts, if necessary.

=> **User-defined interrupt priorities**

Interrupt Handling Partitioning

Disadvantages:

- Writing to/reading from the buffer takes time.
- Disabling/enabling interrupts takes time.

Advantage:

- Other(!), possibly important(!), interrupts are handled earlier.

Shared Interrupts

In the case of *shared interrupts*:

- Data must first be read from all involved devices, if necessary, and buffered temporarily
- Interrupts can then be enabled again
- The buffered data from all devices involved can then be processed further

Agenda



Context

Interrupt Handling Partitioning

- Examples
- Concept
- Implementation
- SoftIRQs
- SoftIRQ Example

Related Concepts

Summary

Note: `keyboard_soft_irq()` may not need to be invoked at all:

- A key has been released.
- A Shift/Alt/Ctrl/... key has been pressed.

Therefore, the `keyboard_hard_irq()` function could set a bit indicating whether the `keyboard_soft_irq()` function should be executed.

→ This corresponds to an interrupt mechanism (this time implemented in software): So-called **SoftIRQs**

SoftIRQ Interface

Typical SoftIRQ interface (per SoftIRQ):

`configure`: Register the handler function to be invoked when the SoftIRQ occurs.

`trigger`: If the SoftIRQ is disabled: set the pending bit;
otherwise: execute the handler function.

`disable`: Disable the SoftIRQ.

`enable`: Enable the SoftIRQ. If the pending bit is set, execute the handler function.

SoftIRQ Interface



```
void disable() {
        enabled = false;
}

void enable() {
        while (pending) {
                pending = false;
                handler();
        }

        enabled = true;
}

void trigger() {
        if (enabled) {
                asm volatile ("sti\n");
                handler();
                asm volatile ("cli\n");
        } else {
                pending = true;
        }
}
```

Caution! Race-Condition:

```
void disable() {
        enabled = false;
}

void enable() {
        while (pending) {
                pending = false;
                handler();
        }
        enabled = true;
}

void trigger() {
        if (enabled) {
                asm volatile ("sti\n");
                handler();
                asm volatile ("cli\n");
        } else {
                pending = true;
        }
}
```

⇐ trigger has no effect!

SoftIRQ Interface

`trigger` is only called from the interrupt handler.
It can therefore be synchronised hard using `cli/sti`:

```
void enable() {
        asm volatile("cli\n");
        while (pending) {
                pending = false;
                asm volatile("sti\n");
                handler();
                asm volatile("cli\n");
        }
        enabled = true;
        asm volatile("sti\n");
}
```

SoftIRQ Interface

Idea: *BSD: `trigger` is only called from the interrupt handler and therefore has access to the saved instruction pointer (IP).

```
void enable() {
        asm volatile("enable_begin:\n");
        while (pending) {
                pending = false;
                handler();
        }
        enabled = true;
        asm volatile("enable_end:\n");
}

void trigger() {
        if (enabled) {
                asm volatile("sti\n");
                handler();
                asm volatile("cli\n");
        } else {
                pending = 1;
                if (enable_begin <= IP && IP <= enable_end) {
                        IP = enable_begin;
                }
        }
}
```

SoftIRQ Interface



When an interrupt is signalled, hardware IRQs are only started by the hardware if they are not already being handled.

This behaviour can be emulated using an additional flag (`running`).

Agenda



Context

Interrupt Handling Partitioning

- Examples
- Concept
- Implementation
- SoftIRQs
- SoftIRQ Example

Related Concepts

Summary

SoftIRQ Example

```
void keyboard_hard_irq() {
        uint8_t code = read_code_from_keyboard_controller();
        if (count < sizeof(buffer)) {
                buffer[head] = code;
                head = (head + 1) % sizeof(buffer);
                count++;
        }
        apic_disable_keyboard_irq();
        trigger();
        apic_enable_keyboard_irq();
}

void keyboard_soft_irq() {
        asm volatile ("cli\n");
        while (0 < count) {
                uint8_t code = buffer[tail];
                tail = (tail + 1) % sizeof(buffer);
                count--;
                asm volatile ("sti\n");
                print_to_screen(code);
                asm volatile ("cli\n");
        }
        asm volatile ("sti\n");
}
```

SoftIRQ Example

```
void keyboard_soft_irq() {
        asm volatile ("cli\n");
        while (0 < count) {
                uint8_t code = buffer[tail];
                tail = (tail + 1) % sizeof(buffer);
                count--;
                asm volatile ("sti\n");
                print_to_screen(code);
                asm volatile ("cli\n");
        }
        asm volatile ("sti\n");
}

void console_write(char code) {
        disable();
        print_to_screen(code);
        enable();
}
```

Agenda



Context

Interrupt Handling Partitioning

Related Concepts

Summary

Related Concepts – Top/Bottom Halves



Idea (Linux):

- Interrupt handler reads data from device into a buffer and **sets a bit**.
- **Before returning to user mode**, the buffer is read and the data is processed **if the bit is set**.

Related Concepts – `ksoftirqd`

Idea (Linux):

- `while (pending) { pending = false; handler(); }` loop is executed only a limited number of times.
- Afterwards, **`ksoftirqd` (kernel thread)** is woken up and takes over SoftIRQ processing.
- **Advantage**: Interrupts cannot monopolise CPUs.

Related Concepts – Tasklets

Idea (Linux):

- The hard interrupt handler does not trigger the execution of a specific function (identified by a number)
- Instead: it adds an **arbitrary function** to a **list**.
- The list is then processed **before returning to user mode**.

Related Concepts – Interrupt Threads



Idea:

- Interrupt handler reads data from the device into a buffer.
- It wakes up a **thread**.
- **Thread** reads the data from the buffer and ...
  - Threads can have **priorities**.
  - Threads can **wait**.

Related Concepts – Interrupt Workqueues

Idea:

- Interrupt handlers add work items to a **workqueue**.
- **Workqueue** is processed by a **worker-thread pool**.

Agenda



Context

Interrupt Handling Partitioning

Related Concepts

Summary

Summary



- Many different approaches to interrupt handling.
- The goal is always to respond quickly to (important) interrupts.
- Almost always: a *hard* and a *soft* interrupt part.
- SoftIRQs: “interrupts” implemented in software.
- Modern operating systems provide further concepts derived from SoftIRQs.