4. Interrupt

NXKR_YoungSikYang

Exported on 02/29/2024

Table of Contents

1. Polling

Each case is checked by one by one in an infinite loop. This busy-waiting causes overheads. This is why the interrupt is used instead.

Polling

```
while True:
  if request==0:
    response0()
  elif request==1:
    response1()
  else
    response2()
```

2. About interrupt

An **interrupt** is a request for the processor to temporarily halt the current task and switch to another task with a higher priority.

2.1 Types of the interrupt

- **external interrupt(hardware interrupt)** happens as a result of outside interference such as from the user or from the peripherals. It functions as a notifier. (Considered a type of exception)
- **internal interrupt**, also called Trap, happens when wrong instructions or data are used. (exceptions like divided by zero, overflow)
- **software interrupt(system call)**: is a type of interrupt that is triggered by a specific instruction in a program, rather than by an external event or hardware malfunction. It's a mechanism for a program to interrupt the current process flow and request a service from the operating system.
  A program running in user mode needs to send a system call to the operating system to access system resources and perform tasks in kernel mode.

2.2 Steps of an interrupt

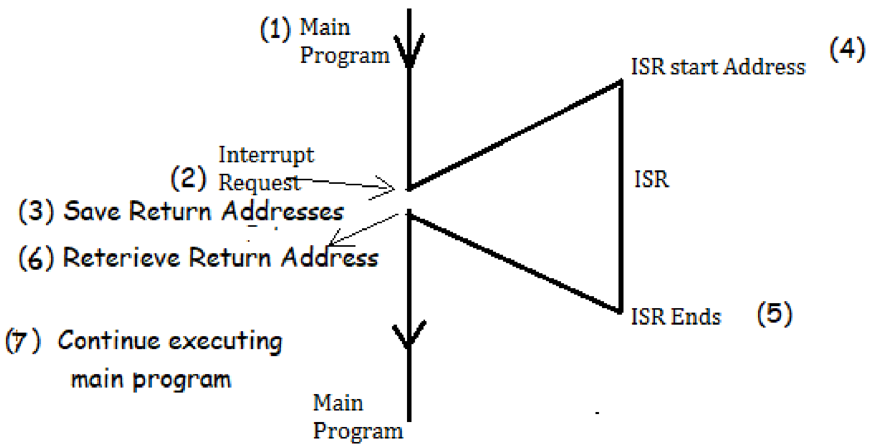
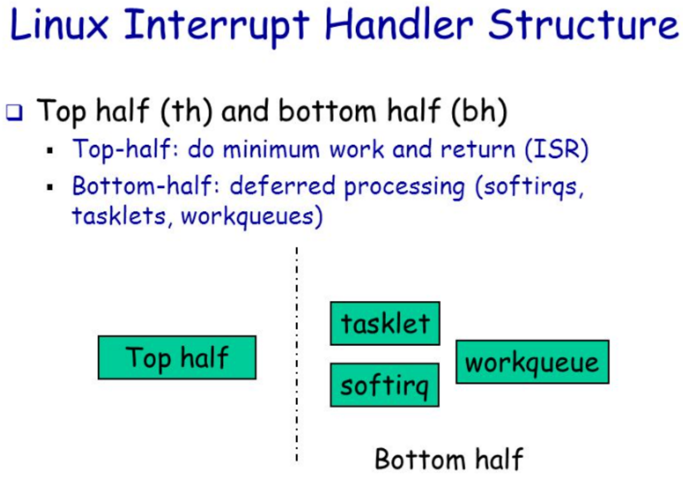


Figure 3.2 Interrupt Cycle

1. An interrupt occurs
2. Current program status is saved onto a stack
3. Jump to the interrupt vector
4. The ISR is executed
5. Jump to the previous program

2.3. ISR(Interrupt Service Routine)

ISR handling in Linux is divided into top-half and bottom-half to balance the need for immediate response to interrupts with the need to perform more complex processing without compromising system responsiveness.



2.3.1 Top-half

- The top half refers to the initial response to an interrupt.
- It performs minimal work to ensure the system can resume its operation as quickly as possible and defers the rest of the time-consuming work to the bottom half.
- It involves acknowledging the interrupt, reading or writing the minimal necessary data.

2.3.2 Bottom Half

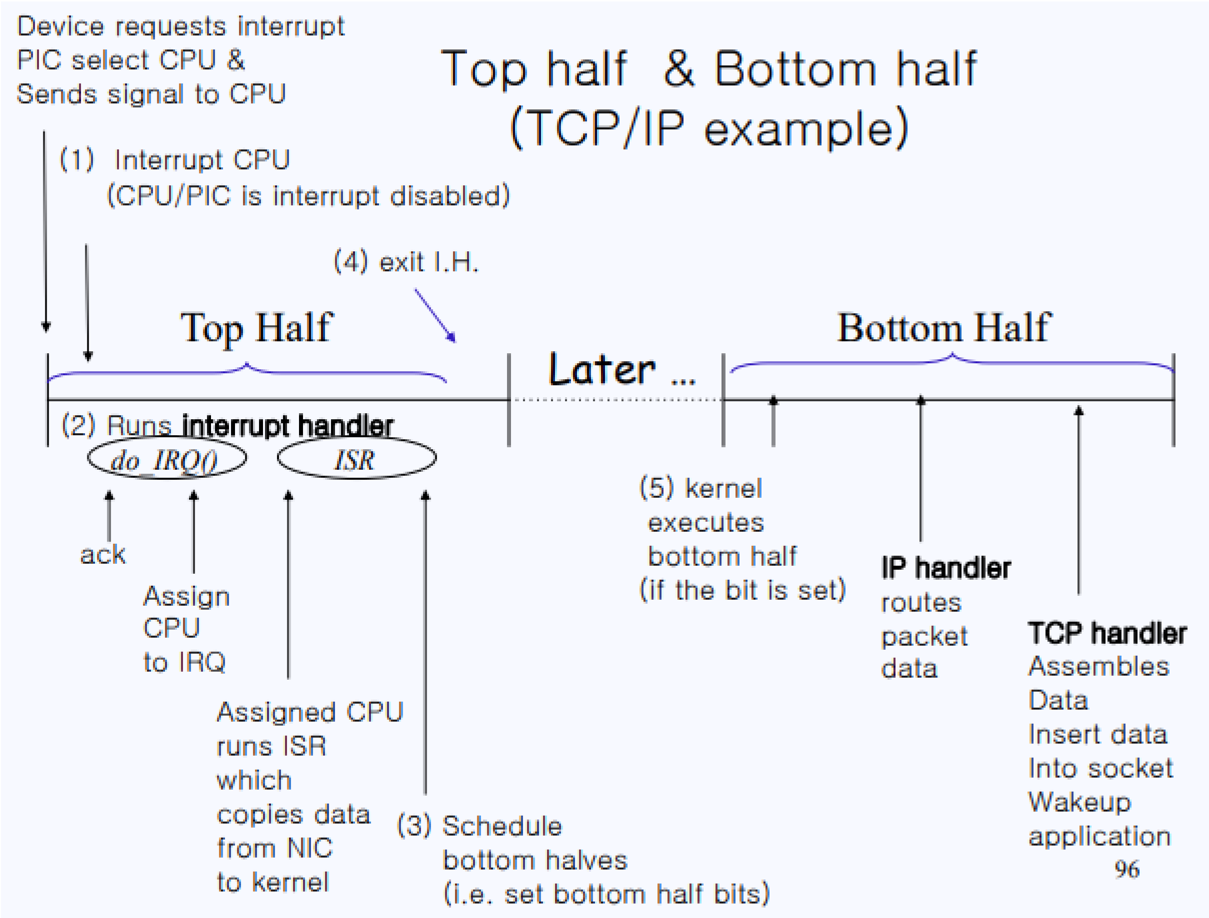
This is where the deferred work from the interrupt is handled and can be done in various ways as listed below:

Feature	SoftIRQs	Tasklets	Workqueues	Threaded IRQs
Definition	Low-level mechanism for bottom-half interrupt handling.	Higher-level mechanism built on top of softirqs for bottom-half interrupt handling.	Mechanism that allows kernel functions to be executed in the context of a kernel thread.	Mechanism that allows interrupt handling to be performed in the context of a kernel thread.

Feature	SoftIRQs	Tasklets	Workqueues	Threaded IRQs
Concurrency	Can run simultaneously on multiple CPUs, but the same softirq type will not run concurrently on two CPUs.	Serialized; two tasklets of the same type will not run simultaneously on two CPUs.	Work can run concurrently on multiple CPUs; workqueues can be configured to be non-reentrant.	Can run concurrently, allows for synchronization mechanisms to manage access to shared resources.
Context	Interrupt context; cannot sleep.	Interrupt context; cannot sleep.	Process context; can sleep.	Process context; can sleep.
Use Case	Suitable for high-speed and low-latency interrupt handling.	Suitable for tasks that don't require immediate action and can be serialized.	Suitable for longer-running jobs that might need to sleep, wait for I/O operations, or require scheduling.	Used for interrupt handling that requires sleeping, waiting for I/O, or complex processing that should not be done in interrupt context.
Limitations	Cannot sleep; must be quick and non-blocking.	Inherits softirqs' limitations	Overhead of scheduling and running a kernel thread; not suitable for high-speed, low-latency requirements.	Additional overhead compared to traditional IRQ handling; complexity of managing a threaded context.
Example Usage	Network packet processing, timer updates.	Deferring work from a softirq, simple background tasks.	Filesystem operations, scheduled maintenance tasks.	Handling interrupts from devices that require complex processing or waiting on I/O operations.

2.3.3 Example

Example of the top-half & bottom-half in TCP/IP


Device requests interrupt
PIC select CPU &
Sends signal to CPU
Top half & Bottom half
(TCP/IP example)
(1) Interrupt CPU
(CPU/PIC is interrupt disabled)
(4) exit I.H.
Top Half
Later ...
Bottom Half
(2) Runs interrupt handler
do_IRQ()
ISR
ack
Assign
CPU
to IRQ
Assigned CPU
runs ISR
which
copies data
from NIC
to kernel
(3) Schedule
bottom halves
(i.e. set bottom half bits)
(5) kernel
executes
bottom half
(if the bit is set)
IP handler
routes
packet
data
TCP handler
Assembles
Data
Insert data
Into socket
Wakeup
application
96

References

- https://karatus.tistory.com/196
- https://hackmd.io/@happy-kernel-learning/rJOX15zqU