Interrupts

www.tutorialsdaddy.com

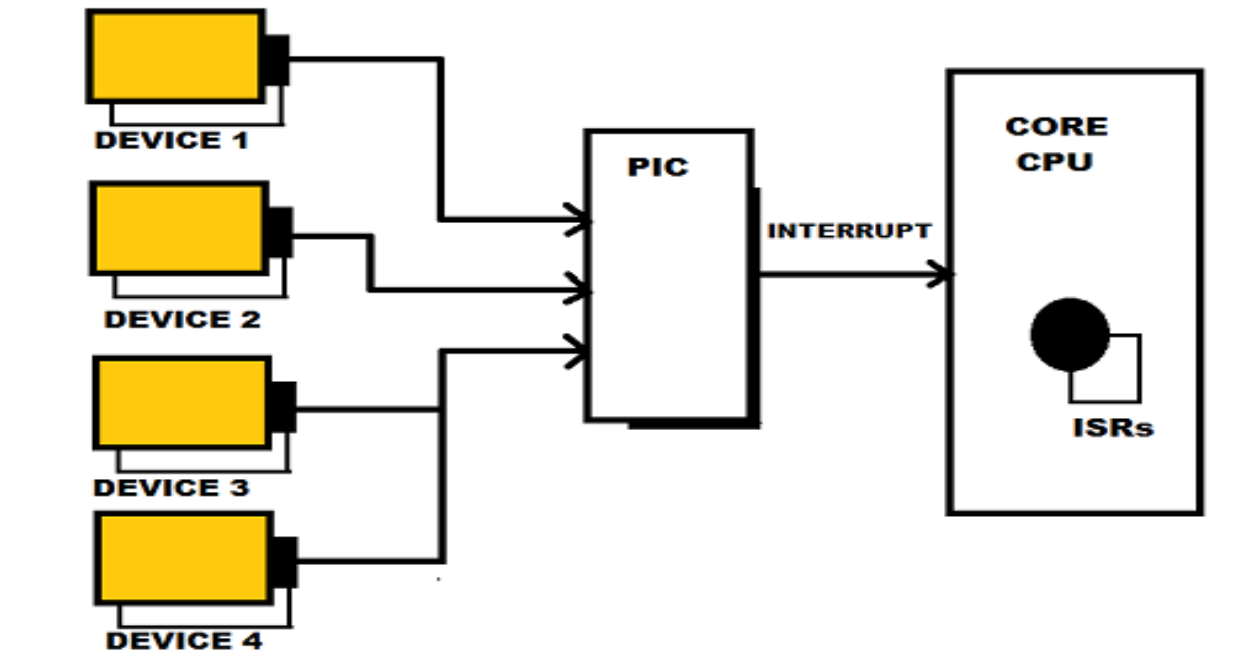
OVERVIEW

Interrupts enable hardware to signal to processor. Suppose you are cooking something and somebody enters into your kitchen and ask you to give a glass of water . You will pause your cooking and will give water to that person . after serving the person you will again start cooking and will continue your work as usual.

Same way when CPU is busy in doing or processing something any hardware can request to CPU for some service. CPU saves it current state and serves the hardware request. After full filling the request of hardware CPU restore its previous state and start processing previous that it was doing before receiving from hardware.

As you can see in the given picture , these devices are connected to interrupt controller (which controls the hardware and forward the interrupt request to CPU) via some physical wires or lines. These lines are generally called Interrupt Request lines(IRQ) lines. Each of these lines are given a unique number as an identity .

IRQ(0) is reserved for timer interrupt.



Interrupt Handlers:-

This is the function which kernel runs in response to a specific interrupt is called interrupt handler or ISR. This function actually handle and process the request of interrupts. Interrupt handler are mainly divided into two parts

1)Top Halves

Top halves perform the work which is time critical such as acknowledgement receipt of interrupt.

2) Bottom Halves

work that can be done later is called bottom halves.

Registering an Interrupt handler:-

It is the responsibility of device driver to write the interrupt handler and take care of interrupt request generated from its device. Kernel provide below function to register an interrupt.

int request_irq()

Below are the arguments we need to pass to register an interrupt

1)interrupt number:- this number is fixed for some device otherwise determined pro-grammatically

2)Handler:- it's a function pointer to the actual interrupt handler that service this interrupt.

return type:- typedef irqreturn_t()();

3)Interrupt handler flags:- This parameter flag either with 0 bitmask of one or more of the flags defined in <linux/interrupt.h>

IRQF_DISABLED:- when this flag is set. It indicates the kernel to disable all interrupts when executing this interrupt handler.

when unset , interrupt handler run with all interrupts except their own enabled.
IRQF_TIMER:- this flag specify that this handler processes interrupts for the system timer.
IRQF_SHARED:- This flag specifies that interrupt line can be shared among multiple interrupt handler.

4) Name:- Its an ascii text representation of the device associated with the interrupt. we can dispaly these names by launching /proc/irq and proc/interrupts command on shell.

5)Dev:- is used for shared interrupt lines when an interrupt handler is freed , dev provides kernel a unique cookies to enables removal of only the desired particular handler from the interrupt line.

Freeing an Interrupt Handler:-Below is the function availed by kernel to free the interrupt handler.

void free_irq(unsigned int irq, void *dev)

Writing an interrupt handler:-In writing interrupt handler we need to think about what are the services our device may request. kernel provide below syntax of interrupt handler.

static irqreturn_t intr_handler(int irq,void *dev)

first argument is the interrupt line and second argument is dev structure which is used in case of shared line.

Status of the interrupt system:–

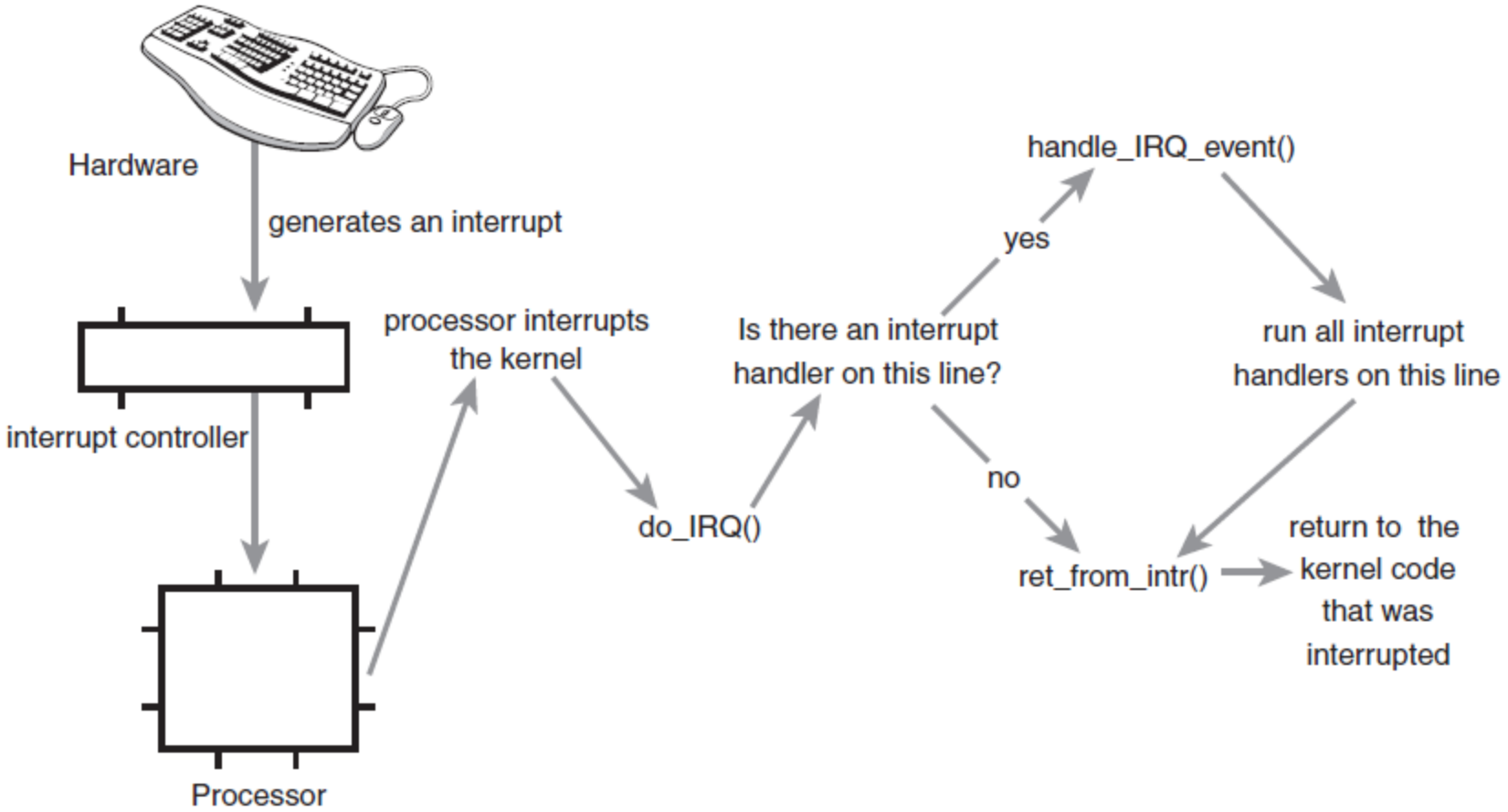
Kernel provides below function to know that weather we are handling the interrupt currently or not.

Functions	Descriptions
local_irq_disable()	Disable local interrupt
local_irq_enable()	Enable local interrupt
local_irq_save()	Save the current state of local interrupt and then disable it
local_irq_restore()	Restore local interrupt to the previous state
disable_irq()	Disable the given interrupt line and ensure no handler on the line is executing before returning
disable_irq_nosync()	Disable the given interrupt line
enable_irq()	Enable the given interrupt line

Status of the interrupt system:

Kernel provides below function to know that weather we are handling the interrupt currently or not.

Function	Description
In_interrupt()	Return non zero if interrupt context and zero if in process context
In_irq()	Return non zero if currently executing an interrupt handler and zero otherwise



Q/A

Post your question @

http://www.tutorialsdaddy.com/forums/forum/linux-device-driver/

Please like and subscribe our channel for more videos

Connect with us on Facebook
www.facebook.com/tutorialsdaddy

Visit our website and subscribe for more updates
www.tutorialsdaddy.com

Thank You