General Agenda

1. Hardware interrupts
2. Softirq
3. Tasklet
4. Workqueue

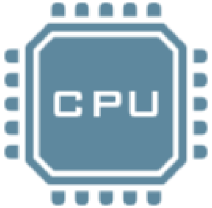
Detail Agenda

1. Interrupts
2. Realtime and Interrupts
3. Maskable interrupt
4. Non-Maskable interrupt
5. Exceptions
6. APIC + PIC
7. Softirq
8. Tasklet
9. Workqueue
10. Detail

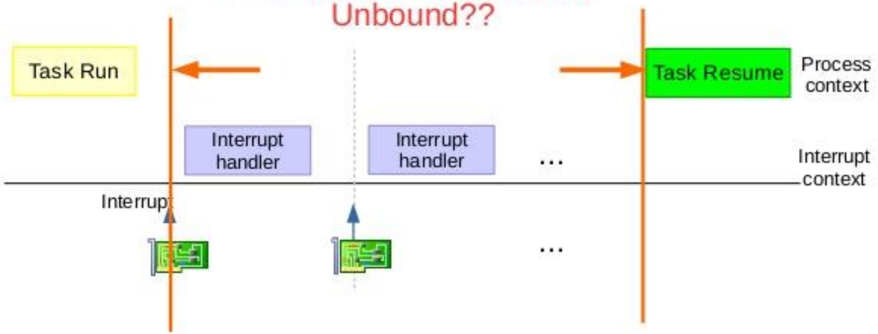
Mr. Micro-controller, May I have your attention?

Interrupts

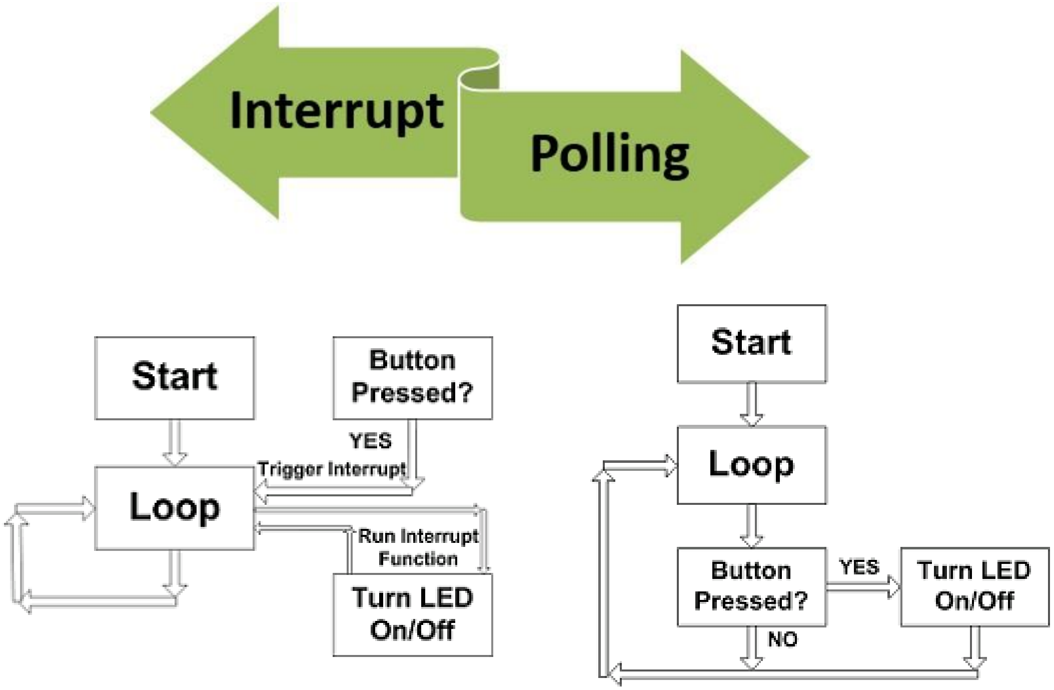
I am a peripheral!



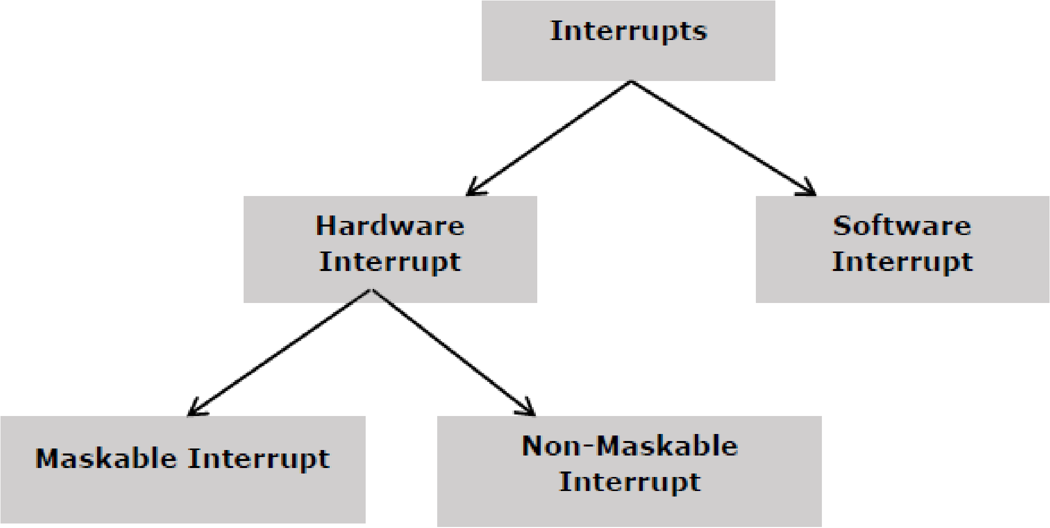
Task interrupted by IRQ



- **Difficult** is something that can be done **immediately**;
- **Impossible** is that which will take a little **longer**”

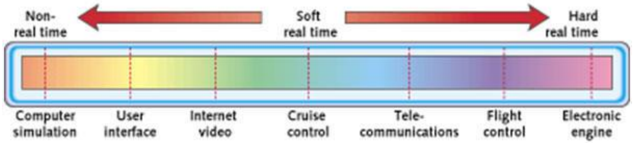


Interrupts



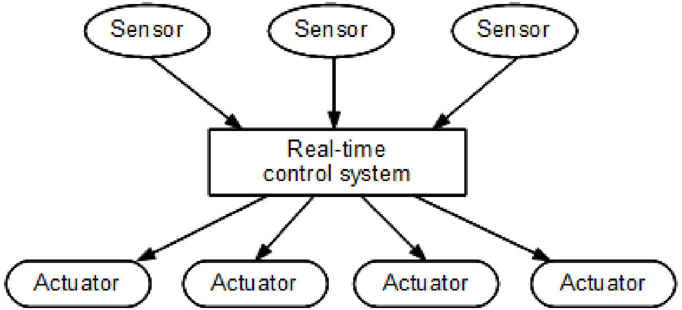
- **Difficult** is something that can be done **immediately**;
- **Impossible** is that which will take a little **longer**”

IRQ - Interrupt ReQuest



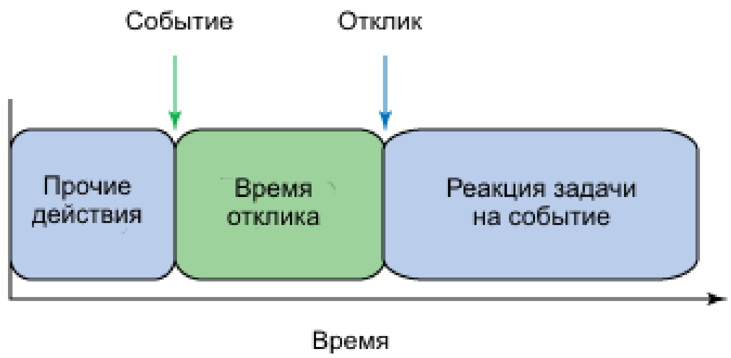
RealTime and Interrupts

- ***Hard*** – missing a deadline is a total system failure.
- ***Firm*** – infrequent deadline misses are tolerable, but may degrade the system's quality of service. The usefulness of a result is zero after its deadline.
- ***Soft*** – the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.



RealTime and Interrupts

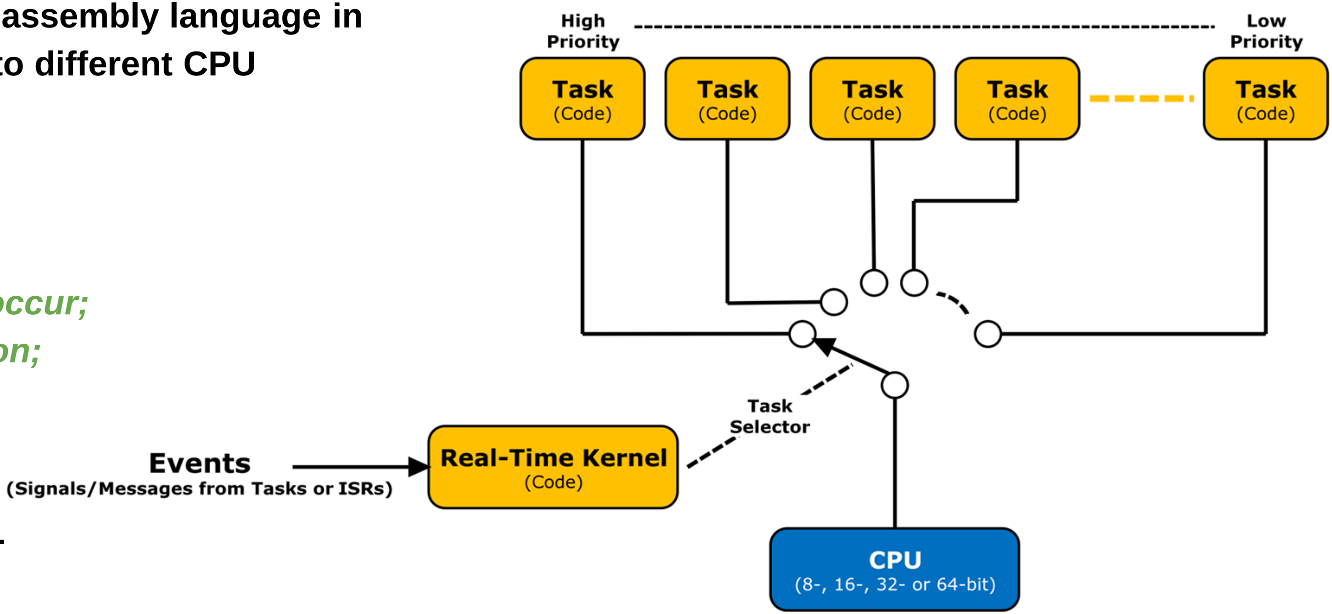
- ***Асинхронные события*** — полностью непредсказуемые события. Например, вызов абонента телефонной станции.
- ***Синхронные события*** — предсказуемые события, случающиеся с определённой регулярностью. Например, вывод аудио и видео.
- ***Изохронные события*** — регулярные события (разновидность асинхронных), случающиеся в течение интервала времени. Например, в мультимедийном приложении данные аудиопотока должны прийти за время прихода соответствующей части потока видео.



RealTime and Interrupts

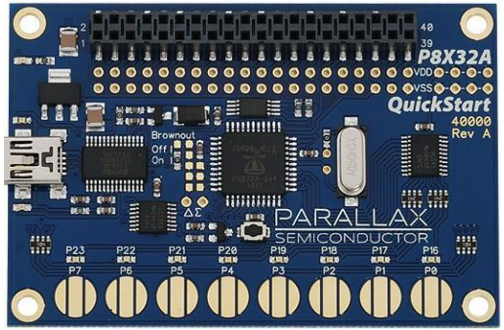
- Real-Time Kernel
- A real-time kernel is software that manages the time of a CPU (Central Processing Unit) or MPU (Micro Processing Unit) as efficiently as possible. Most kernels are written in C and require a small portion of code written in assembly language in order to adapt the kernel to different CPU architectures.
- *void MyTask (void)*

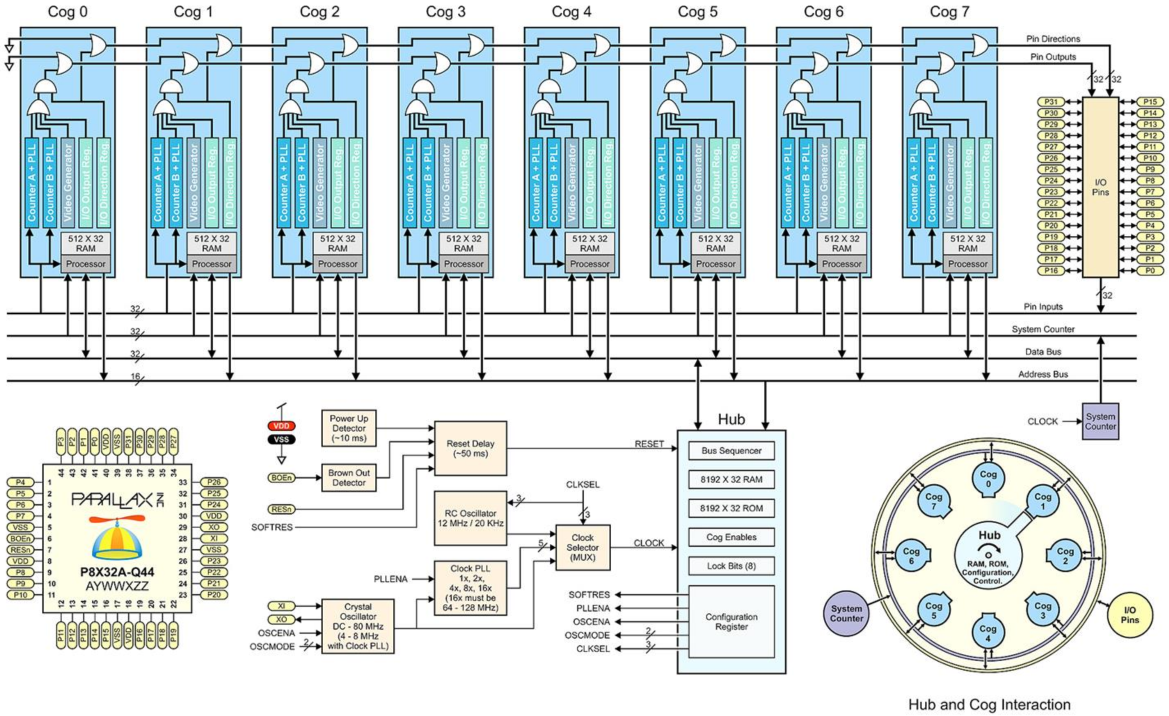
```
{
    while (1) {
        Wait for an event to occur;
        Perform task operation;
    }
}
```



Paralax Propeller

- Нет понятия прерываний. Вместо это предлагается запускать конкурирующие задачи разных ядрах (cog'ах).





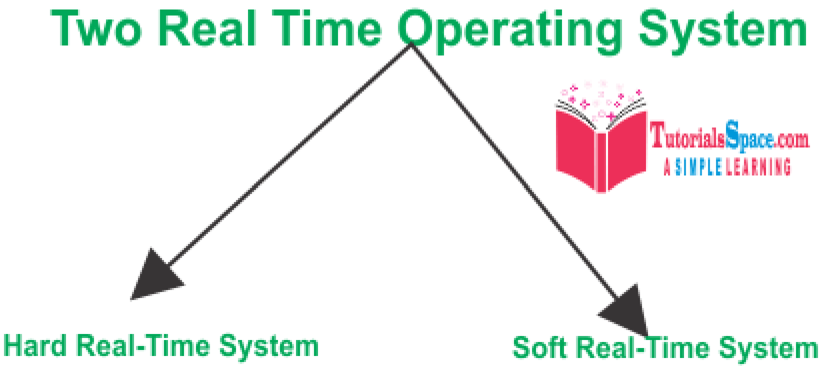
RealTime and Interrupts

- *Difference between real time operating system and non real time operating system?*

RealTime and Interrupts

Real time OS

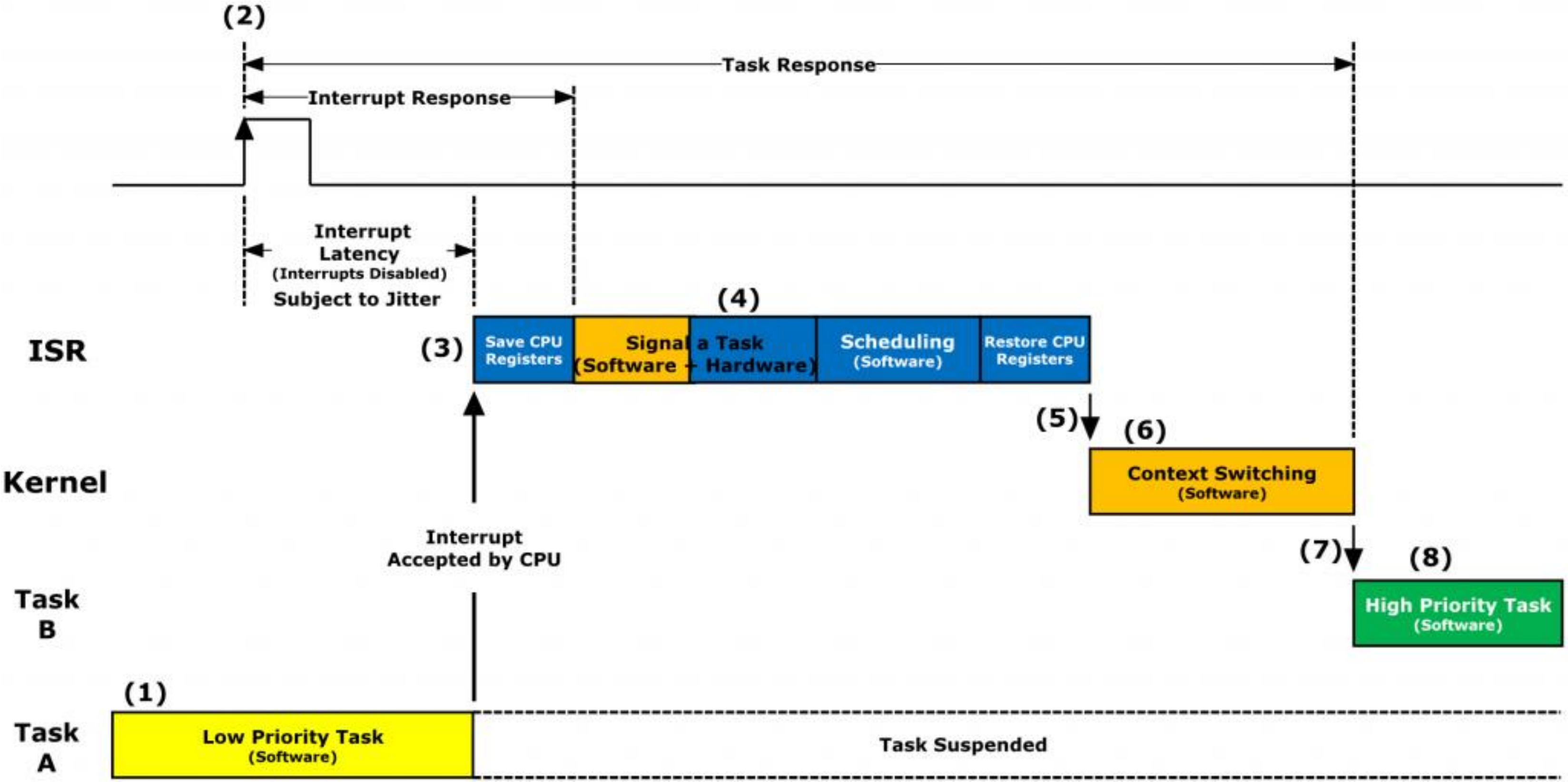
1. A real-time operating system is an operating system intended to serve real-time applications that process data as it comes in, typically without buffer delays.
2. It is deterministic.
3. It is time sensitive.
4. It can’t use virtual memory.
5. It is dedicated to single work.
6. It has flat memory model.
7. It has low interrupt latency.



RealTime and Interrupts

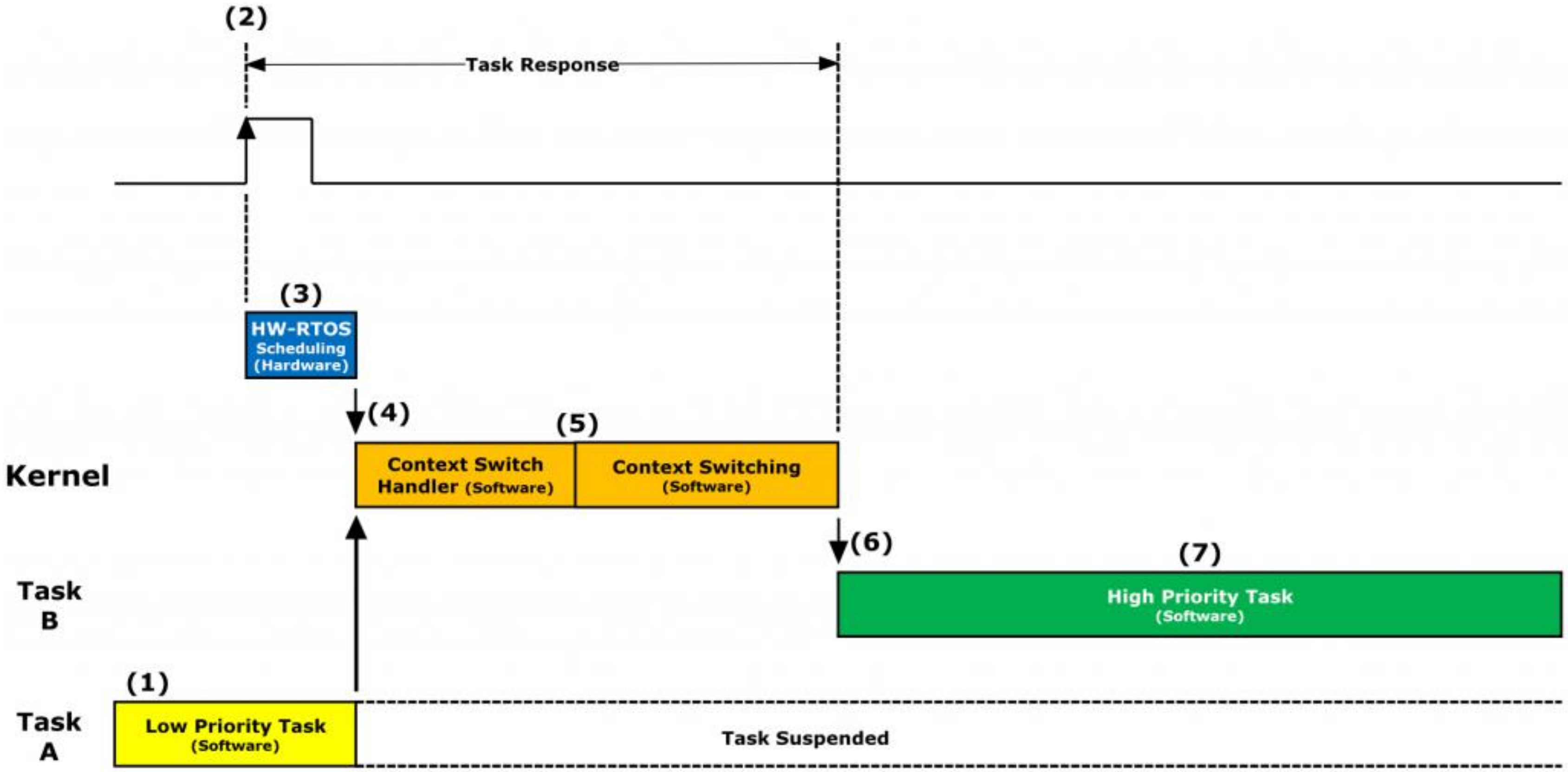
RTOS

- HW-RTOS can handle interrupting devices using ISRs like other kernels



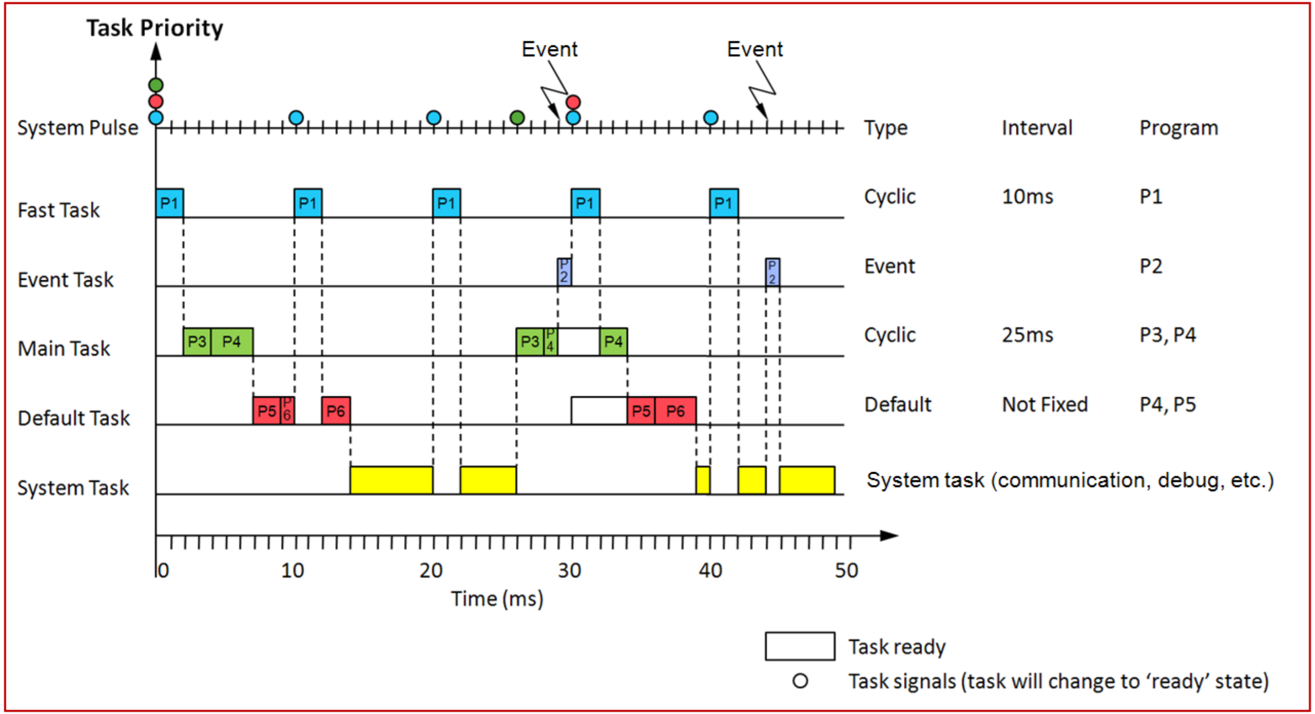
RTOS

- Interrupting devices can directly signal a task



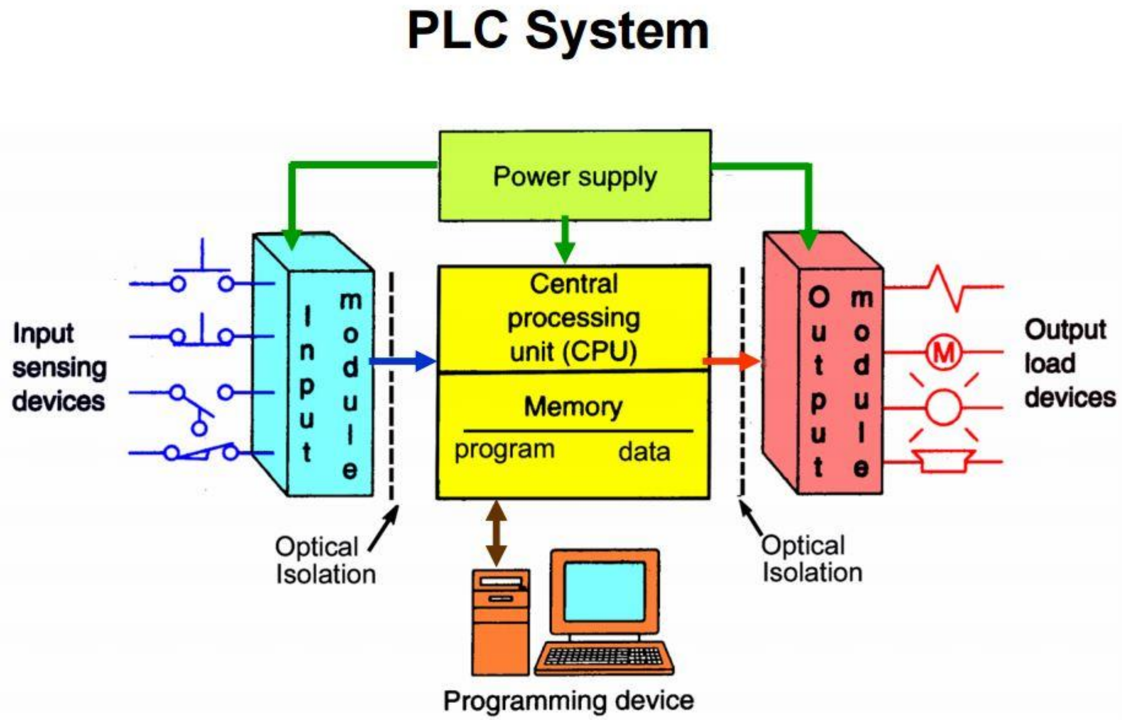
RealTime and Interrupts

- IEC 61131-3 programming 5 tasks in one project.



RealTime and Interrupts

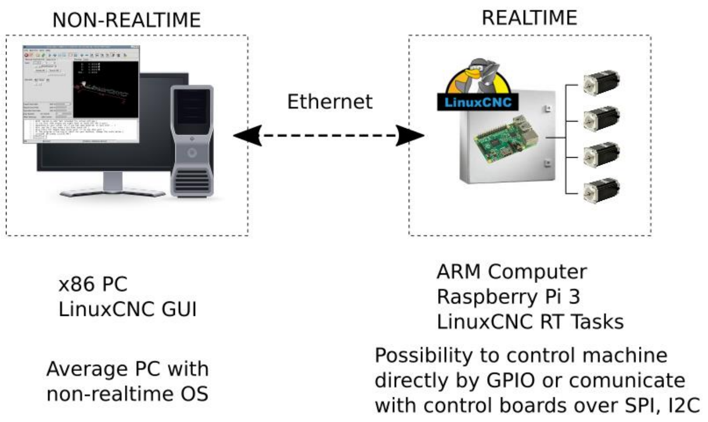
- PLC model



RealTime and Interrupts

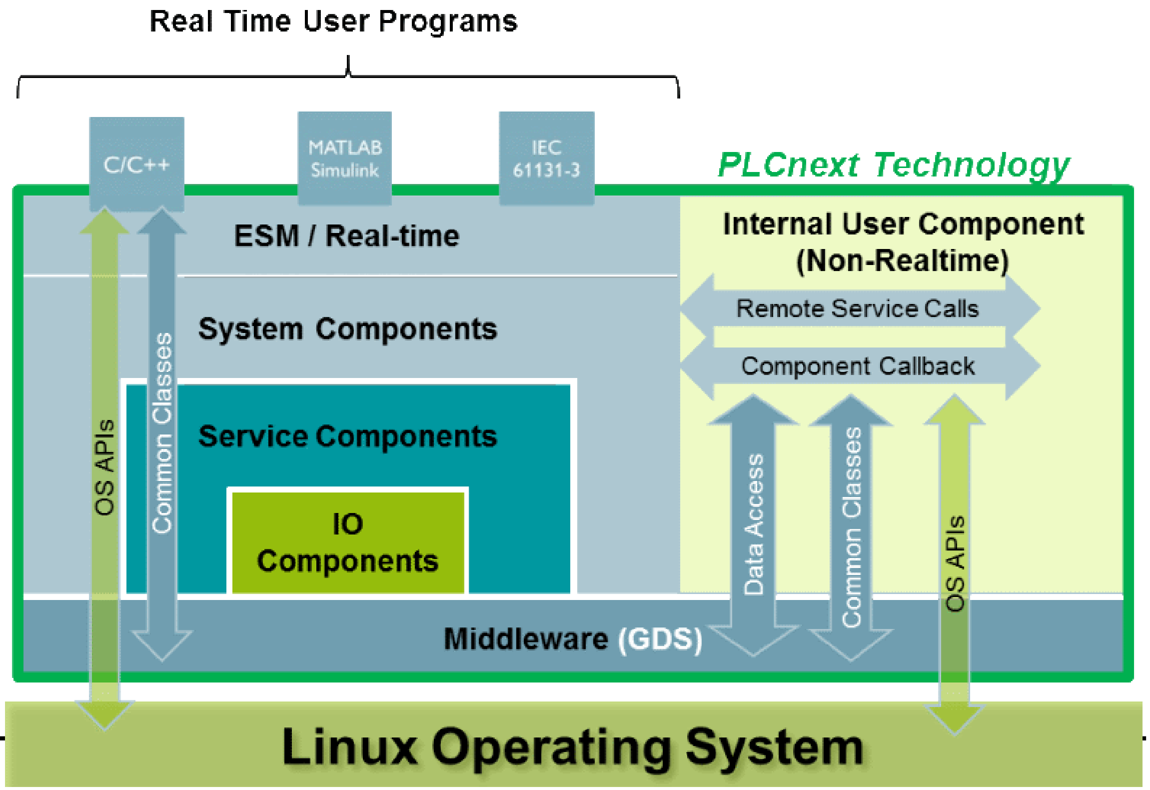
Non-real time OS

1. A Non-real time OS or General purpose OS is the operating system made for high end, general purpose systems like a personal computer, a work station, a server system etc.
2. It is not deterministic.
3. It is time insensitive.
4. It can use virtual memory concept.
5. It is used in multi-user environment.
6. It has protected memory model.
7. It has high interrupt latency



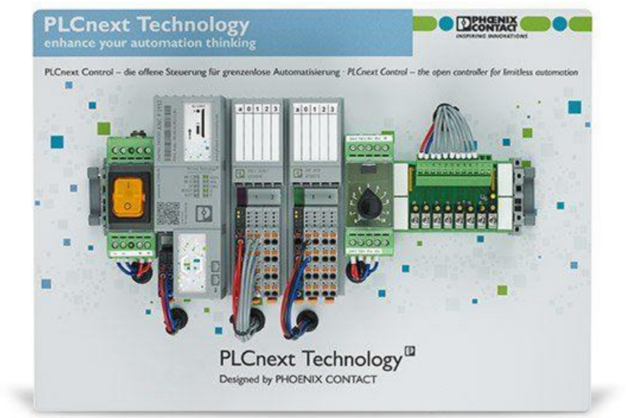
RealTime and Interrupts

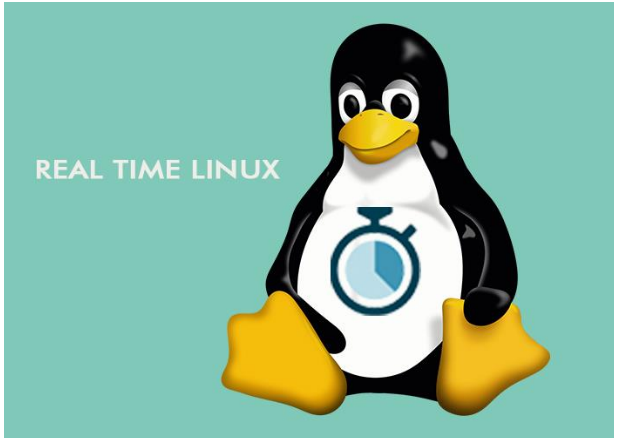
- PLCnext model



RealTime and Interrupts

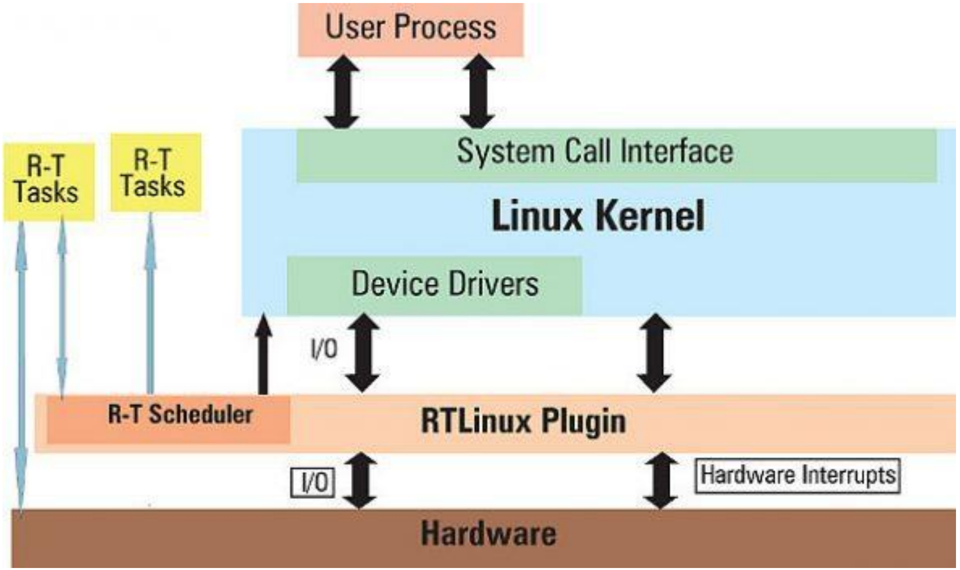
- PLC next





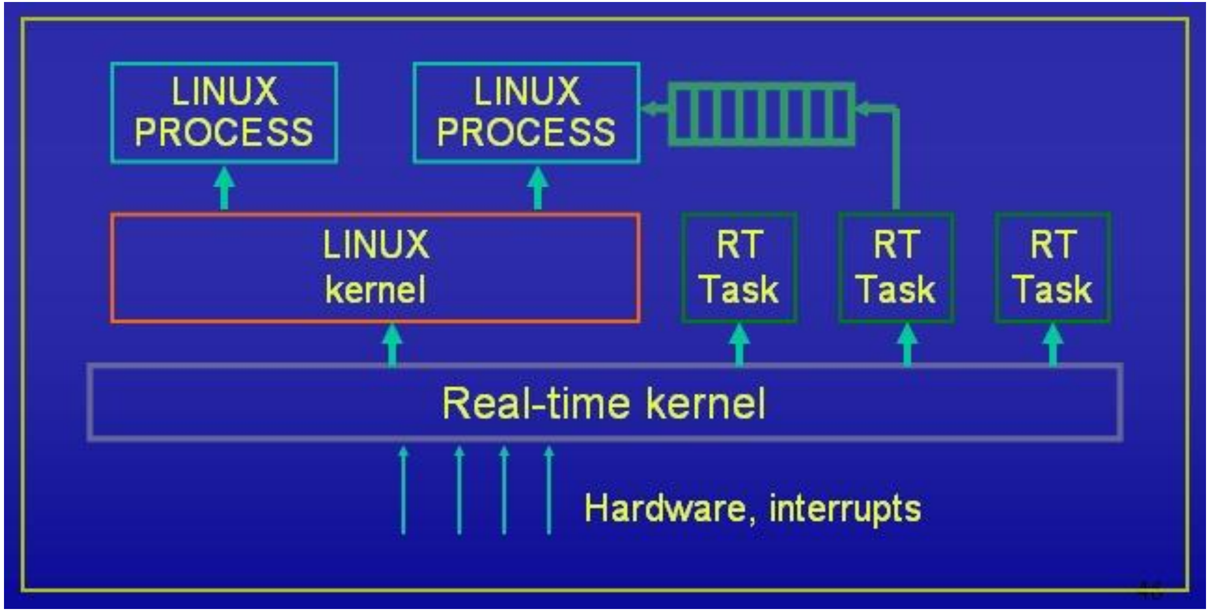
RealTime and Interrupts

- **RTLinux**
  RTLinux is a micronuclear hard real-time operating system that runs Linux as a completely extruded process.

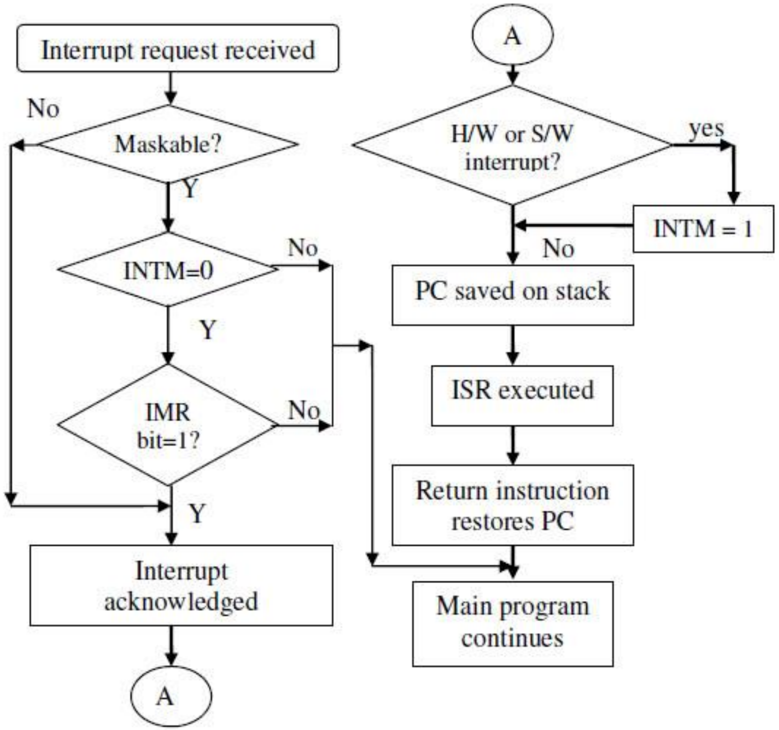


Hardware interrupt

- **RTLinux**

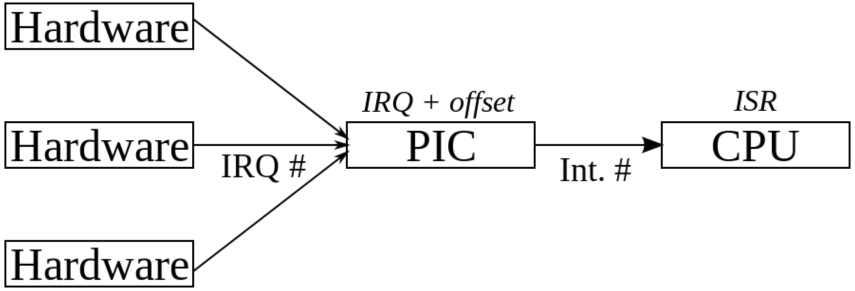


Maskable interrupts



- Maskable interrupt (IRQ): a hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask.

Maskable interrupts



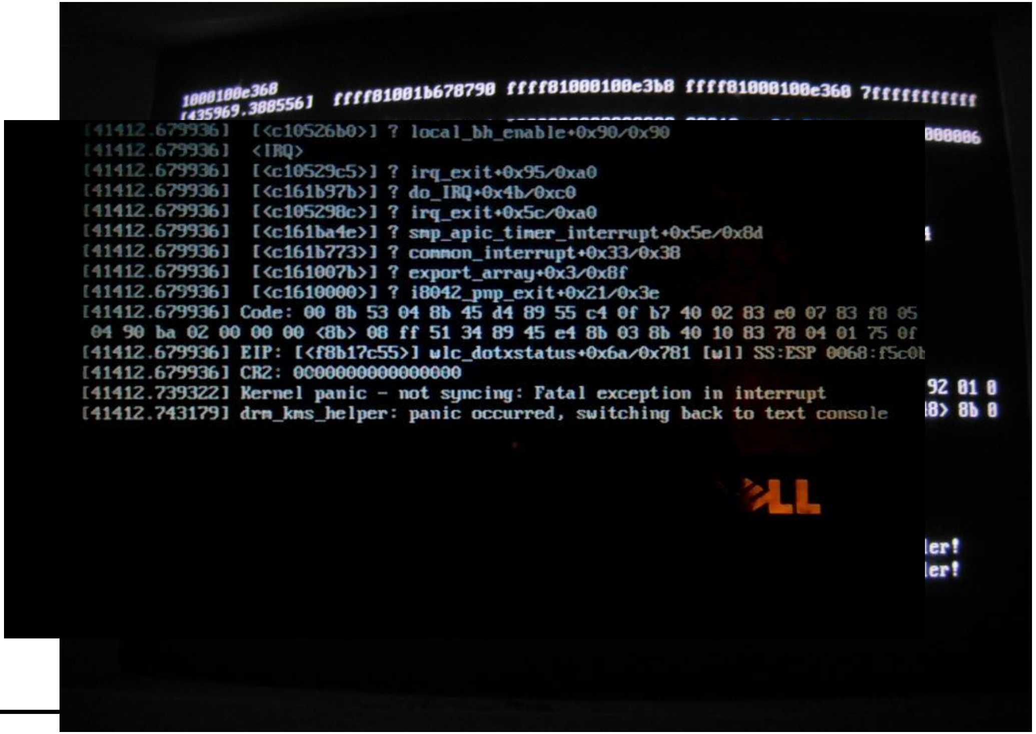
- All Interrupt Requests (IRQs) issued by I/O devices give rise to **maskable interrupts** . A maskable interrupt can be in two states: masked or unmasked;
- a masked interrupt is ignored by the control unit as long as it remains masked.

IRQ

- Each IRQ line is assigned a numeric value. For example, on the classic PC, IRQ zero is the timer interrupt and IRQ one is the keyboard interrupt.
- Some interrupts are dynamically assigned, such as interrupts associated with devices on the PCI bus. Other non-PC architectures have similar dynamic assignments for interrupt values.
- The kernel knows that a specific interrupt is associated with a specific device. The hardware then issues interrupts to get the kernel's attention.

IRQ


A problem has been detected and Windows has been shut down to prevent damage to your computer.
DRIVER_IRQL_NOT_LESS_OR_EQUAL
If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
Technical information:
*** STOP: 0x000000D1 (0xE10C5678,0x00000002,0x00000000,0xF8C48579)
*** myfault.sys - Address F8C48579 base at F8C48000, DateStamp 4d26670c


[41412.679936] [<c10526b0>] ? local_bh_enable+0x90/0x90
[41412.679936] <IRQ>
[41412.679936] [<c10529c5>] ? irq_exit+0x95/0xa0
[41412.679936] [<c161b97b>] ? do_IRQ+0x4b/0xc0
[41412.679936] [<c105298c>] ? irq_exit+0x5c/0xa0
[41412.679936] [<c161ba4e>] ? smp_apic_timer_interrupt+0x5e/0x8d
[41412.679936] [<c161b773>] ? common_interrupt+0x33/0x38
[41412.679936] [<c161007b>] ? export_array+0x3/0x8f
[41412.679936] [<c1610000>] ? i8042_pnp_exit+0x21/0x3e
[41412.679936] Code: 00 8b 53 04 8b 45 d4 89 55 c4 0f b7 40 02 83 e0 07 83 f8 05
04 90 ba 02 00 00 00 <8b> 08 ff 51 34 89 45 e4 8b 03 8b 40 10 83 78 04 01 75 0f
[41412.679936] EIP: [<f8b17c55>] wlc_dotxstatus+0x6a/0x781 [wl] SS:ESP 0068:f5c0b
[41412.679936] CR2: 0000000000000000
[41412.739322] Kernel panic - not syncing: Fatal exception in interrupt
[41412.743179] drm_kms_helper: panic occurred, switching back to text console

Non- Maskable interrupts

- Only a few critical events (such as **hardware failures**) give rise to nonmaskable interrupts;
- Nonmaskable interrupts are always recognized by the CPU.

Exceptions

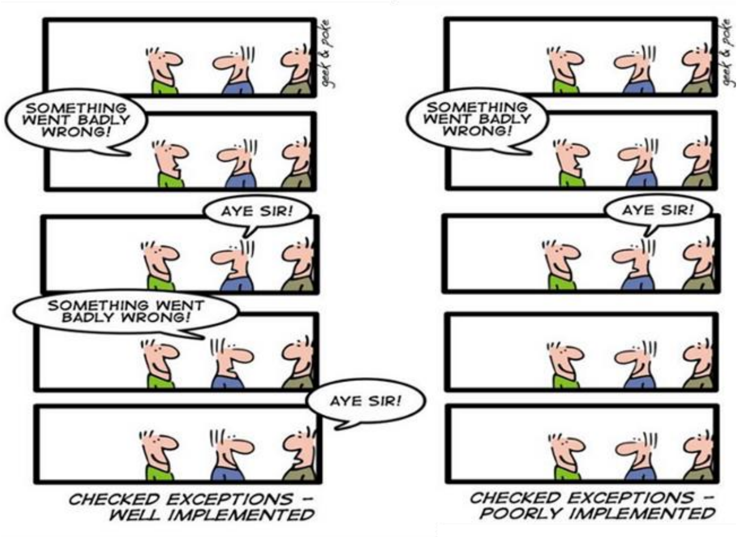
- Processor-detected exceptions
- Faults
- Traps
- Aborts
- Programmed exceptions

Exceptions

- Processor-detected exceptions
  Generated when the CPU detects an anomalous condition while executing an instruction. These are further divided into three groups, depending on the value of the eip register that is saved on the Kernel Mode stack when the CPU control unit raises the exception.

Exceptions

- Faults (**Ошибки**)
  Can generally be corrected; once corrected, the program is allowed to restart with no loss of continuity.
- The saved value of eip is the address of the instruction that caused the fault, and hence that instruction can be resumed when the exception handler terminates.

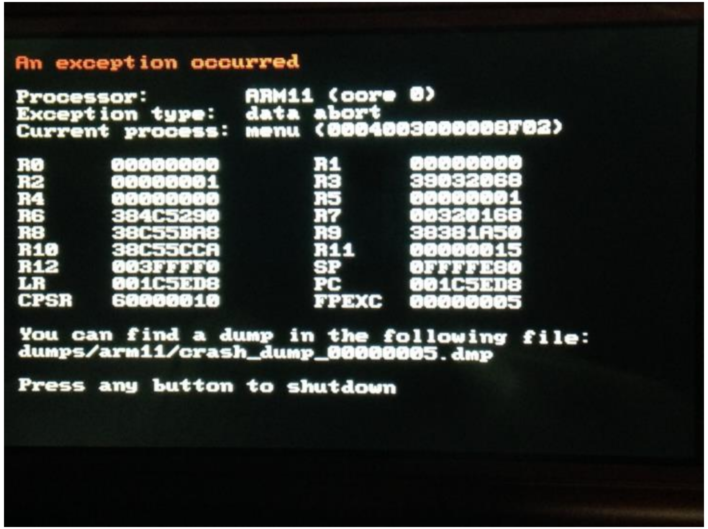


Exceptions

- Traps (Ловушки)
- Reported immediately following the execution of the trapping instruction;
- after the kernel returns control to the program, it is allowed to continue its execution with no loss of continuity.
- The saved value of eip is the address of the instruction that should be executed after the one that caused the trap. A trap is triggered only when there is no need to reexecute the instruction that terminated. The main use of traps is for debugging purposes.
- The role of the interrupt signal in this case is to notify the debugger that a specific instruction has been executedCan generally be corrected; once corrected, the program is allowed to restart with no loss of continuity.
- The saved value of eip is the address of the instruction that caused the fault, and hence that instruction can be resumed when the exception handler terminates.

Exceptions

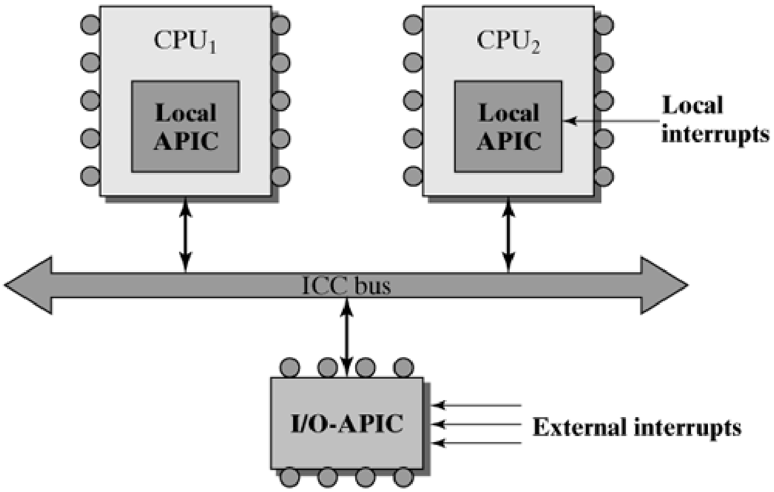
- Aborts (**Аварии**)
- A serious error occurred;
- The control unit is in trouble, and it may be unable to store in the eip register the precise location of the instruction causing the exception.
- Aborts are used to report severe errors, such as hardware failures and invalid or inconsistent values in system tables.
- The interrupt signal sent by the control unit is an emergency signal used to switch control to the corresponding abort exception handler.
- This handler has no choice but to force the affected process to terminate.



Exceptions

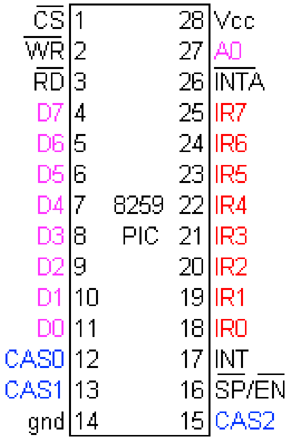
- Programmed exceptions
- Occur at the request of the programmer. They are triggered by int or int3 instructions; the into (check for overflow) and bound (check on address bound) instructions also give rise to a programmed exception when the condition they are checking is not true.
- Programmed exceptions are handled by the control unit as traps; they are often called software interrupts .

APIC

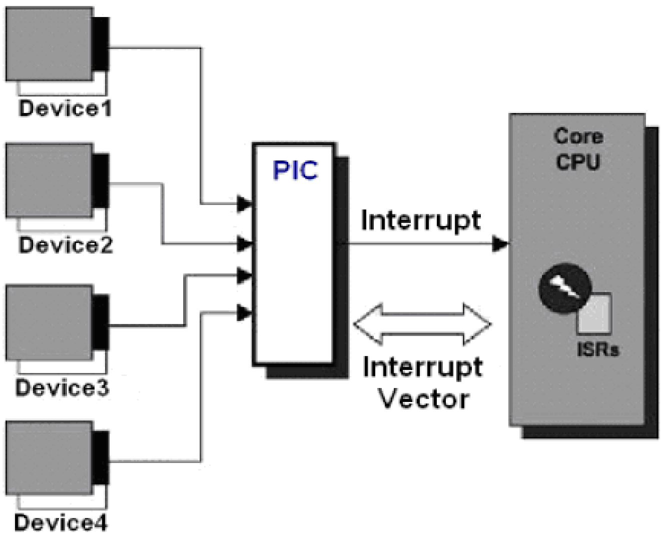


APIC - ***Advanced Programmable Interrupt Controller***)

APIC was used in multi-core / multi-processor systems, starting with Intel Pentium (core P54). Starting with this processor, each one was supplied with an integrated Local APIC.



PIC



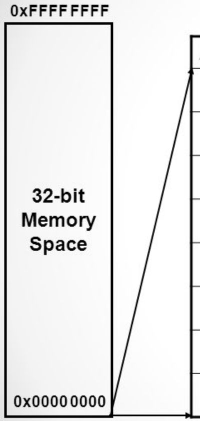
PIC (***Programmable Interrupt Controller***

APIC was used in multi-core / multi-processor systems, starting with Intel Pentium (core P54). Starting with this processor, each one was supplied with an integrated Local APIC.

Vector table

ARM

The ARM Vector table



Address	Vector
0x0000 0000	RESET
0x0000 0004	Undefined Instruction
0x0000 0008	SWI Software Interrupt
0x0000 000C	Abort (Prefetch) Bus Error Inst Fetch
0x0000 0010	Abort (Data) Bus Error Data Fetch
0x0000 0014	Reserved (not used)
0x0000 0018	IRQ Interrupt
0x0000 001C	FIQ Interrupt

AVR

16.1. Interrupt Vectors in ATmega328/P

Table 16-1. Reset and Interrupt Vectors in ATmega328/P

Vector No	Program Address[2]	Source	Interrupts definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event

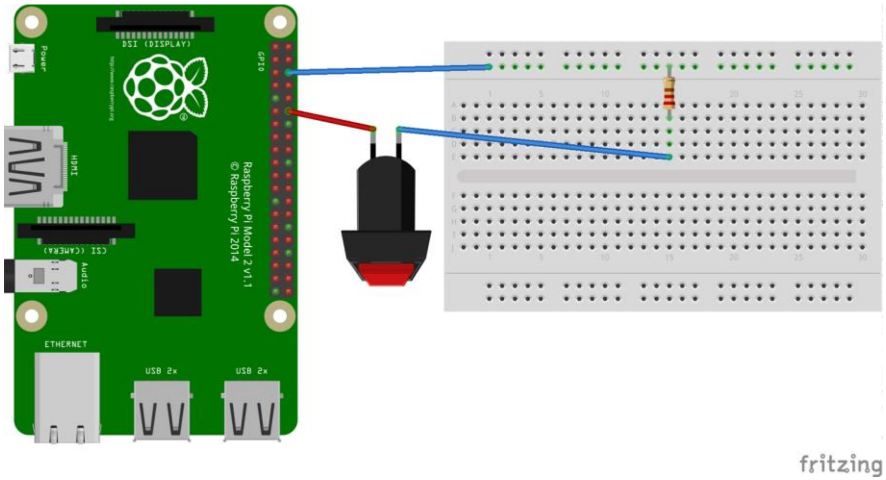
Vector table

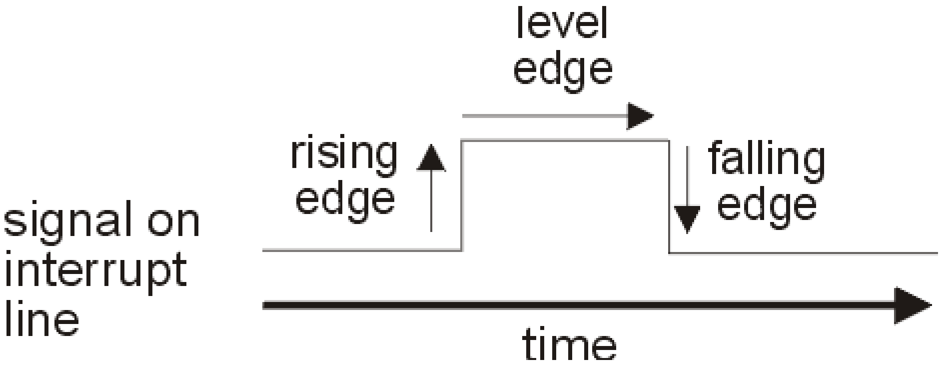
Interrupt Vectors in Linux

Vector range	Use
0-19 (0x0-0x13)	Nonmaskable interrupts and exceptions
20-31 (0x14-0x1f)	Intel-reserved
32-127 (0x20-0x7f)	External interrupts (*IRQ*s)
128 (0x80)	Programmed exception for system calls (see Chapter 10)
129-238 (0x81-0xee)	External interrupts (*IRQ*s)
239 (0xef)	Local **APIC** timer interrupt (see Chapter 6)
240 (0xf0)	Local **APIC** thermal interrupt (introduced in the Pentium 4 models)
241-250 (0xf1-0xfa)	Reserved by Linux for future use
251-253 (0xfb-0xfd)	Interprocessor interrupts (see the section "Interprocessor Interrupt Handling" later in this chapter)
254 (0xfe)	Local **APIC** error interrupt (generated when the local **APIC** detects an erroneous condition)
255 (0xff)	Local **APIC** spurious interrupt (generated if the **CPU** masks an interrupt while the hardware device raises it)

Interrupts

Interrupts control:





Interrupts

Interrupts control:
https://lxr.missinglinkelectronics.com/linux/include/linux/interrupt.h

- include/linux/interrupt.h
    - **disable_irq**()
    - **enable_irq**()
    - etc.
- include/linux/irqflags.h
    - **local_irq_disable**()
    - **local_irq_enable**()
    - **local_irq_save**(flags)

Procfs interface:

- /proc/interrupts
- /proc/irq/

Interrupts

- /proc/stat
- /proc/irq

```
geeko@da1:~/Documents> cat /proc/interrupts
           CPU0
  0:         91   IO-APIC-edge      timer
  1:       1708   IO-APIC-edge      i8042
  3:          1   IO-APIC-edge
  4:          1   IO-APIC-edge
  6:          5   IO-APIC-edge      floppy
  7:          0   IO-APIC-edge      parport0
  8:          0   IO-APIC-edge      rtc0
  9:          0   IO-APIC-fasteoi   acpi
 12:       3608   IO-APIC-edge      i8042
 14:          0   IO-APIC-edge      ata_piix
 15:      61463   IO-APIC-edge      ata_piix
 16:          0   IO-APIC-fasteoi   ehci_hcd:usb1
 17:      17659   IO-APIC-fasteoi   ioc0
 18:       1124   IO-APIC-fasteoi   vmxnet ether
 19:          0   IO-APIC-fasteoi   vmci, uhci_hcd:usb2
NMI:          0   Non-maskable interrupts
LOC:     612282   Local timer interrupts
RES:          0   Rescheduling interrupts
CAL:          0   function call interrupts
TLB:          0   TLB shootdowns
TRM:          0   Thermal event interrupts
SPU:          0   Spurious interrupts
```

IRQ interface

Interrupts control:

- include/linux/interrupt.h
  - **disable_irq**()
  - **enable_irq**()
  - etc.
- include/linux/irqflags.h
  - **local_irq_disable**()
  - **local_irq_enable**()
  - **local_irq_save**(flags)

Procfs interface:

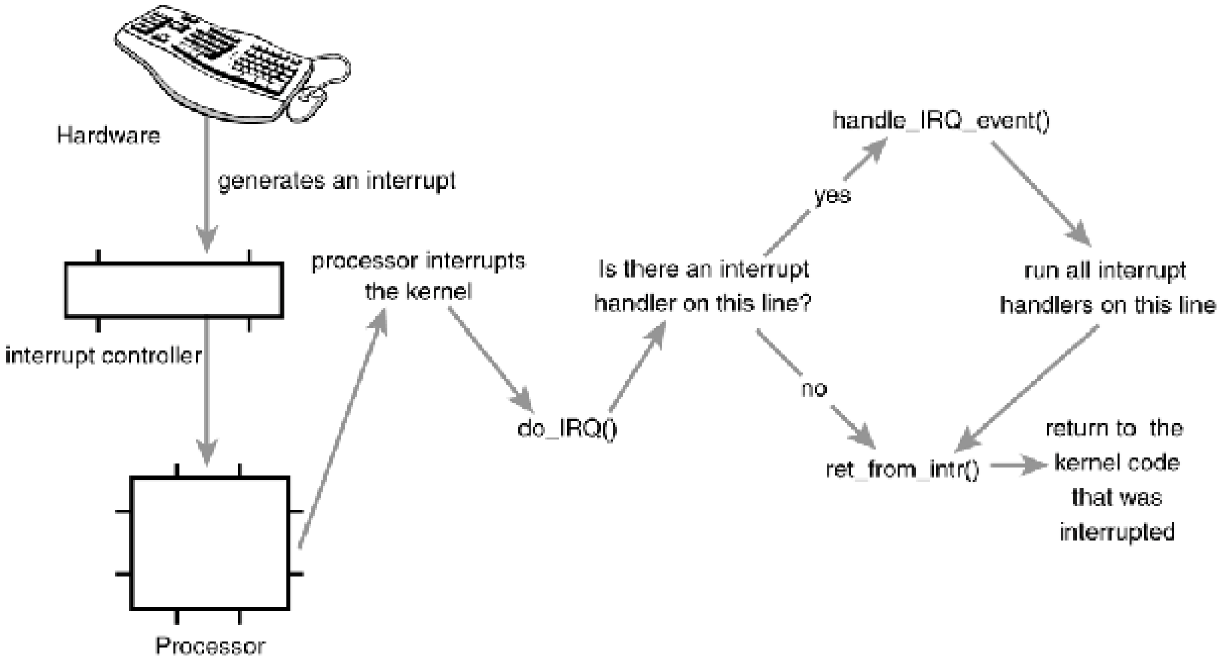
- /proc/interrupts
- /proc/irq/

IRQ interface

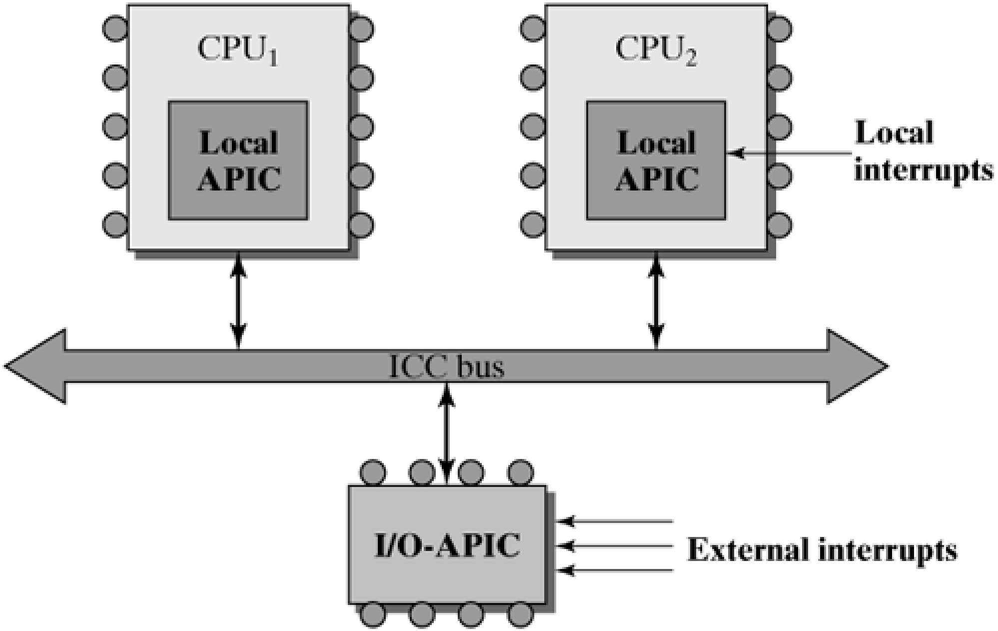
Interrupts control:

- include/linux/interrupt.h
  - **disable_irq**()
  - **enable_irq**()
  - etc.
- include/linux/irqflags.h
  - **local_irq_disable**() - запретить прерывания на локальном CPU;
  - **local_irq_enable**() - разрешить прерывания на локальном CPU
  - **local_irq_save**(flags)
  - **int irqs_disabled()** - возвратить ненулевое значение, если запрещены прерывания на локальном CPU, в противном случае возвращается нуль;

IRQ interface



APIC


CPU$_1$
Local APIC
CPU$_2$
Local APIC
Local interrupts
ICC bus
I/O-APIC
External interrupts

Interrupt handler

- include/linux/interrupt.h
  - typedef **irqreturn_t** (***irq_handler_t**)(int, void *)
  - int **request_irq**(unsigned int irq, irq_handler_t handler, unsigned long flags, const char *name, void *dev)
  - void **free_irq**(unsigned int, void *)

- **irq** - номер линии запрашиваемого прерывания.
- **handler** - указатель на функцию-обработчик.
- **flags** - битовая маска опций (описываемая далее), связанная с управлением прерыванием.
- **name** - символьная строка, используемая в /proc/interrupts, для отображения владельца прерывания.
- **dev** - указатель на уникальный идентификатор устройства на линии IRQ, для не разделяемых прерываний (например шины ISA) может указываться NULL. Данные по указателю dev требуются для удаления только специфицируемого устройства на разделяемой линии IRQ.

Interrupt handler

- Для линии **IRQ** регистрируется функция обработчика «верхней половины» (это та же ISR функция по смыслу, «верхняя половина» обработчика), который выполняется при запрещённых прерываниях локального процессора. Именно этой функции передаётся управление при возникновении аппаратного прерывания.
- Синтаксически функция-обработчик должна иметь строго описанный функциональный тип **irq_handler_t**, и возвращает управление ядру системы традиционным **return**, с возвращаемым значением **IRQ_NONE** или **IRQ_HANDLED**.
- При возникновении аппаратного прерывания по линии **IRQ функция-обработчик получит управление**. Эта функция выполняется в контексте прерывания — это одно из самых важных ограничений, накладываемых Linux, мы не раз будем возвращаться к нему. Перед своим завершением функция-обработчик регистрирует для последующего выполнения функцию нижней половины обработчика, которая и завершит позже начатую работу по обработке этого прерывания...

Interrupt handler

- **При возникновении аппаратного прерывания по линии IRQ** функция-обработчик получит управление. Эта функция выполняется в контексте прерывания — это одно из самых важных ограничений, накладываемых Linux, мы не раз будем возвращаться к нему. Перед своим завершением функция-обработчик регистрирует для последующего выполнения функцию нижней половины обработчика, которая и завершит позже начатую работу по обработке этого прерывания...
- В этой точке (после return из обработчика верхней половины) **ядро завершает всё взаимодействие с аппаратурой контроллера прерываний, разрешает последующие прерывания, восстанавливает контроллер командой завершения обработки прерывания** (посылает EOI) и возвращает управление из прерывания (из ядра!) уже именно командой iret. После этого будет восстановлен контекст прерванного процесса (потока).
- **А вот запланированная выше к выполнению функция нижней половины будет вызвана ядром в некоторый момент позже** (часто это может быть и непосредственно после завершения return из верхней половины, но это непредсказуемо), тогда, когда удобнее будет ядру системы. Принципиально важное отличие функции нижней половины состоит в том, что она выполняется уже при разрешённых прерываниях.

free_irq()

- Предупреждение относительно удаления обработчика free_irq()
- Если же в модуле при его завершении (выгрузке) вы не выполните явно free_irq(), то почти со 100% вероятностью произойдёт следующее:
- модуль будет выгружен, но вектор прерывания будет установлен на тот адрес, который перед тем занимала зарегистрированная функция **handler()** ...
- по истечению некоторого времени эта **область памяти будет переписана ядром** под какие-то иные цели...
- и **первое же произошедшее после этого аппаратное прерывание по этой линии IRQ** приведёт к немедленному краху всей системы.

Softirq

All vectors are registered in **softirq_vec** array.
Kernel uses per-CPU threads **ksoftirqd** to launch deferred interrupts.

- include/linux/interrupt.h
  - struct **softirq_action**
  - void (***action**)(struct softirq_action *)
  - void **open_softirq**(int nr, void (*action)(struct softirq_action *))
  - void **raise_softirq**(unsigned int nr)

Procfs interface:

- /proc/softirqs

Softirq

- Создание нового уровня softirq

1. Определить новый индекс (уровень) отложенного прерывания, вписав (файл <linux/interrupt.h>) свою константу вида XXX_SOFT_IRQ в перечисление, где-то, очевидно, на одну позицию выше **TASKLET_SOFTIRQ** (иначе зачем переопределять новый уровень и не использовать тасклет?).
2. Во время инициализации модуля должен быть зарегистрирован (объявлен) обработчик отложенного прерывания с помощью вызова **open_softirq()**, который принимает три параметра: этот индекс отложенного прерывания, функция-обработчик и значение поля **data**
3. Функция-обработчик отложенного прерывания должна в точности соответствовать правильному прототипу:
void xxx_analyze(unsigned long data);

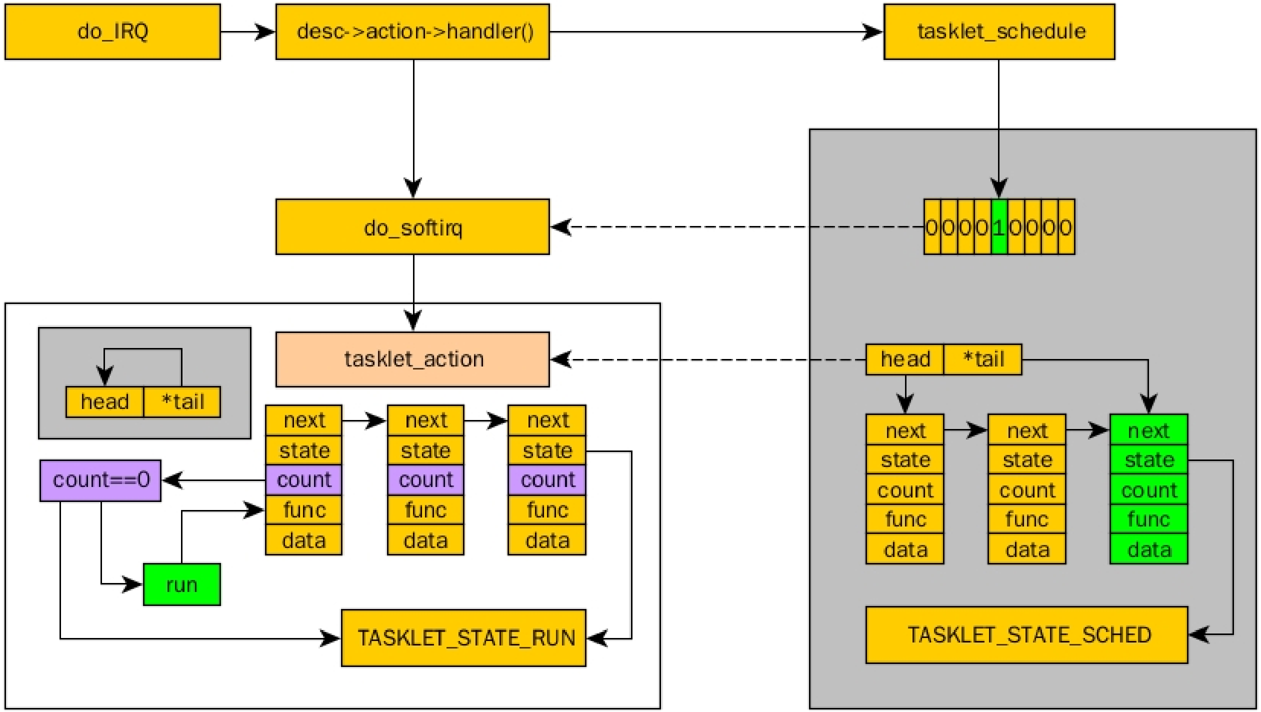
1. Зарегистрированное отложенное прерывание.
2. Затем, в подходящий (не для вас, для системы) момент времени отложенное прерывание начнёт выполнятся

Tasklet

Tasklets are simplified interface to schedule deferred tasks.
It’s implemented as softirqs (TASKLET_SOFTIRQ or HI_SOFTIRQ).

- include/linux/interrupt.h
  - struct **tasklet_struct**
  - void **tasklet_init**(struct tasklet_struct *t, void (*func)(unsigned long), unsigned long data)
    - **DECLARE_TASKLET**(name, func, data)
    - **DECLARE_TASKLET_DISABLED**(name, func, data)
  - void **tasklet_schedule**(struct tasklet_struct *t)
    - void **tasklet_hi_schedule**(struct tasklet_struct *t)
  - void **tasklet_disable**(struct tasklet_struct *t)
  - void **tasklet_enable**(struct tasklet_struct *t)
  - void **tasklet_kill**(struct tasklet_struct *t)

Tasklet



do_IRQ
desc->action->handler()
tasklet_schedule
do_softirq
0 0 0 0 1 0 0 0 0
tasklet_action
head
*tail
head
*tail
next
state
count
func
data
next
state
count
func
data
next
state
count
func
data
count==0
run
TASKLET_STATE_RUN
next
state
count
func
data
next
state
count
func
data
next
state
count
func
data
TASKLET_STATE_SCHED

Workqueue

The workqueue is another concept for handling deferred functions.
Workqueue functions run in the context of a kernel process.
Kernel provides per-cpu threads (**kworker**) to launch scheduled works.

- include/linux/workqueue.h
  - struct **work_struct**
    - struct **delayed_work**
  - INIT_WORK(_work, _func)
  - DECLARE_WORK(n, f)
    - DECLARE_DELAYED_WORK(n, f)
    - DECLARE_DEFERRABLE_WORK(n, f)
  - bool **schedule_work**(struct work_struct *work)
    - bool **schedule_delayed_work**(struct delayed_work *dwork, unsigned long delay)
  - void **flush_scheduled_work**(void)
  - bool **cancel_delayed_work**(struct delayed_work *dwork)

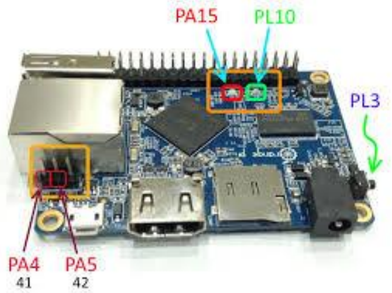
Custom workqueues

You can create your own work queues instead of using global ones:

- include/linux/workqueue.h
  - struct workqueue_struct *create_workqueue(name)
  - void destroy_workqueue(struct workqueue_struct *)
  - bool **queue_work**(struct workqueue_struct *wq, struct work_struct *work)
  - bool **queue_delayed_work**(struct workqueue_struct *wq, struct delayed_work *dwork, unsigned long delay)
  - bool **queue_work_on**(int cpu, struct workqueue_struct *wq, struct work_struct *work)

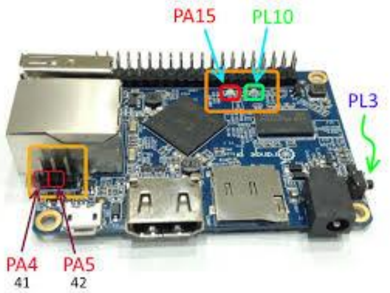
Interrupts

- gpv@orangepione:/sys/class/gpio$ cat /proc/interrupts
- CPU0 CPU1 CPU2 CPU3
- 17: 0 0 0 0 GICv2 50 Level /soc/timer@01c20c00
- 18: 0 0 0 0 GICv2 29 Level arch_timer
- 19: 119798 41591 52656 25090 GICv2 30 Level arch_timer
- 22: 0 0 0 0 GICv2 120 Level 1ee0000.hdmi, dw-hdmi-cec
- 23: 0 0 0 0 GICv2 82 Level 1c02000.dma-controller
- 24: 0 0 0 0 GICv2 118 Level 1c0c000.lcd-controller
- 25: 250724 0 0 0 GICv2 92 Level sunxi-mmc
- 26: 0 0 0 0 GICv2 103 Level musb-hdrc.1.auto
- 27: 0 0 0 0 GICv2 104 Level ehci_hcd:usb1
- 28: 0 0 0 0 GICv2 105 Level ohci_hcd:usb2
- 29: 0 0 0 0 GICv2 106 Level ehci_hcd:usb3
- 30: 0 0 0 0 GICv2 107 Level ohci_hcd:usb4
- 34: 23984 0 0 0 GICv2 114 Level eth0
- 37: 2034 0 0 0 GICv2 32 Level ttyS0
- 38: 0 0 0 0 GICv2 72 Level 1f00000.rtc
- 92: 1 0 0 0 sunxi_pio_edge 44 Edge usb0-id-det
- IPI0: 0 0 0 0 CPU wakeup interrupts
- IPI1: 0 0 0 0 Timer broadcast interrupts
- IPI2: 10088 34948 15124 13968 Rescheduling interrupts
- IPI3: 13 13 11 11 Function call interrupts
- IPI4: 0 0 0 0 CPU stop interrupts
- IPI5: 5030 1574 2250 311 IRQ work interrupts
- IPI6: 0 0 0 0 completion interrupts
- Err: 0
- gpv@orangepione:/sys/class/gpio$



Interrupts

- cat /sys/kernel/debug/gpio
- gpiochip0: GPIOs 0-223, parent: platform/1c20800.pinctrl, 1c20800.pinctrl:
- gpio-13 (|sysfs) out hi
- gpio-15 (|orangepi:red:status) out lo
- gpio-16 (|sysfs) out lo
- gpio-166 (|cd) in lo
- gpio-204 (|usb0_id_det) in hi IRQ



- gpiochip1: GPIOs 352-383, parent: platform/1f02c00.pinctrl, 1f02c00.pinctrl:
- gpio-354 (|usb0-vbus) out lo
- gpio-358 (|?) out lo
- gpio-362 (|orangepi:green:pwr) out hi
- root@orangepione:/sys#

Do It

- http://blablacode.ru/yadro-linux/540
- http://blablacode.ru/yadro-linux/543
- https://github.com/lamazavr/linux_kernel_mod
- https://github.com/lamazavr/linux_kernel_mod/tree/master/blablamod_interrupts
- See https://www.youtube.com/watch?v=ltCZydX_zmk&t=775s
- See https://drive.google.com/drive/folders/1wgZfagxHicM8iz1yVCGtRF5sQxXpKZIw

Do It

- **Создание "нижней половины" обработчика прерываний**
  https://www.ibm.com/developerworks/ru/library/l-linux_kernel_59/index.html
- **Создание "верхней половины" обработчика прерываний**
  https://www.ibm.com/developerworks/ru/library/l-linux_kernel_58/
- **Тасклеты и очереди отложенных действий**
  https://www.ibm.com/developerworks/ru/library/l-linux_kernel_60/