Linux generic IRQ handling

Thomas Gleixner

tglx@linutronix.de

Ingo Molnar

mingo@elte.hu

Linux generic IRQ handling

by Thomas Gleixner and Ingo Molnar

Copyright © 2005-2010 Thomas Gleixner
Copyright © 2005-2006 Ingo Molnar

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License version 2 as published by the Free Software Foundation.
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
For more details see the file COPYING in the source distribution of Linux.

Table of Contents

1. Introduction **1**
2. Rationale **3**
3. Known Bugs And Assumptions **5**
4. Abstraction layers **7**
4.1. Interrupt control flow 7
4.2. Highlevel Driver API 7
4.3. Highlevel IRQ flow handlers 8
4.3.1. Default flow implementations 8
4.3.1.1. Helper functions 8
4.3.2. Default flow handler implementations 9
4.3.2.1. Default Level IRQ flow handler 9
4.3.2.2. Default Fast EOI IRQ flow handler 9
4.3.2.3. Default Edge IRQ flow handler 10
4.3.2.4. Default simple IRQ flow handler 10
4.3.2.5. Default per CPU flow handler 10
4.3.2.6. EOI Edge IRQ flow handler 11
4.3.2.7. Bad IRQ flow handler 11
4.3.3. Quirks and optimizations 11
4.3.4. Delayed interrupt disable 11
4.4. Chiplevel hardware encapsulation 12
5. __do_IRQ entry point **13**
6. Locking on SMP **15**
7. Structures **17**
struct irq_data 17
struct irq_chip 18
struct irq_chip_regs 21
struct irq_chip_type 22
struct irq_chip_generic 23
enum irq_gc_flags 25
struct irqaction 26
struct irq_affinity_notify 28
8. Public Functions Provided **31**
synchronize_irq 31
irq_set_affinity_notifier 31
disable_irq_nosync 32
disable_irq 33
enable_irq 34
irq_set_irq_wake 35
setup_irq 36

remove_irq 37
free_irq 38
request_threaded_irq 39
request_any_context_irq 40
irq_set_chip 42
irq_set_irq_type 42
irq_set_handler_data 43
irq_set_chip_data 44
handle_simple_irq 45
handle_level_irq 46
9. Internal Functions Provided 49
irq_reserve_irqs 49
irq_get_next_irq 49
dynamic_irq_cleanup 50
kstat_irqs_cpu 51
kstat_irqs 52
kstat_irqs_usr 53
handle_bad_irq 53
irq_set_msi_desc 54
handle_fasteoi_irq 55
handle_edge_irq 56
handle_edge_eoi_irq 57
handle_percpu_irq 58
irq_cpu_online 59
irq_cpu_offline 60
10. Credits 61

Chapter 1. Introduction

The generic interrupt handling layer is designed to provide a complete abstraction of interrupt handling for device drivers. It is able to handle all the different types of interrupt controller hardware. Device drivers use generic API functions to request, enable, disable and free interrupts. The drivers do not have to know anything about interrupt hardware details, so they can be used on different platforms without code changes.

This documentation is provided to developers who want to implement an interrupt subsystem based for their architecture, with the help of the generic IRQ handling layer.

Chapter 2. Rationale

The original implementation of interrupt handling in Linux is using the __do_IRQ() super-handler, which is able to deal with every type of interrupt logic.

Originally, Russell King identified different types of handlers to build a quite universal set for the ARM interrupt handler implementation in Linux 2.5/2.6. He distinguished between:

- Level type
- Edge type
- Simple type

During the implementation we identified another type:

- Fast EOI type

In the SMP world of the __do_IRQ() super-handler another type was identified:

- Per CPU type

This split implementation of highlevel IRQ handlers allows us to optimize the flow of the interrupt handling for each specific interrupt type. This reduces complexity in that particular codepath and allows the optimized handling of a given type.

The original general IRQ implementation used hw_interrupt_type structures and their ->ack(), ->end() [etc.] callbacks to differentiate the flow control in the super-handler. This leads to a mix of flow logic and lowlevel hardware logic, and it also leads to unnecessary code duplication: for example in i386, there is a ioapic_level_irq and a ioapic_edge_irq irq-type which share many of the lowlevel details but have different flow handling.

A more natural abstraction is the clean separation of the 'irq flow' and the 'chip details'.

Analysing a couple of architecture's IRQ subsystem implementations reveals that most of them can use a generic set of 'irq flow' methods and only need to add the chip level specific code. The separation is also valuable for (sub)architectures which need specific quirks in the irq flow itself but not in the chip-details - and thus provides a more transparent IRQ subsystem design.

Each interrupt descriptor is assigned its own highlevel flow handler, which is normally one of the generic implementations. (This highlevel flow handler implementation also makes it simple to provide demultiplexing handlers which can be found in embedded platforms on various architectures.)

The separation makes the generic interrupt handling layer more flexible and extensible. For example, an (sub)architecture can use a generic irq-flow implementation for 'level type' interrupts and add a (sub)architecture specific 'edge type' implementation.

To make the transition to the new model easier and prevent the breakage of existing implementations, the __do_IRQ() super-handler is still available. This leads to a kind of duality for the time being. Over time the new model should be used in more and more architectures, as it enables smaller and cleaner IRQ subsystems. It's deprecated for three years now and about to be removed.

Chapter 3. Known Bugs And Assumptions

None (knock on wood).

Chapter 4. Abstraction layers

There are three main levels of abstraction in the interrupt code:

1. Highlevel driver API
2. Highlevel IRQ flow handlers
3. Chiplevel hardware encapsulation

4.1. Interrupt control flow

Each interrupt is described by an interrupt descriptor structure irq_desc. The interrupt is referenced by an 'unsigned int' numeric value which selects the corresponding interrupt decription structure in the descriptor structures array. The descriptor structure contains status information and pointers to the interrupt flow method and the interrupt chip structure which are assigned to this interrupt.

Whenever an interrupt triggers, the lowlevel arch code calls into the generic interrupt code by calling desc->handle_irq(). This highlevel IRQ handling function only uses desc->irq_data.chip primitives referenced by the assigned chip descriptor structure.

4.2. Highlevel Driver API

The highlevel Driver API consists of following functions:

- request_irq()
- free_irq()
- disable_irq()
- enable_irq()
- disable_irq_nosync() (SMP only)
- synchronize_irq() (SMP only)
- irq_set_irq_type()
- irq_set_irq_wake()

- irq_set_handler_data()
- irq_set_chip()
- irq_set_chip_data()

See the autogenerated function documentation for details.

4.3. Highlevel IRQ flow handlers

The generic layer provides a set of pre-defined irq-flow methods:

- handle_level_irq
- handle_edge_irq
- handle_fasteoi_irq
- handle_simple_irq
- handle_percpu_irq
- handle_edge_eoi_irq
- handle_bad_irq

The interrupt flow handlers (either predefined or architecture specific) are assigned to specific interrupts by the architecture either during bootup or during device initialization.

4.3.1. Default flow implementations

4.3.1.1. Helper functions

The helper functions call the chip primitives and are used by the default flow implementations. The following helper functions are implemented (simplified excerpt):

```
default_enable(struct irq_data *data)
{
 desc->irq_data.chip->irq_unmask(data);
}

default_disable(struct irq_data *data)
{
 if (!delay_disable(data))
  desc->irq_data.chip->irq_mask(data);
```

```
}

default_ack(struct irq_data *data)
{
 chip->irq_ack(data);
}

default_mask_ack(struct irq_data *data)
{
 if (chip->irq_mask_ack) {
  chip->irq_mask_ack(data);
 } else {
  chip->irq_mask(data);
  chip->irq_ack(data);
 }
}

noop(struct irq_data *data))
{
}
```

4.3.2. Default flow handler implementations

4.3.2.1. Default Level IRQ flow handler

handle_level_irq provides a generic implementation for level-triggered interrupts.

The following control flow is implemented (simplified excerpt):

```
desc->irq_data.chip->irq_mask_ack();
handle_irq_event(desc->action);
desc->irq_data.chip->irq_unmask();
```

4.3.2.2. Default Fast EOI IRQ flow handler

handle_fasteoi_irq provides a generic implementation for interrupts, which only need an EOI at the end of the handler

The following control flow is implemented (simplified excerpt):

```
handle_irq_event(desc->action);
desc->irq_data.chip->irq_eoi();
```

4.3.2.3. Default Edge IRQ flow handler

handle_edge_irq provides a generic implementation for edge-triggered interrupts.

The following control flow is implemented (simplified excerpt):

```
if (desc->status & running) {
 desc->irq_data.chip->irq_mask_ack();
 desc->status |= pending | masked;
 return;
}
desc->irq_data.chip->irq_ack();
desc->status |= running;
do {
 if (desc->status & masked)
  desc->irq_data.chip->irq_unmask();
 desc->status &= ~pending;
 handle_irq_event(desc->action);
} while (status & pending);
desc->status &= ~running;
```

4.3.2.4. Default simple IRQ flow handler

handle_simple_irq provides a generic implementation for simple interrupts.

Note: The simple flow handler does not call any handler/chip primitives.

The following control flow is implemented (simplified excerpt):

```
handle_irq_event(desc->action);
```

4.3.2.5. Default per CPU flow handler

handle_percpu_irq provides a generic implementation for per CPU interrupts.

Per CPU interrupts are only available on SMP and the handler provides a simplified version without locking.

The following control flow is implemented (simplified excerpt):

```
if (desc->irq_data.chip->irq_ack)
 desc->irq_data.chip->irq_ack();
handle_irq_event(desc->action);
if (desc->irq_data.chip->irq_eoi)
        desc->irq_data.chip->irq_eoi();
```

4.3.2.6. EOI Edge IRQ flow handler

handle_edge_eoi_irq provides an abnomination of the edge handler which is solely used to tame a badly wreckaged irq controller on powerpc/cell.

4.3.2.7. Bad IRQ flow handler

handle_bad_irq is used for spurious interrupts which have no real handler assigned..

4.3.3. Quirks and optimizations

The generic functions are intended for 'clean' architectures and chips, which have no platform-specific IRQ handling quirks. If an architecture needs to implement quirks on the 'flow' level then it can do so by overriding the highlevel irq-flow handler.

4.3.4. Delayed interrupt disable

This per interrupt selectable feature, which was introduced by Russell King in the ARM interrupt implementation, does not mask an interrupt at the hardware level when disable_irq() is called. The interrupt is kept enabled and is masked in the flow handler when an interrupt event happens. This prevents losing edge interrupts on hardware which does not store an edge interrupt event while the interrupt is disabled at the hardware level. When an interrupt arrives while the

IRQ_DISABLED flag is set, then the interrupt is masked at the hardware level and the IRQ_PENDING bit is set. When the interrupt is re-enabled by enable_irq() the pending bit is checked and if it is set, the interrupt is resent either via hardware or by a software resend mechanism. (It's necessary to enable CONFIG_HARDIRQS_SW_RESEND when you want to use the delayed interrupt disable feature and your hardware is not capable of retriggering an interrupt.) The delayed interrupt disable is not configurable.

4.4. Chiplevel hardware encapsulation

The chip level hardware descriptor structure irq_chip contains all the direct chip relevant functions, which can be utilized by the irq flow implementations.

- irq_ack()
- irq_mask_ack() - Optional, recommended for performance
- irq_mask()
- irq_unmask()
- irq_eoi() - Optional, required for eoi flow handlers
- irq_retrigger() - Optional
- irq_set_type() - Optional
- irq_set_wake() - Optional

These primitives are strictly intended to mean what they say: ack means ACK, masking means masking of an IRQ line, etc. It is up to the flow handler(s) to use these basic units of lowlevel functionality.

Chapter 5. __do_IRQ entry point

The original implementation __do_IRQ() was an alternative entry point for all types of interrupts. It not longer exists.

This handler turned out to be not suitable for all interrupt hardware and was therefore reimplemented with split functionality for edge/level/simple/percpu interrupts. This is not only a functional optimization. It also shortens code paths for interrupts.

Chapter 6. Locking on SMP

The locking of chip registers is up to the architecture that defines the chip primitives. The per-irq structure is protected via desc->lock, by the generic layer.

Chapter 7. Structures

This chapter contains the autogenerated documentation of the structures which are used in the generic IRQ layer.

struct irq_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irq_data` — per irq and irq chip data passed down to chip functions

Synopsis

```
struct irq_data {
  unsigned int irq;
  unsigned int node;
  unsigned int state_use_accessors;
  struct irq_chip * chip;
  void * handler_data;
  void * chip_data;
  struct msi_desc * msi_desc;
#ifdef CONFIG_SMP
  cpumask_var_t affinity;
#endif
};
```

Members

irq

interrupt number

node

node index useful for balancing

state_use_accessors

status information for irq chip functions. Use accessor functions to deal with it

chip

low level interrupt hardware access

handler_data

per-IRQ data for the irq_chip methods

chip_data

platform-specific per-chip private data for the chip methods, to allow shared chip implementations

msi_desc

MSI descriptor

affinity

IRQ affinity on SMP

Description

The fields here need to overlay the ones in irq_desc until we cleaned up the direct references and switched everything over to irq_data.

struct irq_chip

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irq_chip` — hardware interrupt chip descriptor

Synopsis

```
struct irq_chip {
  const char * name;
  unsigned int (* irq_startup) (struct irq_data *data);
  void (* irq_shutdown) (struct irq_data *data);
  void (* irq_enable) (struct irq_data *data);
  void (* irq_disable) (struct irq_data *data);
  void (* irq_ack) (struct irq_data *data);
  void (* irq_mask) (struct irq_data *data);
  void (* irq_mask_ack) (struct irq_data *data);
  void (* irq_unmask) (struct irq_data *data);
  void (* irq_eoi) (struct irq_data *data);
  int (* irq_set_affinity) (struct irq_data *data, const struct cpumask *c
  int (* irq_retrigger) (struct irq_data *data);
  int (* irq_set_type) (struct irq_data *data, unsigned int flow_type);
  int (* irq_set_wake) (struct irq_data *data, unsigned int on);
  void (* irq_bus_lock) (struct irq_data *data);
  void (* irq_bus_sync_unlock) (struct irq_data *data);
  void (* irq_cpu_online) (struct irq_data *data);
  void (* irq_cpu_offline) (struct irq_data *data);
  void (* irq_suspend) (struct irq_data *data);
  void (* irq_resume) (struct irq_data *data);
  void (* irq_pm_shutdown) (struct irq_data *data);
  void (* irq_print_chip) (struct irq_data *data, struct seq_file *p);
  unsigned long flags;
#ifdef CONFIG_IRQ_RELEASE_METHOD
  void (* release) (unsigned int irq, void *dev_id);
#endif
};
```

Members

name

name for /proc/interrupts

irq_startup

start up the interrupt (defaults to ->enable if NULL)

irq_shutdown

shut down the interrupt (defaults to ->disable if NULL)

irq_enable

enable the interrupt (defaults to chip->unmask if NULL)

irq_disable

disable the interrupt

irq_ack

start of a new interrupt

irq_mask

mask an interrupt source

irq_mask_ack

ack and mask an interrupt source

irq_unmask

unmask an interrupt source

irq_eoi

end of interrupt

irq_set_affinity

set the CPU affinity on SMP machines

irq_retrigger

resend an IRQ to the CPU

irq_set_type

set the flow type (IRQ_TYPE_LEVEL/etc.) of an IRQ

irq_set_wake

enable/disable power-management wake-on of an IRQ

irq_bus_lock

function to lock access to slow bus (i2c) chips

irq_bus_sync_unlock

function to sync and unlock slow bus (i2c) chips

irq_cpu_online

configure an interrupt source for a secondary CPU

irq_cpu_offline
: un-configure an interrupt source for a secondary CPU

irq_suspend
: function called from core code on suspend once per chip

irq_resume
: function called from core code on resume once per chip

irq_pm_shutdown
: function called from core code on shutdown once per chip

irq_print_chip
: optional to print special chip info in show_interrupts

flags
: chip specific flags

release
: release function solely used by UML

struct irq_chip_regs

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irq_chip_regs` — register offsets for struct irq_gci

Synopsis

```
struct irq_chip_regs {
  unsigned long enable;
  unsigned long disable;
  unsigned long mask;
```

```
  unsigned long ack;
  unsigned long eoi;
  unsigned long type;
  unsigned long polarity;
};
```

Members

enable

Enable register offset to reg_base

disable

Disable register offset to reg_base

mask

Mask register offset to reg_base

ack

Ack register offset to reg_base

eoi

Eoi register offset to reg_base

type

Type configuration register offset to reg_base

polarity

Polarity configuration register offset to reg_base

struct irq_chip_type

LINUX

Name

`struct irq_chip_type` — Generic interrupt chip instance for a flow type

Synopsis

```
struct irq_chip_type {
  struct irq_chip chip;
  struct irq_chip_regs regs;
  irq_flow_handler_t handler;
  u32 type;
};
```

Members

chip

The real interrupt chip which provides the callbacks

regs

Register offsets for this chip

handler

Flow handler associated with this chip

type

Chip can handle these flow types

Description

A irq_generic_chip can have several instances of irq_chip_type when it requires different functions and register offsets for different flow types.

struct irq_chip_generic

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irq_chip_generic` — Generic irq chip data structure

Synopsis

```
struct irq_chip_generic {
  raw_spinlock_t lock;
  void __iomem * reg_base;
  unsigned int irq_base;
  unsigned int irq_cnt;
  u32 mask_cache;
  u32 type_cache;
  u32 polarity_cache;
  u32 wake_enabled;
  u32 wake_active;
  unsigned int num_ct;
  void * private;
  struct list_head list;
  struct irq_chip_type chip_types[0];
};
```

Members

lock

Lock to protect register and cache data access

reg_base

Register base address (virtual)

irq_base

Interrupt base nr for this chip

irq_cnt

Number of interrupts handled by this chip

mask_cache

Cached mask register

type_cache

Cached type register

polarity_cache

Cached polarity register

wake_enabled

Interrupt can wakeup from suspend

wake_active

Interrupt is marked as an wakeup from suspend source

num_ct

Number of available irq_chip_type instances (usually 1)

private

Private data for non generic chip callbacks

list

List head for keeping track of instances

chip_types[0]

Array of interrupt irq_chip_types

Description

Note, that irq_chip_generic can have multiple irq_chip_type implementations which can be associated to a particular irq line of an irq_chip_generic instance. That allows to share and protect state in an irq_chip_generic instance when we need to implement different flow mechanisms (level/edge) for it.

enum irq_gc_flags

LINUX

Kernel Hackers ManualNovember 2026

Name

`enum irq_gc_flags` — Initialization flags for generic irq chips

Synopsis

```
enum irq_gc_flags {
  IRQ_GC_INIT_MASK_CACHE,
  IRQ_GC_INIT_NESTED_LOCK
};
```

Constants

IRQ_GC_INIT_MASK_CACHE

Initialize the mask_cache by reading mask reg

IRQ_GC_INIT_NESTED_LOCK

Set the lock class of the irqs to nested for irq chips which need to call `irq_set_wake` on the parent irq. Usually GPIO implementations

struct irqaction

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irqaction` — per interrupt action descriptor

Synopsis

```
struct irqaction {
  irq_handler_t handler;
  unsigned long flags;
  void * dev_id;
  struct irqaction * next;
  int irq;
  irq_handler_t thread_fn;
  struct task_struct * thread;
  unsigned long thread_flags;
  unsigned long thread_mask;
  const char * name;
  struct proc_dir_entry * dir;
};
```

Members

handler

interrupt handler function

flags

flags (see IRQF_* above)

dev_id

cookie to identify the device

next

pointer to the next irqaction for shared interrupts

irq

interrupt number

thread_fn

interrupt handler function for threaded interrupts

thread

thread pointer for threaded interrupts

thread_flags

flags related to *thread*

thread_mask

bitmask for keeping track of *thread* activity

name

name of the device

dir

pointer to the proc/irq/NN/name entry

struct irq_affinity_notify

LINUX

Kernel Hackers ManualNovember 2026

Name

`struct irq_affinity_notify` — context for notification of IRQ affinity changes

Synopsis

```
struct irq_affinity_notify {
  unsigned int irq;
  struct kref kref;
  struct work_struct work;
  void (* notify) (struct irq_affinity_notify *, const cpumask_t *mask);
  void (* release) (struct kref *ref);
};
```

Members

irq

Interrupt to which notification applies

kref

Reference count, for internal use

work

Work item, for internal use

notify

Function to be called on change. This will be called in process context.

release

Function to be called on release. This will be called in process context. Once registered, the structure must only be freed when this function is called or later.

Chapter 8. Public Functions Provided

This chapter contains the autogenerated documentation of the kernel API functions which are exported.

synchronize_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`synchronize_irq` — wait for pending IRQ handlers (on other CPUs)

Synopsis

```
void synchronize_irq (unsigned int irq);
```

Arguments

irq

interrupt number to wait for

Description

This function waits for any pending IRQ handlers for this interrupt to complete before returning. If you use this function while holding a resource the IRQ handler may need you will deadlock.

This function may be called - with care - from IRQ context.

irq_set_affinity_notifier

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_affinity_notifier` — control notification of IRQ affinity changes

Synopsis

```
int irq_set_affinity_notifier (unsigned int irq, struct
irq_affinity_notify * notify);
```

Arguments

irq

Interrupt for which to enable/disable notification

notify

Context for notification, or `NULL` to disable notification. Function pointers must be initialised; the other fields will be initialised by this function.

Description

Must be called in process context. Notification may only be enabled after the IRQ is allocated and must be disabled before the IRQ is freed using `free_irq`.

disable_irq_nosync

LINUX

Kernel Hackers ManualNovember 2026

Name

`disable_irq_nosync` — disable an irq without waiting

Synopsis

```
void disable_irq_nosync (unsigned int irq);
```

Arguments

irq

Interrupt to disable

Description

Disable the selected interrupt line. Disables and Enables are nested. Unlike `disable_irq`, this function does not ensure existing instances of the IRQ handler have completed before returning.

This function may be called from IRQ context.

disable_irq

LINUX

Name

`disable_irq` — disable an irq and wait for completion

Synopsis

```
void disable_irq (unsigned int irq);
```

Arguments

`irq`

Interrupt to disable

Description

Disable the selected interrupt line. Enables and Disables are nested. This function waits for any pending IRQ handlers for this interrupt to complete before returning. If you use this function while holding a resource the IRQ handler may need you will deadlock.

This function may be called - with care - from IRQ context.

enable_irq

LINUX

Name

`enable_irq` — enable handling of an irq

Synopsis

```
void enable_irq (unsigned int irq);
```

Arguments

irq

Interrupt to enable

Description

Undoes the effect of one call to `disable_irq`. If this matches the last disable, processing of interrupts on this IRQ line is re-enabled.

This function may be called from IRQ context only when desc->irq_data.chip->bus_lock and desc->chip->bus_sync_unlock are NULL !

irq_set_irq_wake

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_irq_wake` — control irq power management wakeup

Synopsis

```
int irq_set_irq_wake (unsigned int irq, unsigned int on);
```

Arguments

irq

interrupt to control

on

enable/disable power management wakeup

Description

Enable/disable power management wakeup mode, which is disabled by default. Enables and disables must match, just as they match for non-wakeup mode support.

Wakeup mode lets this IRQ wake the system from sleep states like “suspend to RAM”.

setup_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`setup_irq` — setup an interrupt

Synopsis

```
int setup_irq (unsigned int irq, struct irqaction * act);
```

Arguments

irq

Interrupt line to setup

act

irqaction for the interrupt

Description

Used to statically setup interrupts in the early boot process.

remove_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`remove_irq` — free an interrupt

Synopsis

```
void remove_irq (unsigned int irq, struct irqaction * act);
```

Arguments

irq

Interrupt line to free

act

irqaction for the interrupt

Description

Used to remove interrupts statically setup by the early boot process.

free_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`free_irq` — free an interrupt allocated with request_irq

Synopsis

```
void free_irq (unsigned int irq, void * dev_id);
```

Arguments

irq

Interrupt line to free

dev_id

Device identity to free

Description

Remove an interrupt handler. The handler is removed and if the interrupt line is no longer in use by any driver it is disabled. On a shared IRQ the caller must ensure the interrupt is disabled on the card it drives before calling this function. The function does not return until any executing interrupts for this IRQ have completed.

This function must not be called from interrupt context.

request_threaded_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`request_threaded_irq` — allocate an interrupt line

Synopsis

```
int request_threaded_irq (unsigned int irq, irq_handler_t
handler, irq_handler_t thread_fn, unsigned long irqflags,
const char * devname, void * dev_id);
```

Arguments

irq

Interrupt line to allocate

handler

Function to be called when the IRQ occurs. Primary handler for threaded interrupts If NULL and thread_fn != NULL the default primary handler is installed

thread_fn

Function called from the irq handler thread If NULL, no irq thread is created

irqflags

Interrupt type flags

devname

An ascii name for the claiming device

dev_id

A cookie passed back to the handler function

Description

This call allocates interrupt resources and enables the interrupt line and IRQ handling. From the point this call is made your handler function may be invoked. Since your handler function must clear any interrupt the board raises, you must take care both to initialise your hardware and to set up the interrupt handler in the right order.

If you want to set up a threaded irq handler for your device then you need to supply *handler* and *thread_fn*. *handler* ist still called in hard interrupt context and has to check whether the interrupt originates from the device. If yes it needs to disable the interrupt on the device and return IRQ_WAKE_THREAD which will wake up the handler thread and run *thread_fn*. This split handler design is necessary to support shared interrupts.

Dev_id must be globally unique. Normally the address of the device data structure is used as the cookie. Since the handler receives this value it makes sense to use it.

If your interrupt is shared you must pass a non NULL dev_id as this is required when freeing the interrupt.

Flags

IRQF_SHARED Interrupt is shared IRQF_TRIGGER_* Specify active edge(s) or level

request_any_context_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`request_any_context_irq` — allocate an interrupt line

Synopsis

```
int request_any_context_irq (unsigned int irq, irq_handler_t
handler, unsigned long flags, const char * name, void *
dev_id);
```

Arguments

irq

Interrupt line to allocate

handler

Function to be called when the IRQ occurs. Threaded handler for threaded interrupts.

flags

Interrupt type flags

name

An ascii name for the claiming device

dev_id

A cookie passed back to the handler function

Description

This call allocates interrupt resources and enables the interrupt line and IRQ handling. It selects either a hardirq or threaded handling method depending on the context.

On failure, it returns a negative value. On success, it returns either IRQC_IS_HARDIRQ or IRQC_IS_NESTED.

irq_set_chip

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_chip` — set the irq chip for an irq

Synopsis

```
int irq_set_chip (unsigned int irq, struct irq_chip * chip);
```

Arguments

`irq`

irq number

`chip`

pointer to irq chip description structure

irq_set_irq_type

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_irq_type` — set the irq trigger type for an irq

Synopsis

```
int irq_set_irq_type (unsigned int irq, unsigned int type);
```

Arguments

irq

irq number

type

IRQ_TYPE_{LEVEL,EDGE}_* value - see include/linux/irq.h

irq_set_handler_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_handler_data` — set irq handler data for an irq

Synopsis

```
int irq_set_handler_data (unsigned int irq, void * data);
```

Arguments

irq

Interrupt number

data

Pointer to interrupt specific data

Description

Set the hardware irq controller data for an irq

irq_set_chip_data

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_set_chip_data` — set irq chip data for an irq

Synopsis

```
int irq_set_chip_data (unsigned int irq, void * data);
```

Arguments

irq

Interrupt number

data

Pointer to chip specific data

Description

Set the hardware irq chip data for an irq

handle_simple_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`handle_simple_irq` — Simple and software-decoded IRQs.

Synopsis

```
void handle_simple_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

desc

the interrupt description structure for this irq

Description

Simple interrupts are either sent from a demultiplexing interrupt handler or come from hardware, where no interrupt hardware control is necessary.

Note

The caller is expected to handle the ack, clear, mask and unmask issues if necessary.

handle_level_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`handle_level_irq` — Level type irq handler

Synopsis

```
void handle_level_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

`desc`

the interrupt description structure for this irq

Description

Level type interrupts are active as long as the hardware line has the active level. This may require to mask the interrupt and unmask it after the associated handler has acknowledged the device, so the interrupt line is back to inactive.

Chapter 9. Internal Functions Provided

This chapter contains the autogenerated documentation of the internal functions.

irq_reserve_irqs

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_reserve_irqs` — mark irqs allocated

Synopsis

```
int irq_reserve_irqs (unsigned int from, unsigned int cnt);
```

Arguments

`from`

mark from irq number

`cnt`

number of irqs to mark

Description

Returns 0 on success or an appropriate error code

irq_get_next_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_get_next_irq` — get next allocated irq number

Synopsis

```
unsigned int irq_get_next_irq (unsigned int offset);
```

Arguments

`offset`

where to start the search

Description

Returns next irq number after offset or nr_irqs if none is found.

dynamic_irq_cleanup

LINUX

Kernel Hackers ManualNovember 2026

Name

`dynamic_irq_cleanup` — cleanup a dynamically allocated irq

Synopsis

```
void dynamic_irq_cleanup (unsigned int irq);
```

Arguments

irq

irq number to initialize

kstat_irqs_cpu

LINUX

Kernel Hackers ManualNovember 2026

Name

`kstat_irqs_cpu` — Get the statistics for an interrupt on a cpu

Synopsis

```
unsigned int kstat_irqs_cpu (unsigned int irq, int cpu);
```

Arguments

irq

The interrupt number

cpu

The cpu number

Description

Returns the sum of interrupt counts on *cpu* since boot for *irq*. The caller must ensure that the interrupt is not removed concurrently.

kstat_irqs

LINUX

Kernel Hackers ManualNovember 2026

Name

kstat_irqs — Get the statistics for an interrupt

Synopsis

```
unsigned int kstat_irqs (unsigned int irq);
```

Arguments

irq

The interrupt number

Description

Returns the sum of interrupt counts on all cpus since boot for *irq*. The caller must ensure that the interrupt is not removed concurrently.

kstat_irqs_usr

LINUX

Kernel Hackers ManualNovember 2026

Name

`kstat_irqs_usr` — Get the statistics for an interrupt

Synopsis

```
unsigned int kstat_irqs_usr (unsigned int irq);
```

Arguments

irq

The interrupt number

Description

Returns the sum of interrupt counts on all cpus since boot for *irq*. Contrary to `kstat_irqs` this can be called from any preemptible context. It's protected against concurrent removal of an interrupt descriptor when sparse irqs are enabled.

handle_bad_irq

LINUX

Name

`handle_bad_irq` — handle spurious and unhandled irqs

Synopsis

```
void handle_bad_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

desc

description of the interrupt

Description

Handles spurious and unhandled IRQ's. It also prints a debugmessage.

irq_set_msi_desc

LINUX

Name

`irq_set_msi_desc` — set MSI descriptor data for an irq

Synopsis

```
int irq_set_msi_desc (unsigned int irq, struct msi_desc *
entry);
```

Arguments

irq

Interrupt number

entry

Pointer to MSI descriptor data

Description

Set the MSI descriptor entry for an irq

handle_fasteoi_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`handle_fasteoi_irq` — irq handler for transparent controllers

Synopsis

```
void handle_fasteoi_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

desc

the interrupt description structure for this irq

Only a single callback will be issued to the chip

an ->eoi call when the interrupt has been serviced. This enables support for modern forms of interrupt handlers, which handle the flow details in hardware, transparently.

handle_edge_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

handle_edge_irq — edge type IRQ handler

Synopsis

```
void handle_edge_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

desc

the interrupt description structure for this irq

Description

Interrupt occures on the falling and/or rising edge of a hardware signal. The occurrence is latched into the irq controller hardware and must be acked in order to be reenabled. After the ack another interrupt can happen on the same source even before the first one is handled by the associated event handler. If this happens it might be necessary to disable (mask) the interrupt depending on the controller hardware. This requires to reenable the interrupt inside of the loop which handles the interrupts which have arrived while the handler was running. If all pending interrupts are handled, the loop is left.

handle_edge_eoi_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`handle_edge_eoi_irq` — edge eoi type IRQ handler

Synopsis

```
void handle_edge_eoi_irq (unsigned int irq, struct irq_desc * desc);
```

Arguments

irq

the interrupt number

desc

the interrupt description structure for this irq

Description

Similar as the above handle_edge_irq, but using eoi and w/o the mask/unmask logic.

handle_percpu_irq

LINUX

Kernel Hackers ManualNovember 2026

Name

`handle_percpu_irq` — Per CPU local irq handler

Synopsis

```
void handle_percpu_irq (unsigned int irq, struct irq_desc *
desc);
```

Arguments

irq

the interrupt number

desc

the interrupt description structure for this irq

Description

Per CPU interrupts on SMP machines without locking requirements

irq_cpu_online

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_cpu_online` — Invoke all irq_cpu_online functions.

Synopsis

```
void irq_cpu_online ( void);
```

Arguments

void

no arguments

Description

Iterate through all irqs and invoke the chip.`irq_cpu_online` for each.

irq_cpu_offline

LINUX

Kernel Hackers ManualNovember 2026

Name

`irq_cpu_offline` — Invoke all irq_cpu_offline functions.

Synopsis

```
void irq_cpu_offline ( void);
```

Arguments

`void`

no arguments

Description

Iterate through all irqs and invoke the chip.`irq_cpu_offline` for each.

Chapter 10. Credits

The following people have contributed to this document:

1. Thomas Gleixner<tglx@linutronix.de>
2. Ingo Molnar<mingo@elte.hu>