## NAME

**BUS_SETUP_INTR**, **bus_setup_intr**, **BUS_TEARDOWN_INTR**, **bus_teardown_intr** - create, attach and teardown an interrupt handler

## SYNOPSIS

**#include <sys/param.h>**
**#include <sys/bus.h>**

*int*
**BUS_SETUP_INTR**(*device_t dev, device_t child, struct resource *irq, int flags, driver_filter_t *filter, driver_intr_t *ithread, void *arg, void **cookiep*);

*int*
**bus_setup_intr**(*device_t dev, struct resource *r, int flags, driver_filter_t filter, driver_intr_t ithread, void *arg, void **cookiep*);

*int*
**BUS_TEARDOWN_INTR**(*device_t dev, device_t child, struct resource *irq, void *cookiep*);

*int*
**bus_teardown_intr**(*device_t dev, struct resource *r, void *cookiep*);

## DESCRIPTION

The **BUS_SETUP_INTR**() method will create and attach an interrupt handler to an interrupt previously allocated by the resource manager's BUS_ALLOC_RESOURCE(9) method. The *flags* are found in *<sys/bus.h>*, and give the broad category of interrupt. The *flags* also tell the interrupt handlers about certain device driver characteristics. INTR_EXCL marks the handler as being an exclusive handler for this interrupt. INTR_MPSAFE tells the scheduler that the interrupt handler is well behaved in a preemptive environment (''SMP safe''), and does not need to be protected by the ''Giant Lock'' mutex. INTR_ENTROPY marks the interrupt as being a good source of entropy - this may be used by the entropy device */dev/random*.

To define a time-critical handler that will not execute any potentially blocking operation, use the *filter* argument. See the *Filter Routines* section below for information on writing a filter. Otherwise, use the *ithread* argument. The defined handler will be called with the value *arg* as its only argument. See the *ithread Routines* section below for more information on writing an interrupt handler.

The *cookiep* argument is a pointer to a *void* * that **BUS_SETUP_INTR**() will write a cookie for the parent bus' use to if it is successful in establishing an interrupt. Driver writers may assume that this cookie will be non-zero. The nexus driver will write 0 on failure to *cookiep*.

The interrupt handler will be detached by **BUS_TEARDOWN_INTR**(). The cookie needs to be passed to **BUS_TEARDOWN_INTR**() in order to tear down the correct interrupt handler. Once **BUS_TEARDOWN_INTR**() returns, it is guaranteed that the interrupt function is not active and will no longer be called.

Mutexes are not allowed to be held across calls to these functions.

**Filter Routines**

A filter runs in primary interrupt context. In this context, normal mutexes cannot be used. Only the spin lock version of these can be used (specified by passing MTX_SPIN to **mtx_init**() when initializing the mutex). wakeup(9) and similar routines can be called. Atomic operations from *machine/atomic* may be used. Reads and writes to hardware through bus_space(9) may be used. PCI configuration registers may be read and written. All other kernel interfaces cannot be used.

In this restricted environment, care must be taken to account for all races. A careful analysis of races should be done as well. It is generally cheaper to take an extra interrupt, for example, than to protect variables with spinlocks. Read, modify, write cycles of hardware registers need to be carefully analyzed if other threads are accessing the same registers.

Generally, a filter routine will use one of two strategies. The first strategy is to simply mask the interrupt in hardware and allow the ithread routine to read the state from the hardware and then reenable interrupts. The ithread also acknowledges the interrupt before re-enabling the interrupt source in hardware. Most PCI hardware can mask its interrupt source.

The second common approach is to use a filter with multiple taskqueue(9) tasks. In this case, the filter acknowledges the interrupts and queues the work to the appropriate taskqueue. Where one has to multiplex different kinds of interrupt sources, like a network card's transmit and receive paths, this can reduce lock contention and increase performance.

You should not malloc(9) from inside a filter. You may not call anything that uses a normal mutex. Witness may complain about these.

**ithread Routines**

You can do whatever you want in an ithread routine, except sleep. Care must be taken not to sleep in an ithread. In addition, one should minimize lock contention in an ithread routine because contested locks ripple over to all other ithread routines on that interrupt.

**Sleeping**

Sleeping is voluntarily giving up control of your thread. All the sleep routine found in msleep(9) sleep. Waiting for a condition variable described in condvar(9) is sleeping. Calling any function that does any

of these things is sleeping.

## RETURN VALUES

Zero is returned on success, otherwise an appropriate error is returned.

## SEE ALSO

random(4), device(9), driver(9), locking(9)

## AUTHORS

This manual page was written by Jeroen Ruigrok van der Werven <*asmodai@FreeBSD.org*> based on the manual pages for **BUS_CREATE_INTR**() and **BUS_CONNECT_INTR**() written by Doug Rabson <*dfr@FreeBSD.org*>.