

The Linux Kernel API

The Linux Kernel API

This documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

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. Driver Basics.....	1
1.1. Driver Entry and Exit points	1
module_init	1
module_exit.....	1
1.2. Atomic and pointer manipulation	2
atomic_read.....	2
atomic_set	3
atomic_add.....	4
atomic_sub	5
atomic_sub_and_test.....	5
atomic_inc.....	6
atomic_dec	7
atomic_dec_and_test.....	7
atomic_inc_and_test	8
atomic_add_negative.....	9
atomic_add_return	10
get_unaligned.....	11
put_unaligned.....	11
2. Data Types	13
2.1. Doubly Linked Lists	13
list_add.....	13
list_add_tail.....	13
list_add_rcu.....	14
list_add_tail_rcu.....	15
list_del.....	16
list_del_rcu.....	17
list_del_init	18
list_move.....	18
list_move_tail.....	19
list_empty.....	20
list_empty_careful.....	20
list_splice	21
list_splice_init.....	22
list_entry	23
list_for_each.....	24
__list_for_each.....	24
list_for_each_prev.....	25
list_for_each_safe	26
list_for_each_entry	27
list_for_each_entry_reverse	27

list_prepare_entry	28
list_for_each_entry_continue.....	29
list_for_each_entry_safe	30
list_for_each_rcu.....	31
list_for_each_safe_rcu	32
list_for_each_entry_rcu	33
list_for_each_continue_rcu.....	34
hlist_del_rcu.....	35
hlist_add_head_rcu	35
hlist_for_each_entry	37
hlist_for_each_entry_continue.....	37
hlist_for_each_entry_from.....	38
hlist_for_each_entry_safe	39
hlist_for_each_entry_rcu	40
3. Basic C Library Functions	43
3.1. String Conversions	43
simple_strtoll	43
simple_strtoul.....	43
simple_strtol.....	44
simple_strtoull	45
vsnprintf	46
vscnprintf	47
snprintf	48
scnprintf	49
vsprintf	50
sprintf	51
vsscanf.....	52
sscanf.....	53
3.2. String Manipulation	53
/usr/src/packages/BUILD/linux-2.6.11.4-20a/lib/string.c.....	53
strnicmp.....	54
strcpy	55
strncpy	55
strncpy	56
strcat.....	57
strncat.....	58
strlcat.....	59
strcmp.....	59
strncmp.....	60
strchr	61
strrchr	61
strnchr	62
strlen.....	63

strlen.....	63
strspn.....	64
strcspn.....	65
strpbrk.....	66
strsep.....	66
memset.....	67
memcpy.....	68
memmove.....	69
memcmp.....	70
memscan.....	70
strstr.....	71
memchr.....	72
3.3. Bit Operations.....	73
set_bit.....	73
__set_bit.....	74
clear_bit.....	75
__change_bit.....	76
change_bit.....	77
test_and_set_bit.....	77
__test_and_set_bit.....	78
test_and_clear_bit.....	79
__test_and_clear_bit.....	80
test_and_change_bit.....	81
test_bit.....	81
find_first_zero_bit.....	82
find_next_zero_bit.....	83
find_first_bit.....	84
find_next_bit.....	84
ffz.....	85
__ffs.....	86
ffs.....	87
hweight32.....	87
4. Memory Management in Linux.....	89
4.1. The Slab Cache.....	89
kmem_cache_create.....	89
kmem_cache_shrink.....	90
kmem_cache_destroy.....	91
kmem_cache_alloc.....	92
kmem_cache_alloc_node.....	93
__kmallocc.....	93
__alloc_percpu.....	94
kmem_cache_free.....	95
kcallocc.....	96

kfree	97
free_percpu	98
4.2. User Space Memory Access	98
access_ok	98
verify_area	99
get_user	101
put_user	102
__get_user	103
__put_user	104
__copy_to_user_inatomic	105
__copy_from_user_inatomic	106
strlen_user	107
__strncpy_from_user	108
strncpy_from_user	109
clear_user	110
__clear_user	111
strnlen_user	111
5. The proc filesystem	113
5.1. sysctl interface	113
register_sysctl_table	113
unregister_sysctl_table	115
proc_dostring	115
proc_dointvec	116
proc_dointvec_minmax	118
proc_doulongvec_minmax	119
proc_doulongvec_ms_jiffies_minmax	120
proc_dointvec_jiffies	121
proc_dointvec_userhz_jiffies	123
proc_dointvec_ms_jiffies	124
6. The debugfs filesystem	127
6.1. debugfs interface	127
debugfs_create_file	127
debugfs_create_dir	128
debugfs_remove	130
debugfs_create_u8	131
debugfs_create_u16	132
debugfs_create_u32	134
debugfs_create_bool	135
7. The Linux VFS	139
7.1. The Directory Cache	139
d_invalidate	139
shrink_dcache_sb	139

have_submounts.....	140
shrink_dcache_parent	141
d_alloc.....	142
d_instantiate	142
d_instantiate_unique	143
d_alloc_root	144
d_alloc_anon.....	145
d_splice_alias.....	146
d_lookup	147
d_validate	148
d_delete.....	149
d_rehash	150
d_move.....	151
__d_path.....	152
find_inode_number	153
__d_drop	154
d_add.....	155
d_add_unique.....	155
dget.....	156
d_unhashed	157
7.2. Inode Handling.....	158
clear_inode.....	158
invalidate_inodes.....	159
new_inode	159
iunique.....	160
ilookup5	161
ilookup	162
iget5_locked.....	163
iget_locked.....	165
__insert_inode_hash	165
remove_inode_hash	166
iput	167
bmap.....	168
update_atime.....	168
inode_update_time	169
make_bad_inode	170
is_bad_inode	171
7.3. Registration and Superblocks	172
deactivate_super.....	172
generic_shutdown_super.....	172
sget	173
get_super.....	174
7.4. File Locks.....	175

posix_lock_file.....	175
posix_lock_file_wait.....	176
locks_mandatory_area	177
__break_lease.....	178
lease_get_mtime	178
flock_lock_file_wait.....	179
posix_block_lock	180
posix_unblock_lock	181
lock_may_read.....	182
lock_may_write.....	183
locks_mandatory_locked	184
fcntl_getlease	184
__setlease	186
fcntl_setlease.....	186
sys_flock	187
get_locks_status	188
8. Linux Networking	191
8.1. Socket Buffer Functions.....	191
struct sk_buff.....	191
skb_queue_empty	195
skb_get	196
kfree_skb.....	196
skb_cloned	197
skb_shared.....	198
skb_share_check	199
skb_unshare.....	200
skb_peek	200
skb_peek_tail	201
skb_queue_len.....	202
skb_put.....	203
skb_push	204
skb_pull.....	205
skb_headroom.....	205
skb_tailroom	206
skb_reserve	207
skb_trim	207
skb_orphan.....	208
__dev_alloc_skb	209
dev_alloc_skb	210
skb_cow	211
skb_padto	212
skb_over_panic	212
skb_under_panic	213

alloc_skb	214
__kfree_skb.....	215
skb_clone	216
skb_copy	217
pskb_copy	218
pskb_expand_head.....	219
skb_copy_expand.....	220
skb_pad	221
__pskb_pull_tail.....	222
skb_dequeue.....	223
skb_dequeue_tail.....	224
skb_queue_purge	224
skb_queue_head.....	225
skb_queue_tail	226
skb_unlink.....	227
skb_append	228
skb_insert	229
skb_split	229
8.2. Socket Filter	230
sk_run_filter	230
sk_chk_filter.....	231
8.3. Generic Network Statistics	232
struct gnet_stats_basic	232
struct gnet_stats_rate_est	233
struct gnet_stats_queue	233
struct gnet_estimator.....	234
gnet_stats_start_copy_compat	235
gnet_stats_start_copy	236
gnet_stats_copy_basic.....	237
gnet_stats_copy_rate_est	238
gnet_stats_copy_queue	239
gnet_stats_copy_app	240
gnet_stats_finish_copy	241
gen_new_estimator	242
gen_kill_estimator.....	243
gen_replace_estimator	244
9. Network device support.....	247
9.1. Driver Support.....	247
dev_add_pack	247
__dev_remove_pack	247
dev_remove_pack	248
netdev_boot_setup_check	249
__dev_get_by_name	250

dev_get_by_name	251
__dev_get_by_index	251
dev_get_by_index	252
dev_get_by_flags	253
dev_alloc_name	254
netdev_state_change	255
dev_load	256
dev_open	256
dev_close	257
register_netdevice_notifier	258
unregister_netdevice_notifier	259
dev_queue_xmit	259
netif_rx	260
register_gifconf	261
netdev_set_master	262
dev_set_promiscuity	263
dev_set_allmulti	264
dev_ioctl	265
register_netdevice	266
register_netdev	266
alloc_netdev	267
free_netdev	268
unregister_netdevice	269
unregister_netdev	270
9.2. 8390 Based Network Cards	270
ei_open	271
ei_close	271
ei_interrupt	272
__alloc_ei_netdev	273
NS8390_init	274
9.3. Synchronous PPP	275
sppp_input	275
sppp_close	276
sppp_open	276
sppp_reopen	277
sppp_change_mtu	278
sppp_do_ioctl	279
sppp_attach	280
sppp_detach	281
10. Module Support	283
10.1. Module Loading	283
request_module	283
call_usermodehelper	284

10.2. Inter Module support.....	285
11. Hardware Interfaces	287
11.1. Interrupt Handling.....	287
/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/irq.c	287
11.2. MTRR Handling	287
mtrr_add.....	288
mtrr_del.....	289
11.3. PCI Support Library.....	290
pci_bus_max_busnr	290
pci_max_busnr	291
pci_find_capability.....	292
pci_bus_find_capability	293
pci_find_parent_resource.....	294
pci_set_power_state	294
pci_choose_state	295
pci_save_state	296
pci_restore_state	297
pci_enable_device_bars	298
pci_enable_device.....	298
pci_disable_device.....	299
pci_enable_wake.....	300
pci_release_region	301
pci_request_region.....	302
pci_release_regions.....	303
pci_request_regions	304
pci_set_master.....	305
pci_set_mwi	305
pci_clear_mwi.....	306
11.4. PCI Hotplug Support Library.....	307
pci_hp_register.....	307
pci_hp_deregister.....	308
pci_hp_change_slot_info	308
11.5. MCA Architecture	309
11.5.1. MCA Device Functions	310
11.5.2. MCA Bus DMA.....	310
mca_enable_dma.....	310
mca_disable_dma.....	310
mca_set_dma_addr	311
mca_get_dma_addr	312
mca_set_dma_count.....	313
mca_get_dma_residue.....	313
mca_set_dma_io	314
mca_set_dma_mode.....	315

12. The Device File System.....	317
devfs_mk_dir	317
13. Security Framework	319
register_security	319
unregister_security	319
mod_reg_security	320
mod_unreg_security	321
capable	322
14. Power Management	325
pm_register	325
pm_unregister	325
pm_unregister_all	326
pm_send_all	327
15. Block Devices.....	329
blk_get_backing_dev_info.....	329
blk_queue_prep_rq	329
blk_queue_merge_bvec	330
blk_queue_make_request.....	331
blk_queue_ordered.....	332
blk_queue_issue_flush_fn.....	333
blk_complete_barrier_rq.....	334
blk_complete_barrier_rq_locked	335
blk_queue_bounce_limit.....	336
blk_queue_max_sectors	337
blk_queue_max_phys_segments.....	338
blk_queue_max_hw_segments	338
blk_queue_max_segment_size	339
blk_queue_hardsect_size	340
blk_queue_stack_limits.....	341
blk_queue_segment_boundary	342
blk_queue_dma_alignment.....	342
blk_queue_find_tag.....	343
blk_queue_free_tags	344
blk_queue_init_tags	345
blk_queue_resize_tags	346
blk_queue_end_tag	347
blk_queue_start_tag	348
blk_queue_invalidate_tags.....	349
generic_unplug_device	349
blk_start_queue.....	350
blk_stop_queue	351
blk_sync_queue.....	352

blk_run_queue.....	353
blk_cleanup_queue	353
blk_init_queue.....	354
blk_requeue_request	356
blk_insert_request.....	356
blk_rq_map_user.....	358
blk_rq_unmap_user.....	359
blk_execute_rq	360
blkdev_issue_flush	361
blkdev_scsi_issue_flush_fn.....	361
blk_end_sync_rq.....	362
blk_congestion_wait	363
blk_attempt_remerge	364
generic_make_request.....	365
submit_bio	366
end_that_request_first.....	367
end_that_request_chunk	368
16. Miscellaneous Devices.....	371
misc_register	371
misc_deregister	371
17. Video4Linux	373
video_register_device	373
video_unregister_device	374
18. Sound Devices.....	375
register_sound_special.....	375
register_sound_mixer.....	375
register_sound_midi.....	376
register_sound_dsp	377
register_sound_synth	378
unregister_sound_special.....	379
unregister_sound_mixer.....	379
unregister_sound_midi.....	380
unregister_sound_dsp	381
unregister_sound_synth	382
19. 16x50 UART Driver	385
uart_update_timeout	385
uart_get_baud_rate.....	385
uart_get_divisor	387
uart_register_driver.....	387
uart_unregister_driver.....	388
uart_add_one_port	389
uart_remove_one_port	390

uart_register_port.....	391
uart_unregister_port.....	392
serial8250_suspend_port.....	393
serial8250_resume_port.....	393
serial8250_register_port	394
serial8250_unregister_port	395
register_serial.....	396
unregister_serial.....	396
20. Z85230 Support Library	399
z8530_interrupt.....	399
z8530_sync_open.....	400
z8530_sync_close	400
z8530_sync_dma_open.....	401
z8530_sync_dma_close	402
z8530_sync_txdma_open.....	403
z8530_sync_txdma_close	404
z8530_describe	404
z8530_init	405
z8530_shutdown	406
z8530_channel_load	407
z8530_null_rx	408
z8530_queue_xmit.....	409
z8530_get_stats.....	410
21. Frame Buffer Library.....	411
21.1. Frame Buffer Memory	411
register_framebuffer.....	411
unregister_framebuffer.....	412
fb_register_client	413
fb_unregister_client	413
fb_set_suspend.....	414
fb_get_options.....	414
21.2. Frame Buffer Console.....	415
/usr/src/packages/BUILD/linux-2.6.11.4-	
20a/drivers/video/console/fbcon.c	
416	
21.3. Frame Buffer Colormap	416
fb_alloc_cmap.....	416
fb_dealloc_cmap.....	417
fb_copy_cmap.....	418
fb_set_cmap	419
fb_default_cmap	419
fb_invert_cmaps.....	420

21.4. Frame Buffer Video Mode Database	421
fb_try_mode	421
fb_find_mode	422
fb_var_to_videomode	424
fb_videomode_to_var	424
fb_mode_is_equal	425
fb_find_best_mode	426
fb_match_mode	427
fb_add_videomode	428
fb_delete_videomode	428
fb_destroy_modelist	429
fb_videomode_to_modelist	430
21.5. Frame Buffer Macintosh Video Mode Database	431
mac_vmode_to_var	431
mac_var_to_vmode	432
mac_map_monitor_sense	433
mac_find_mode	433
21.6. Frame Buffer Fonts	434

Chapter 1. Driver Basics

1.1. Driver Entry and Exit points

module_init

Name

`module_init` — driver initialization entry point

Synopsis

```
module_init ( x );
```

Arguments

`x`

function to be run at kernel boot time or module insertion

Description

`module_init` will either be called during `do_initcalls` (if builtin) or at module insertion time (if a module). There can only be one per module.

module_exit

Name

`module_exit` — driver exit entry point

Synopsis

```
module_exit ( x );
```

Arguments

`x`

function to be run when driver is removed

Description

`module_exit` will wrap the driver clean-up code with `cleanup_module` when used with `rmmod` when the driver is a module. If the driver is statically compiled into the kernel, `module_exit` has no effect. There can only be one per module.

1.2. Atomic and pointer manipulation

atomic_read

Name

`atomic_read` — read atomic variable

Synopsis

```
atomic_read ( v );
```

Arguments

v

pointer of type `atomic_t`

Description

Atomically reads the value of *v*.

atomic_set

Name

`atomic_set` — set atomic variable

Synopsis

```
atomic_set ( v, i );
```

Arguments

v

pointer of type `atomic_t`

i

required value

Description

Atomically sets the value of *v* to *i*.

atomic_add

Name

`atomic_add` — add integer to atomic variable

Synopsis

```
void atomic_add (int i, atomic_t * v);
```

Arguments

i

integer value to add

v

pointer of type `atomic_t`

Description

Atomically adds *i* to *v*.

atomic_sub

Name

`atomic_sub` — subtract the atomic variable

Synopsis

```
void atomic_sub (int i, atomic_t * v);
```

Arguments

i
integer value to subtract

v
pointer of type `atomic_t`

Description

Atomically subtracts *i* from *v*.

atomic_sub_and_test

Name

`atomic_sub_and_test` — subtract value from variable and test result

Synopsis

```
int atomic_sub_and_test (int i, atomic_t * v);
```

Arguments

i

integer value to subtract

v

pointer of type `atomic_t`

Description

Atomically subtracts *i* from *v* and returns true if the result is zero, or false for all other cases.

`atomic_inc`

Name

`atomic_inc` — increment atomic variable

Synopsis

```
void atomic_inc (atomic_t * v);
```

Arguments

v

pointer of type `atomic_t`

Description

Atomically increments v by 1.

`atomic_dec`

Name

`atomic_dec` — decrement atomic variable

Synopsis

```
void atomic_dec (atomic_t * v);
```

Arguments

v

pointer of type `atomic_t`

Description

Atomically decrements v by 1.

atomic_dec_and_test

Name

`atomic_dec_and_test` — decrement and test

Synopsis

```
int atomic_dec_and_test (atomic_t * v);
```

Arguments

`v`
pointer of type `atomic_t`

Description

Atomically decrements `v` by 1 and returns true if the result is 0, or false for all other cases.

atomic_inc_and_test

Name

`atomic_inc_and_test` — increment and test

Synopsis

```
int atomic_inc_and_test (atomic_t * v);
```

Arguments

v

pointer of type `atomic_t`

Description

Atomically increments *v* by 1 and returns true if the result is zero, or false for all other cases.

atomic_add_negative

Name

`atomic_add_negative` — add and test if negative

Synopsis

```
int atomic_add_negative (int i, atomic_t * v);
```

Arguments

i

integer value to add

v

pointer of type `atomic_t`

Description

Atomically adds *i* to *v* and returns true if the result is negative, or false when result is greater than or equal to zero.

atomic_add_return

Name

`atomic_add_return` — add and return

Synopsis

```
int atomic_add_return (int i, atomic_t * v);
```

Arguments

i

integer value to add

v

pointer of type `atomic_t`

Description

Atomically adds i to v and returns $i + v$

get_unaligned

Name

`get_unaligned` — get value from possibly mis-aligned location

Synopsis

```
get_unaligned ( ptr );
```

Arguments

ptr

pointer to value

Description

This macro should be used for accessing values larger in size than single bytes at locations that are expected to be improperly aligned, e.g. retrieving a u16 value from a location not u16-aligned.

Note that unaligned accesses can be very expensive on some architectures.

put_unaligned

Name

`put_unaligned` — put value to a possibly mis-aligned location

Synopsis

```
put_unaligned ( val, ptr );
```

Arguments

val

value to place

ptr

pointer to location

Description

This macro should be used for placing values larger in size than single bytes at locations that are expected to be improperly aligned, e.g. writing a u16 value to a location not u16-aligned.

Note that unaligned accesses can be very expensive on some architectures.

Chapter 2. Data Types

2.1. Doubly Linked Lists

list_add

Name

`list_add` — add a new entry

Synopsis

```
void list_add (struct list_head * new, struct list_head *  
head);
```

Arguments

new

new entry to be added

head

list head to add it after

Description

Insert a new entry after the specified head. This is good for implementing stacks.

list_add_tail

Name

`list_add_tail` — add a new entry

Synopsis

```
void list_add_tail (struct list_head * new, struct list_head *  
head);
```

Arguments

new

new entry to be added

head

list head to add it before

Description

Insert a new entry before the specified head. This is useful for implementing queues.

list_add_rcu

Name

`list_add_rcu` — add a new entry to rcu-protected list

Synopsis

```
void list_add_rcu (struct list_head * new, struct list_head *  
head);
```

Arguments

new

new entry to be added

head

list head to add it after

Description

Insert a new entry after the specified head. This is good for implementing stacks.

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `list_add_rcu` or `list_del_rcu`, running on this same list. However, it is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `list_for_each_entry_rcu`.

`list_add_tail_rcu`

Name

`list_add_tail_rcu` — add a new entry to rcu-protected list

Synopsis

```
void list_add_tail_rcu (struct list_head * new, struct  
list_head * head);
```

Arguments

new

new entry to be added

head

list head to add it before

Description

Insert a new entry before the specified head. This is useful for implementing queues.

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `list_add_tail_rcu` or `list_del_rcu`, running on this same list. However, it is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `list_for_each_entry_rcu`.

list_del

Name

`list_del` — deletes entry from list.

Synopsis

```
void list_del (struct list_head * entry);
```

Arguments

entry

the element to delete from the list.

Note

`list_empty` on `entry` does not return true after this, the entry is in an undefined state.

list_del_rcu

Name

`list_del_rcu` — deletes entry from list without re-initialization

Synopsis

```
void list_del_rcu (struct list_head * entry);
```

Arguments

entry

the element to delete from the list.

Note

`list_empty` on entry does not return true after this, the entry is in an undefined state. It is useful for RCU based lockfree traversal.

In particular, it means that we can not poison the forward pointers that may still be used for walking the list.

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `list_del_rcu` or `list_add_rcu`, running on this same list. However, it is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `list_for_each_entry_rcu`.

Note that the caller is not permitted to immediately free the newly deleted entry. Instead, either `synchronize_kernel` or `call_rcu` must be used to defer freeing until an RCU grace period has elapsed.

`list_del_init`

Name

`list_del_init` — deletes entry from list and reinitialize it.

Synopsis

```
void list_del_init (struct list_head * entry);
```

Arguments

entry

the element to delete from the list.

list_move

Name

`list_move` — delete from one list and add as another's head

Synopsis

```
void list_move (struct list_head * list, struct list_head *  
head);
```

Arguments

list

the entry to move

head

the head that will precede our entry

list_move_tail

Name

`list_move_tail` — delete from one list and add as another's tail

Synopsis

```
void list_move_tail (struct list_head * list, struct list_head  
* head);
```

Arguments

list

the entry to move

head

the head that will follow our entry

list_empty

Name

`list_empty` — tests whether a list is empty

Synopsis

```
int list_empty (const struct list_head * head);
```

Arguments

head

the list to test.

list_empty_careful

Name

`list_empty_careful` — tests whether a list is

Synopsis

```
int list_empty_careful (const struct list_head * head);
```

Arguments

head

the list to test.

Description

`empty_and_` checks that no other CPU might be in the process of still modifying either member

NOTE

using `list_empty_careful` without synchronization can only be safe if the only activity that can happen to the list entry is `list_del_init`. Eg. it cannot be used if another CPU could `re-list_add` it.

list_splice

Name

`list_splice` — join two lists

Synopsis

```
void list_splice (struct list_head * list, struct list_head *  
head);
```

Arguments

list

the new list to add.

head

the place to add it in the first list.

list_splice_init

Name

`list_splice_init` — join two lists and reinitialise the emptied list.

Synopsis

```
void list_splice_init (struct list_head * list, struct  
list_head * head);
```

Arguments

list

the new list to add.

head

the place to add it in the first list.

Description

The list at *list* is reinitialised

list_entry

Name

`list_entry` — get the struct for this entry

Synopsis

```
list_entry ( ptr, type, member );
```

Arguments

ptr

the &struct list_head pointer.

type

the type of the struct this is embedded in.

member

the name of the list_struct within the struct.

list_for_each

Name

list_for_each — iterate over a list

Synopsis

```
list_for_each ( pos, head );
```

Arguments

pos

the &struct list_head to use as a loop counter.

head

the head for your list.

__list_for_each

Name

`__list_for_each` — iterate over a list

Synopsis

```
__list_for_each ( pos, head );
```

Arguments

pos

the `&struct list_head` to use as a loop counter.

head

the head for your list.

Description

This variant differs from `list_for_each` in that it's the simplest possible list iteration code, no prefetching is done. Use this for code that knows the list to be very short (empty or 1 entry) most of the time.

list_for_each_prev

Name

`list_for_each_prev` — iterate over a list backwards

Synopsis

```
list_for_each_prev ( pos, head );
```

Arguments

pos

the &struct list_head to use as a loop counter.

head

the head for your list.

list_for_each_safe

Name

`list_for_each_safe` — iterate over a list safe against removal of list entry

Synopsis

```
list_for_each_safe ( pos, n, head );
```

Arguments

pos

the &struct list_head to use as a loop counter.

n

another &struct list_head to use as temporary storage

head

the head for your list.

list_for_each_entry

Name

list_for_each_entry — iterate over list of given type

Synopsis

```
list_for_each_entry ( pos, head, member );
```

Arguments

pos

the type * to use as a loop counter.

head

the head for your list.

member

the name of the list_struct within the struct.

list_for_each_entry_reverse

Name

`list_for_each_entry_reverse` — iterate backwards over list of given type.

Synopsis

```
list_for_each_entry_reverse ( pos, head, member );
```

Arguments

pos

the type * to use as a loop counter.

head

the head for your list.

member

the name of the list_struct within the struct.

list_prepare_entry

Name

`list_prepare_entry` — prepare a pos entry for use as a start point in

Synopsis

```
list_prepare_entry ( pos, head, member );
```

Arguments

pos

the type * to use as a start point

head

the head of the list

member

the name of the list_struct within the struct.

Description

list_for_each_entry_continue

list_for_each_entry_continue

Name

list_for_each_entry_continue — iterate over list of given type

Synopsis

```
list_for_each_entry_continue ( pos, head, member );
```

Arguments

pos

the type * to use as a loop counter.

head

the head for your list.

member

the name of the list_struct within the struct.

Description

continuing after existing point

list_for_each_entry_safe

Name

`list_for_each_entry_safe` — iterate over list of given type safe against removal of list entry

Synopsis

```
list_for_each_entry_safe ( pos, n, head, member );
```

Arguments

pos

the type * to use as a loop counter.

n

another type * to use as temporary storage

head

the head for your list.

member

the name of the list_struct within the struct.

list_for_each_rcu

Name

`list_for_each_rcu` — iterate over an rcu-protected list

Synopsis

```
list_for_each_rcu ( pos, head );
```

Arguments

pos

the &struct list_head to use as a loop counter.

head

the head for your list.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

list_for_each_safe_rcu

Name

`list_for_each_safe_rcu` — iterate over an rcu-protected list safe

Synopsis

```
list_for_each_safe_rcu ( pos,  n,  head );
```

Arguments

pos

the `&struct list_head` to use as a loop counter.

n

another `&struct list_head` to use as temporary storage

head

the head for your list.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

list_for_each_entry_rcu

Name

`list_for_each_entry_rcu` — iterate over rcu list of given type

Synopsis

```
list_for_each_entry_rcu ( pos, head, member );
```

Arguments

pos

the type `*` to use as a loop counter.

head

the head for your list.

member

the name of the `list_struct` within the struct.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

list_for_each_continue_rcu

Name

`list_for_each_continue_rcu` — iterate over an rcu-protected list

Synopsis

```
list_for_each_continue_rcu ( pos, head );
```

Arguments

pos

the &struct list_head to use as a loop counter.

head

the head for your list.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `list_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

hlist_del_rcu

Name

`hlist_del_rcu` — deletes entry from hash list without re-initialization

Synopsis

```
void hlist_del_rcu (struct hlist_node * n);
```

Arguments

n

the element to delete from the hash list.

Note

`list_unhashed` on entry does not return true after this, the entry is in an undefined state. It is useful for RCU based lockfree traversal.

In particular, it means that we can not poison the forward pointers that may still be used for walking the hash list.

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `hlist_add_head_rcu` or `hlist_del_rcu`, running on this same list. However, it is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `hlist_for_each_entry`.

hlist_add_head_rcu

Name

`hlist_add_head_rcu` — adds the specified element to the specified hlist,

Synopsis

```
void hlist_add_head_rcu (struct hlist_node * n, struct  
hlist_head * h);
```

Arguments

n

the element to add to the hash list.

h

the list to add to.

Description

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `hlist_add_head_rcu` or `hlist_del_rcu`, running on this same list. However, it is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `hlist_for_each_rcu`, used to prevent memory-consistency problems on Alpha CPUs. Regardless of the type of CPU, the list-traversal primitive must be guarded by `rcu_read_lock`.

Description

The caller must take whatever precautions are necessary (such as holding appropriate locks) to avoid racing with another list-mutation primitive, such as `hlist_add_head_rcu` or `hlist_del_rcu`, running on this same list. However, it

is perfectly legal to run concurrently with the `_rcu` list-traversal primitives, such as `hlist_for_each_rcu`, used to prevent memory-consistency problems on Alpha CPUs. Regardless of the type of CPU, the list-traversal primitive must be guarded by `rcu_read_lock`.

hlist_for_each_entry

Name

`hlist_for_each_entry` — iterate over list of given type

Synopsis

```
hlist_for_each_entry ( tpos, pos, head, member );
```

Arguments

tpos

the type `*` to use as a loop counter.

pos

the `&struct hlist_node` to use as a loop counter.

head

the head for your list.

member

the name of the `hlist_node` within the struct.

hlist_for_each_entry_continue

Name

`hlist_for_each_entry_continue` — iterate over a hlist continuing after existing point

Synopsis

```
hlist_for_each_entry_continue ( tpos, pos, member );
```

Arguments

tpos

the type * to use as a loop counter.

pos

the &struct hlist_node to use as a loop counter.

member

the name of the hlist_node within the struct.

hlist_for_each_entry_from

Name

`hlist_for_each_entry_from` — iterate over a hlist continuing from existing point

Synopsis

```
hlist_for_each_entry_from ( tpos, pos, member );
```

Arguments

tpos

the type * to use as a loop counter.

pos

the &struct hlist_node to use as a loop counter.

member

the name of the hlist_node within the struct.

hlist_for_each_entry_safe

Name

hlist_for_each_entry_safe — iterate over list of given type safe against removal of list entry

Synopsis

```
hlist_for_each_entry_safe ( tpos, pos, n, head, member );
```

Arguments

tpos

the type * to use as a loop counter.

pos

the &struct hlist_node to use as a loop counter.

n

another &struct hlist_node to use as temporary storage

head

the head for your list.

member

the name of the hlist_node within the struct.

hlist_for_each_entry_rcu

Name

`hlist_for_each_entry_rcu` — iterate over rcu list of given type

Synopsis

```
hlist_for_each_entry_rcu ( tpos, pos, head, member );
```

Arguments

tpos

pos

the `&struct hlist_node` to use as a loop counter.

head

the head for your list.

member

the name of the `hlist_node` within the struct.

Description

This list-traversal primitive may safely run concurrently with the `_rcu` list-mutation primitives such as `hlist_add_rcu` as long as the traversal is guarded by `rcu_read_lock`.

Chapter 3. Basic C Library Functions

When writing drivers, you cannot in general use routines which are from the C Library. Some of the functions have been found generally useful and they are listed below. The behaviour of these functions may vary slightly from those defined by ANSI, and these deviations are noted in the text.

3.1. String Conversions

simple_strtoll

Name

`simple_strtoll` — convert a string to a signed long long

Synopsis

```
long long simple_strtoll (const char * cp, char ** endp,  
unsigned int base);
```

Arguments

cp

The start of the string

endp

A pointer to the end of the parsed string will be placed here

base

The number base to use

simple_strtoul

Name

`simple_strtoul` — convert a string to an unsigned long

Synopsis

```
unsigned long simple_strtoul (const char * cp, char ** endp,  
unsigned int base);
```

Arguments

cp

The start of the string

endp

A pointer to the end of the parsed string will be placed here

base

The number base to use

simple_strtol

Name

`simple_strtol` — convert a string to a signed long

Synopsis

```
long simple_strtol (const char * cp, char ** endp, unsigned  
int base);
```

Arguments

cp

The start of the string

endp

A pointer to the end of the parsed string will be placed here

base

The number base to use

simple_strtoul

Name

`simple_strtoul` — convert a string to an unsigned long long

Synopsis

```
unsigned long long simple_strtoul (const char * cp, char **  
endp, unsigned int base);
```

Arguments

cp

The start of the string

endp

A pointer to the end of the parsed string will be placed here

base

The number base to use

vsnprintf

Name

`vsnprintf` — Format a string and place it in a buffer

Synopsis

```
int vsnprintf (char * buf, size_t size, const char * fmt,  
va_list args);
```

Arguments

buf

The buffer to place the result into

size

The size of the buffer, including the trailing null space

fmt

The format string to use

args

Arguments for the format string

Description

The return value is the number of characters which would be generated for the given input, excluding the trailing `'\0'`, as per ISO C99. If you want to have the exact number of characters written into *buf* as return value (not including the trailing `'\0'`), use `vscnprintf`. If the return is greater than or equal to *size*, the resulting string is truncated.

Call this function if you are already dealing with a `va_list`. You probably want `snprintf` instead.

vscnprintf

Name

`vscnprintf` — Format a string and place it in a buffer

Synopsis

```
int vscnprintf (char * buf, size_t size, const char * fmt,  
va_list args);
```

Arguments

buf

The buffer to place the result into

size

The size of the buffer, including the trailing null space

fmt

The format string to use

args

Arguments for the format string

Description

The return value is the number of characters which have been written into the *buf* not including the trailing `'\0'`. If *size* is ≤ 0 the function returns 0.

Call this function if you are already dealing with a `va_list`. You probably want `snprintf` instead.

snprintf

Name

`snprintf` — Format a string and place it in a buffer

Synopsis

```
int snprintf (char * buf, size_t size, const char * fmt,  
...);
```

Arguments

buf

The buffer to place the result into

size

The size of the buffer, including the trailing null space

fmt

The format string to use @...: Arguments for the format string

...

variable arguments

Description

The return value is the number of characters which would be generated for the given input, excluding the trailing null, as per ISO C99. If the return is greater than or equal to *size*, the resulting string is truncated.

scanfprintf

Name

`scanfprintf` — Format a string and place it in a buffer

Synopsis

```
int scanfprintf (char * buf, size_t size, const char * fmt,  
...);
```

Arguments

buf

The buffer to place the result into

size

The size of the buffer, including the trailing null space

fmt

The format string to use @...: Arguments for the format string

...

variable arguments

Description

The return value is the number of characters written into *buf* not including the trailing `'\0'`. If *size* is ≤ 0 the function returns 0. If the return is greater than or equal to *size*, the resulting string is truncated.

vsprintf

Name

`vsprintf` — Format a string and place it in a buffer

Synopsis

```
int vsprintf (char * buf, const char * fmt, va_list args);
```

Arguments

buf

The buffer to place the result into

fmt

The format string to use

args

Arguments for the format string

Description

The function returns the number of characters written into *buf*. Use `vsnprintf` or `vscnprintf` in order to avoid buffer overflows.

Call this function if you are already dealing with a `va_list`. You probably want `sprintf` instead.

sprintf

Name

`sprintf` — Format a string and place it in a buffer

Synopsis

```
int sprintf (char * buf, const char * fmt, ...);
```

Arguments

buf

The buffer to place the result into

fmt

The format string to use @...: Arguments for the format string

...

variable arguments

Description

The function returns the number of characters written into *buf*. Use `snprintf` or `scnprintf` in order to avoid buffer overflows.

vsscanf

Name

`vsscanf` — Unformat a buffer into a list of arguments

Synopsis

```
int vsscanf (const char * buf, const char * fmt, va_list  
args);
```

Arguments

buf

input buffer

fmt

format of buffer

args

arguments

sscanf

Name

`sscanf` — Unformat a buffer into a list of arguments

Synopsis

```
int sscanf (const char * buf, const char * fmt, ...);
```

Arguments

buf

input buffer

fmt

formatting of buffer @...: resulting arguments

...

variable arguments

3.2. String Manipulation

/usr/src/packages/BUILD/linux-2.6.11.4-20a/lib/string.c

Name

`/usr/src/packages/BUILD/linux-2.6.11.4-20a/lib/string.c`
— Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file

`/usr/src/packages/BUILD/linux-2.6.11.4-20a/lib/string.c`
at this point, but none was found. This dummy section is inserted to allow generation to continue.

strnicmp

Name

`strnicmp` — Case insensitive, length-limited string comparison

Synopsis

```
int strnicmp (const char * s1, const char * s2, size_t len);
```

Arguments

s1

One string

s2

The other string

len

the maximum number of characters to compare

strcpy

Name

`strcpy` — Copy a NUL terminated string

Synopsis

```
char * strcpy (char * dest, const char * src);
```

Arguments

dest

Where to copy the string to

src

Where to copy the string from

strncpy

Name

`strncpy` — Copy a length-limited, NUL-terminated string

Synopsis

```
char * strncpy (char * dest, const char * src, size_t count);
```

Arguments

dest

Where to copy the string to

src

Where to copy the string from

count

The maximum number of bytes to copy

Description

The result is not NUL-terminated if the source exceeds *count* bytes.

strncpy

Name

`strncpy` — Copy a NUL terminated string into a sized buffer

Synopsis

```
size_t strncpy (char * dest, const char * src, size_t size);
```

Arguments

dest

Where to copy the string to

src

Where to copy the string from

size

size of destination buffer

BSD

the result is always a valid NUL-terminated string that fits in the buffer (unless, of course, the buffer size is zero). It does not pad out the result like `strncpy` does.

strcat

Name

`strcat` — Append one NUL-terminated string to another

Synopsis

```
char * strcat (char * dest, const char * src);
```

Arguments

dest

The string to be appended to

src

The string to append to it

strncat

Name

`strncat` — Append a length-limited, NUL-terminated string to another

Synopsis

```
char * strncat (char * dest, const char * src, size_t count);
```

Arguments

dest

The string to be appended to

src

The string to append to it

count

The maximum numbers of bytes to copy

Description

Note that in contrast to `strncpy`, `strncat` ensures the result is terminated.

strlcat

Name

`strlcat` — Append a length-limited, NUL-terminated string to another

Synopsis

```
size_t strlcat (char * dest, const char * src, size_t count);
```

Arguments

dest

The string to be appended to

src

The string to append to it

count

The size of the destination buffer.

strcmp

Name

`strcmp` — Compare two strings

Synopsis

```
int strcmp (const char * cs, const char * ct);
```

Arguments

cs

One string

ct

Another string

strncmp

Name

`strncmp` — Compare two length-limited strings

Synopsis

```
int strncmp (const char * cs, const char * ct, size_t count);
```

Arguments

cs

One string

ct

Another string

count

The maximum number of bytes to compare

strchr

Name

`strchr` — Find the first occurrence of a character in a string

Synopsis

```
char * strchr (const char * s, int c);
```

Arguments

s

The string to be searched

c

The character to search for

strrchr

Name

`strrchr` — Find the last occurrence of a character in a string

Synopsis

```
char * strrchr (const char * s, int c);
```

Arguments

s

The string to be searched

c

The character to search for

strnchr

Name

`strnchr` — Find a character in a length limited string

Synopsis

```
char * strnchr (const char * s, size_t count, int c);
```

Arguments

s

The string to be searched

count

The number of characters to be searched

c

The character to search for

strlen

Name

`strlen` — Find the length of a string

Synopsis

```
size_t strlen (const char * s);
```

Arguments

s

The string to be sized

strlen

Name

`strlen` — Find the length of a length-limited string

Synopsis

```
size_t strlen (const char * s, size_t count);
```

Arguments

s

The string to be sized

count

The maximum number of bytes to search

strspn

Name

`strspn` — Calculate the length of the initial substring of *s* which only

Synopsis

```
size_t strspn (const char * s, const char * accept);
```


Arguments

s

The string to be searched

accept

The string to search for

Description

contain letters in *accept*

strcspn

Name

`strcspn` — Calculate the length of the initial substring of *s* which does

Synopsis

```
size_t strcspn (const char * s, const char * reject);
```

Arguments

s

The string to be searched

reject

The string to avoid

Description

not contain letters in *reject*

strpbrk

Name

`strpbrk` — Find the first occurrence of a set of characters

Synopsis

```
char * strpbrk (const char * cs, const char * ct);
```

Arguments

cs

The string to be searched

ct

The characters to search for

strsep

Name

`strsep` — Split a string into tokens

Synopsis

```
char * strsep (char ** s, const char * ct);
```

Arguments

s

The string to be searched

ct

The characters to search for

Description

`strsep` updates *s* to point after the token, ready for the next call.

It returns empty tokens, too, behaving exactly like the libc function of that name. In fact, it was stolen from glibc2 and de-fancy-fied. Same semantics, slimmer shape. ;)

memset

Name

`memset` — Fill a region of memory with the given value

Synopsis

```
void * memset (void * s, int c, size_t count);
```

Arguments

s

Pointer to the start of the area.

c

The byte to fill the area with

count

The size of the area.

Description

Do not use `memset` to access IO space, use `memset_io` instead.

memcpy

Name

`memcpy` — Copy one area of memory to another

Synopsis

```
void * memcpy (void * dest, const void * src, size_t count);
```

Arguments

dest

Where to copy to

src

Where to copy from

count

The size of the area.

Description

You should not use this function to access IO space, use `memcpy_toio` or `memcpy_fromio` instead.

memcpy

Name

`memcpy` — Copy one area of memory to another

Synopsis

```
void * memcpy (void * dest, const void * src, size_t count);
```

Arguments

dest

Where to copy to

src

Where to copy from

count

The size of the area.

Description

Unlike `memcpy`, `memmove` copes with overlapping areas.

memcmp

Name

`memcmp` — Compare two areas of memory

Synopsis

```
int memcmp (const void * cs, const void * ct, size_t count);
```

Arguments

cs

One area of memory

ct

Another area of memory

count

The size of the area.

memscan

Name

`memscan` — Find a character in an area of memory.

Synopsis

```
void * memscan (void * addr, int c, size_t size);
```

Arguments

addr

The memory area

c

The byte to search for

size

The size of the area.

Description

returns the address of the first occurrence of *c*, or 1 byte past the area if *c* is not found

strstr

Name

`strstr` — Find the first substring in a NUL terminated string

Synopsis

```
char * strstr (const char * s1, const char * s2);
```

Arguments

s1

The string to be searched

s2

The string to search for

memchr

Name

`memchr` — Find a character in an area of memory.

Synopsis

```
void * memchr (const void * s, int c, size_t n);
```


Arguments

s

The memory area

c

The byte to search for

n

The size of the area.

Description

returns the address of the first occurrence of *c*, or `NULL` if *c* is not found

3.3. Bit Operations

set_bit

Name

`set_bit` — Atomically set a bit in memory

Synopsis

```
void set_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

the bit to set

addr

the address to start counting from

Description

This function is atomic and may not be reordered. See `__set_bit` if you do not require the atomic guarantees.

Note

there are no guarantees that this function will not be reordered on non x86 architectures, so if you are writing portable code, make sure not to rely on its reordering guarantees.

Note that *nr* may be almost arbitrarily large; this function is not restricted to acting on a single-word quantity.

`__set_bit`

Name

`__set_bit` — Set a bit in memory

Synopsis

```
void __set_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

the bit to set

addr

the address to start counting from

Description

Unlike `set_bit`, this function is non-atomic and may be reordered. If it's called on the same region of memory simultaneously, the effect may be that only one operation succeeds.

clear_bit

Name

`clear_bit` — Clears a bit in memory

Synopsis

```
void clear_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

Bit to clear

addr

Address to start counting from

Description

`clear_bit` is atomic and may not be reordered. However, it does not contain a memory barrier, so if it is used for locking purposes, you should call `smp_mb__before_clear_bit` and/or `smp_mb__after_clear_bit` in order to ensure changes are visible on other processors.

`__change_bit`

Name

`__change_bit` — Toggle a bit in memory

Synopsis

```
void __change_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

the bit to change

addr

the address to start counting from

Description

Unlike `change_bit`, this function is non-atomic and may be reordered. If it's called on the same region of memory simultaneously, the effect may be that only one operation succeeds.

change_bit

Name

`change_bit` — Toggle a bit in memory

Synopsis

```
void change_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

Bit to change

addr

Address to start counting from

Description

`change_bit` is atomic and may not be reordered. It may be reordered on other architectures than x86. Note that *nr* may be almost arbitrarily large; this function is not restricted to acting on a single-word quantity.

test_and_set_bit

Name

`test_and_set_bit` — Set a bit and return its old value

Synopsis

```
int test_and_set_bit (int nr, volatile unsigned long * addr);
```

Arguments

nr

Bit to set

addr

Address to count from

Description

This operation is atomic and cannot be reordered. It may be reordered on other architectures than x86. It also implies a memory barrier.

__test_and_set_bit

Name

`__test_and_set_bit` — Set a bit and return its old value

Synopsis

```
int __test_and_set_bit (int nr, volatile unsigned long *  
addr);
```

Arguments

nr

Bit to set

addr

Address to count from

Description

This operation is non-atomic and can be reordered. If two examples of this operation race, one can appear to succeed but actually fail. You must protect multiple accesses with a lock.

test_and_clear_bit

Name

`test_and_clear_bit` — Clear a bit and return its old value

Synopsis

```
int test_and_clear_bit (int nr, volatile unsigned long *  
addr);
```

Arguments

nr

Bit to clear

addr

Address to count from

Description

This operation is atomic and cannot be reordered. It can be reordered on other architectures other than x86. It also implies a memory barrier.

__test_and_clear_bit

Name

`__test_and_clear_bit` — Clear a bit and return its old value

Synopsis

```
int __test_and_clear_bit (int nr, volatile unsigned long *  
addr);
```

Arguments

nr

Bit to clear

addr

Address to count from

Description

This operation is non-atomic and can be reordered. If two examples of this operation race, one can appear to succeed but actually fail. You must protect multiple accesses with a lock.

test_and_change_bit

Name

`test_and_change_bit` — Change a bit and return its old value

Synopsis

```
int test_and_change_bit (int nr, volatile unsigned long *  
addr);
```

Arguments

nr

Bit to change

addr

Address to count from

Description

This operation is atomic and cannot be reordered. It also implies a memory barrier.

test_bit

Name

`test_bit` — Determine whether a bit is set

Synopsis

```
int test_bit (int nr, const volatile void * addr);
```

Arguments

nr

bit number to test

addr

Address to start counting from

find_first_zero_bit

Name

`find_first_zero_bit` — find the first zero bit in a memory region

Synopsis

```
int find_first_zero_bit (const unsigned long * addr, unsigned  
size);
```

Arguments

addr

The address to start the search at

size

The maximum size to search

Description

Returns the bit-number of the first zero bit, not the number of the byte containing a bit.

find_next_zero_bit

Name

`find_next_zero_bit` — find the first zero bit in a memory region

Synopsis

```
int find_next_zero_bit (const unsigned long * addr, int size,  
int offset);
```

Arguments

addr

The address to base the search on

size

The maximum size to search

offset

The bitnumber to start searching at

find_first_bit

Name

`find_first_bit` — find the first set bit in a memory region

Synopsis

```
int find_first_bit (const unsigned long * addr, unsigned
size);
```

Arguments

addr

The address to start the search at

size

The maximum size to search

Description

Returns the bit-number of the first set bit, not the number of the byte containing a bit.

find_next_bit

Name

`find_next_bit` — find the first set bit in a memory region

Synopsis

```
int find_next_bit (const unsigned long * addr, int size, int  
offset);
```

Arguments

addr

The address to base the search on

size

The maximum size to search

offset

The bitnumber to start searching at

ffz

Name

`ffz` — find first zero in word.

Synopsis

```
unsigned long ffz (unsigned long word);
```

Arguments

word

The word to search

Description

Undefined if no zero exists, so code should check against ~0UL first.

__ffs

Name

__ffs — find first bit in word.

Synopsis

```
unsigned long __ffs (unsigned long word);
```

Arguments

word

The word to search

Description

Undefined if no bit exists, so code should check against 0 first.

ffs

Name

`ffs` — find first bit set

Synopsis

```
int ffs (int x);
```

Arguments

`x`

the word to search

Description

This is defined the same way as the libc and compiler builtin ffs routines, therefore differs in spirit from the above ffz (man ffs).

hweight32

Name

`hweight32` — returns the hamming weight of a N-bit word

Synopsis

```
hweight32 ( x );
```

Arguments

x

the word to weigh

Description

The Hamming Weight of a number is the total number of bits set in it.

Chapter 4. Memory Management in Linux

4.1. The Slab Cache

kmem_cache_create

Name

`kmem_cache_create` — Create a cache.

Synopsis

```
kmem_cache_t * kmem_cache_create (const char * name, size_t
size, size_t align, unsigned long flags, void (*ctor) (void*,
kmem_cache_t *, unsigned long), void (*dtor) (void*,
kmem_cache_t *, unsigned long));
```

Arguments

name

A string which is used in `/proc/slabinfo` to identify this cache.

size

The size of objects to be created in this cache.

align

The required alignment for the objects.

flags

SLAB flags

ctor

A constructor for the objects.

dtor

A destructor for the objects.

Description

Returns a ptr to the cache on success, NULL on failure. Cannot be called within a int, but can be interrupted. The *ctor* is run when new pages are allocated by the cache and the *dtor* is run before the pages are handed back.

name must be valid until the cache is destroyed. This implies that the module calling this has to destroy the cache before getting unloaded.

The flags are

SLAB_POISON - Poison the slab with a known test pattern (a5a5a5a5) to catch references to uninitialised memory.

SLAB_RED_ZONE - Insert 'Red' zones around the allocated memory to check for buffer overruns.

SLAB_NO_REAP - Don't automatically reap this cache when we're under memory pressure.

SLAB_HWCACHE_ALIGN - Align the objects in this cache to a hardware cacheline. This can be beneficial if you're counting cycles as closely as davem.

kmem_cache_shrink

Name

`kmem_cache_shrink` — Shrink a cache.

Synopsis

```
int kmem_cache_shrink (kmem_cache_t * cachep);
```

Arguments

cachep

The cache to shrink.

Description

Releases as many slabs as possible for a cache. To help debugging, a zero exit status indicates all slabs were released.

kmem_cache_destroy

Name

`kmem_cache_destroy` — delete a cache

Synopsis

```
int kmem_cache_destroy (kmem_cache_t * cachep);
```

Arguments

cachep

the cache to destroy

Description

Remove a `kmem_cache_t` object from the slab cache. Returns 0 on success.

It is expected this function will be called by a module when it is unloaded. This will remove the cache completely, and avoid a duplicate cache being allocated each time a module is loaded and unloaded, if the module doesn't have persistent in-kernel storage across loads and unloads.

The cache must be empty before calling this function.

The caller must guarantee that noone will allocate memory from the cache during the `kmem_cache_destroy`.

`kmem_cache_alloc`

Name

`kmem_cache_alloc` — Allocate an object

Synopsis

```
void * kmem_cache_alloc (kmem_cache_t * cachep, int flags);
```

Arguments

cachep

The cache to allocate from.

flags

See `kmalloc`.

Description

Allocate an object from this cache. The flags are only relevant if the cache has no available objects.

kmem_cache_alloc_node

Name

`kmem_cache_alloc_node` — Allocate an object on the specified node

Synopsis

```
void * kmem_cache_alloc_node (kmem_cache_t * cachep, int  
nodeid);
```

Arguments

cachep

The cache to allocate from.

nodeid

node number of the target node.

Description

Identical to `kmem_cache_alloc`, except that this function is slow and can sleep. And it will allocate memory on the given node, which can improve the performance for cpu bound structures.

___kmalloc

Name

___kmalloc — allocate memory

Synopsis

```
void * ___kmalloc (size_t size, int flags);
```

Arguments

size

how many bytes of memory are required.

flags

the type of memory to allocate.

Description

kmalloc is the normal method of allocating memory in the kernel.

The *flags* argument may be one of:

GFP_USER - Allocate memory on behalf of user. May sleep.

GFP_KERNEL - Allocate normal kernel ram. May sleep.

GFP_ATOMIC - Allocation will not sleep. Use inside interrupt handlers.

Additionally, the GFP_DMA flag may be set to indicate the memory must be suitable for DMA. This can mean different things on different platforms. For example, on i386, it means that the memory must come from the first 16MB.

__alloc_percpu

Name

`__alloc_percpu` — allocate one copy of the object for every present

Synopsis

```
void * __alloc_percpu (size_t size, size_t align);
```

Arguments

size

how many bytes of memory are required.

align

the alignment, which can't be greater than `SMP_CACHE_BYTES`.

Description

cpu in the system, zeroing them. Objects should be dereferenced using the `per_cpu_ptr` macro only.

kmem_cache_free

Name

`kmem_cache_free` — Deallocate an object

Synopsis

```
void kmem_cache_free (kmem_cache_t * cachep, void * objp);
```

Arguments

cachep

The cache the allocation was from.

objp

The previously allocated object.

Description

Free an object which was previously allocated from this cache.

kcalloc

Name

kcalloc — allocate memory for an array. The memory is set to zero.

Synopsis

```
void * kcalloc (size_t n, size_t size, int flags);
```


Arguments

n

number of elements.

size

element size.

flags

the type of memory to allocate.

kfree

Name

`kfree` — free previously allocated memory

Synopsis

```
void kfree (const void * objp);
```

Arguments

objp

pointer returned by `kmalloc`.

Description

Don't free memory not originally allocated by `kmalloc` or you will run into trouble.

free_percpu

Name

`free_percpu` — free previously allocated percpu memory

Synopsis

```
void free_percpu (const void * objp);
```

Arguments

objp

pointer returned by `alloc_percpu`.

Description

Don't free memory not originally allocated by `alloc_percpu`. The complemented `objp` is to check for that.

4.2. User Space Memory Access

access_ok

Name

`access_ok` — Checks if a user space pointer is valid

Synopsis

```
access_ok ( type, addr, size );
```

Arguments

type

Type of access: `VERIFY_READ` or `VERIFY_WRITE`. Note that `VERIFY_WRITE` is a superset of `VERIFY_READ` - if it is safe to write to a block, it is always safe to read from it.

addr

User space pointer to start of block to check

size

Size of block to check

Context

User context only. This function may sleep.

Description

Checks if a pointer to a block of memory in user space is valid.

Returns true (nonzero) if the memory block may be valid, false (zero) if it is definitely invalid.

Note that, depending on architecture, this function probably just checks that the pointer is in the user space range - after calling this function, memory access functions may still return `-EFAULT`.

verify_area

Name

`verify_area` — Obsolete, use `access_ok`

Synopsis

```
int verify_area (int type, const void __user * addr, unsigned  
long size);
```

Arguments

type

Type of access: `VERIFY_READ` or `VERIFY_WRITE`

addr

User space pointer to start of block to check

size

Size of block to check

Context

User context only. This function may sleep.

Description

This function has been replaced by `access_ok`.

Checks if a pointer to a block of memory in user space is valid.

Returns zero if the memory block may be valid, `-EFAULT` if it is definitely invalid.

See `access_ok` for more details.

get_user

Name

`get_user` — Get a simple variable from user space.

Synopsis

```
get_user ( x, ptr );
```

Arguments

x

Variable to store result.

ptr

Source address, in user space.

Context

User context only. This function may sleep.

Description

This macro copies a single simple variable from user space to kernel space. It supports simple types like `char` and `int`, but not larger data types like structures or arrays.

ptr must have pointer-to-simple-variable type, and the result of dereferencing *ptr* must be assignable to *x* without a cast.

Returns zero on success, or `-EFAULT` on error. On error, the variable *x* is set to zero.

put_user

Name

`put_user` — Write a simple value into user space.

Synopsis

```
put_user ( x, ptr );
```

Arguments

x

Value to copy to user space.

ptr

Destination address, in user space.

Context

User context only. This function may sleep.

Description

This macro copies a single simple value from kernel space to user space. It supports simple types like `char` and `int`, but not larger data types like structures or arrays.

ptr must have pointer-to-simple-variable type, and *x* must be assignable to the result of dereferencing *ptr*.

Returns zero on success, or `-EFAULT` on error.

__get_user

Name

`__get_user` — Get a simple variable from user space, with less checking.

Synopsis

```
__get_user ( x, ptr );
```

Arguments

x

Variable to store result.

ptr

Source address, in user space.

Context

User context only. This function may sleep.

Description

This macro copies a single simple variable from user space to kernel space. It supports simple types like char and int, but not larger data types like structures or arrays.

ptr must have pointer-to-simple-variable type, and the result of dereferencing *ptr* must be assignable to *x* without a cast.

Caller must check the pointer with `access_ok` before calling this function.

Returns zero on success, or `-EFAULT` on error. On error, the variable `x` is set to zero.

__put_user

Name

`__put_user` — Write a simple value into user space, with less checking.

Synopsis

```
__put_user ( x, ptr );
```

Arguments

x

Value to copy to user space.

ptr

Destination address, in user space.

Context

User context only. This function may sleep.

Description

This macro copies a single simple value from kernel space to user space. It supports simple types like `char` and `int`, but not larger data types like structures or arrays.

ptr must have pointer-to-simple-variable type, and *x* must be assignable to the result of dereferencing *ptr*.

Caller must check the pointer with `access_ok` before calling this function.

Returns zero on success, or `-EFAULT` on error.

__copy_to_user_inatomic

Name

`__copy_to_user_inatomic` — Copy a block of data into user space, with less checking.

Synopsis

```
unsigned long __must_check __copy_to_user_inatomic (void
__user * to, const void * from, unsigned long n);
```

Arguments

to

Destination address, in user space.

from

Source address, in kernel space.

n

Number of bytes to copy.

Context

User context only. This function may sleep.

Description

Copy data from kernel space to user space. Caller must check the specified block with `access_ok` before calling this function.

Returns number of bytes that could not be copied. On success, this will be zero.

__copy_from_user_inatomic

Name

`__copy_from_user_inatomic` — Copy a block of data from user space, with less checking.

Synopsis

```
unsigned long __copy_from_user_inatomic (void * to, const void
__user * from, unsigned long n);
```

Arguments

to

Destination address, in kernel space.

from

Source address, in user space.

n

Number of bytes to copy.

Context

User context only. This function may sleep.

Description

Copy data from user space to kernel space. Caller must check the specified block with `access_ok` before calling this function.

Returns number of bytes that could not be copied. On success, this will be zero.

If some data could not be copied, this function will pad the copied data to the requested size using zero bytes.

strlen_user

Name

`strlen_user` — Get the size of a string in user space.

Synopsis

```
strlen_user ( str );
```

Arguments

str

The string to measure.

Context

User context only. This function may sleep.

Description

Get the size of a NUL-terminated string in user space.

Returns the size of the string INCLUDING the terminating NUL. On exception, returns 0.

If there is a limit on the length of a valid string, you may wish to consider using `strnlen_user` instead.

__strncpy_from_user

Name

`__strncpy_from_user` — Copy a NUL terminated string from userspace, with less checking.

Synopsis

```
long __strncpy_from_user (char * dst, const char __user * src,  
long count);
```

Arguments

dst

Destination address, in kernel space. This buffer must be at least *count* bytes long.

src

Source address, in user space.

count

Maximum number of bytes to copy, including the trailing NUL.

Description

Copies a NUL-terminated string from userspace to kernel space. Caller must check the specified block with `access_ok` before calling this function.

On success, returns the length of the string (not including the trailing NUL).

If access to userspace fails, returns `-EFAULT` (some data may have been copied).

If *count* is smaller than the length of the string, copies *count* bytes and returns *count*.

strncpy_from_user

Name

`strncpy_from_user` — Copy a NUL terminated string from userspace.

Synopsis

```
long strncpy_from_user (char * dst, const char __user * src,
long count);
```

Arguments

dst

Destination address, in kernel space. This buffer must be at least *count* bytes long.

src

Source address, in user space.

count

Maximum number of bytes to copy, including the trailing NUL.

Description

Copies a NUL-terminated string from userspace to kernel space.

On success, returns the length of the string (not including the trailing NUL).

If access to userspace fails, returns `-EFAULT` (some data may have been copied).

If *count* is smaller than the length of the string, copies *count* bytes and returns *count*.

clear_user

Name

`clear_user` — Zero a block of memory in user space.

Synopsis

```
unsigned long clear_user (void __user * to, unsigned long n);
```

Arguments

to

Destination address, in user space.

n

Number of bytes to zero.

Description

Zero a block of memory in user space.

Returns number of bytes that could not be cleared. On success, this will be zero.

__clear_user

Name

`__clear_user` — Zero a block of memory in user space, with less checking.

Synopsis

```
unsigned long __clear_user (void __user * to, unsigned long  
n);
```

Arguments

to

Destination address, in user space.

n

Number of bytes to zero.

Description

Zero a block of memory in user space. Caller must check the specified block with `access_ok` before calling this function.

Returns number of bytes that could not be cleared. On success, this will be zero.

strnlen_user

Name

`strnlen_user` — Get the size of a string in user space.

Synopsis

```
long strnlen_user (const char __user * s, long n);
```

Arguments

s

The string to measure.

n

The maximum valid length

Description

Get the size of a NUL-terminated string in user space.

Returns the size of the string INCLUDING the terminating NUL. On exception, returns 0. If the string is too long, returns a value greater than *n*.

Chapter 5. The proc filesystem

5.1. sysctl interface

register_sysctl_table

Name

`register_sysctl_table` — register a sysctl hierarchy

Synopsis

```
struct ctl_table_header * register_sysctl_table (ctl_table *  
table, int insert_at_head);
```

Arguments

table

the top-level table structure

insert_at_head

whether the entry should be inserted in front or at the end

Description

Register a sysctl table hierarchy. *table* should be a filled in `ctl_table` array. An entry with a `ctl_name` of 0 terminates the table.

The members of the `&ctl_table` structure are used as follows:

`ctl_name` - This is the numeric sysctl value used by `sysctl(2)`. The number must be unique within that level of sysctl

procname - the name of the sysctl file under /proc/sys. Set to `NULL` to not enter a sysctl file

data - a pointer to data for use by `proc_handler`

maxlen - the maximum size in bytes of the data

mode - the file permissions for the /proc/sys file, and for `sysctl(2)`

child - a pointer to the child sysctl table if this entry is a directory, or `NULL`.

`proc_handler` - the text handler routine (described below)

`strategy` - the strategy routine (described below)

`de` - for internal use by the sysctl routines

`extra1`, `extra2` - extra pointers usable by the proc handler routines

Leaf nodes in the sysctl tree will be represented by a single file under /proc;
non-leaf nodes will be represented by directories.

`sysctl(2)` can automatically manage read and write requests through the sysctl table. The `data` and `maxlen` fields of the `ctl_table` struct enable minimal validation of the values being written to be performed, and the `mode` field allows minimal authentication.

More sophisticated management can be enabled by the provision of a strategy routine with the table entry. This will be called before any automatic read or write of the data is performed.

The strategy routine may return

< 0 - Error occurred (error is passed to user process)

0 - OK - proceed with automatic read or write.

> 0 - OK - read or write has been done by the strategy routine, so return immediately.

There must be a `proc_handler` routine for any terminal nodes mirrored under /proc/sys (non-terminals are handled by a built-in directory handler). Several default handlers are available to cover common cases -

`proc_doststring`, `proc_dointvec`, `proc_dointvec_jiffies`,
`proc_dointvec_userhz_jiffies`, `proc_dointvec_minmax`,
`proc_doulongvec_ms_jiffies_minmax`, `proc_doulongvec_minmax`

It is the handler's job to read the input buffer from user memory and process it. The handler should return 0 on success.

This routine returns `NULL` on a failure to register, and a pointer to the table header on success.

unregister_sysctl_table

Name

`unregister_sysctl_table` — unregister a sysctl table hierarchy

Synopsis

```
void unregister_sysctl_table (struct ctl_table_header *  
header);
```

Arguments

header

the header returned from `register_sysctl_table`

Description

Unregisters the sysctl table and all children. proc entries may not actually be removed until they are no longer used by anyone.

proc_dostring

Name

`proc_dostring` — read a string sysctl

Synopsis

```
int proc_dostring (ctl_table * table, int write, struct file *  
filp, void __user * buffer, size_t * lenp, loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes a string from/to the user buffer. If the kernel buffer provided is not large enough to hold the string, the string is truncated. The copied string is NULL-terminated. If the string is being read by the user process, it is copied and a newline '\n' is added. It is truncated if the buffer is not large enough.

Returns 0 on success.

proc_dointvec

Name

`proc_dointvec` — read a vector of integers

Synopsis

```
int proc_dointvec (ctl_table * table, int write, struct file *  
filp, void __user * buffer, size_t * lenp, loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned int)` integer values from/to the user buffer, treated as an ASCII string.

Returns 0 on success.

`proc_dointvec_minmax`

Name

`proc_dointvec_minmax` — read a vector of integers with min/max values

Synopsis

```
int proc_dointvec_minmax (ctl_table * table, int write, struct  
file * filp, void __user * buffer, size_t * lenp, loff_t *  
ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned int)` integer values from/to the user buffer, treated as an ASCII string.

This routine will ensure the values are within the range specified by `table->extra1` (min) and `table->extra2` (max).

Returns 0 on success.

proc_doulongvec_minmax

Name

`proc_doulongvec_minmax` — read a vector of long integers with min/max values

Synopsis

```
int proc_doulongvec_minmax (ctl_table * table, int write,
struct file * filp, void __user * buffer, size_t * lenp,
loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned long)` unsigned long values from/to the user buffer, treated as an ASCII string.

This routine will ensure the values are within the range specified by `table->extra1` (min) and `table->extra2` (max).

Returns 0 on success.

proc_doulongvec_ms_jiffies_minmax

Name

`proc_doulongvec_ms_jiffies_minmax` — read a vector of millisecond values with min/max values

Synopsis

```
int proc_doulongvec_ms_jiffies_minmax (ctl_table * table, int
write, struct file * filp, void __user * buffer, size_t *
```



```
lenp, loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned long)` unsigned long values from/to the user buffer, treated as an ASCII string. The values are treated as milliseconds, and converted to jiffies when they are stored.

This routine will ensure the values are within the range specified by `table->extra1` (min) and `table->extra2` (max).

Returns 0 on success.

proc_dointvec_jiffies

Name

`proc_dointvec_jiffies` — read a vector of integers as seconds

Synopsis

```
int proc_dointvec_jiffies (ctl_table * table, int write,  
struct file * filp, void __user * buffer, size_t * lenp,  
loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned int)` integer values from/to the user buffer, treated as an ASCII string. The values read are assumed to be in seconds, and are converted into jiffies.

Returns 0 on success.

proc_dointvec_userhz_jiffies

Name

`proc_dointvec_userhz_jiffies` — read a vector of integers as 1/USER_HZ seconds

Synopsis

```
int proc_dointvec_userhz_jiffies (ctl_table * table, int
write, struct file * filp, void __user * buffer, size_t *
lenp, loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned int)` integer values from/to the user buffer, treated as an ASCII string. The values read are assumed to be in 1/USER_HZ seconds, and are converted into jiffies.

Returns 0 on success.

proc_dointvec_ms_jiffies

Name

`proc_dointvec_ms_jiffies` — read a vector of integers as 1 milliseconds

Synopsis

```
int proc_dointvec_ms_jiffies (ctl_table * table, int write,  
struct file * filp, void __user * buffer, size_t * lenp,  
loff_t * ppos);
```

Arguments

table

the sysctl table

write

TRUE if this is a write to the sysctl file

filp

the file structure

buffer

the user buffer

lenp

the size of the user buffer

ppos

-- undescribed --

Description

Reads/writes up to `table->maxlen/sizeof(unsigned int)` integer values from/to the user buffer, treated as an ASCII string. The values read are assumed to be in 1/1000 seconds, and are converted into jiffies.

Returns 0 on success.

Chapter 6. The debugfs filesystem

6.1. debugfs interface

debugfs_create_file

Name

`debugfs_create_file` — create a file in the debugfs filesystem

Synopsis

```
struct dentry * debugfs_create_file (const char * name, mode_t
mode, struct dentry * parent, void * data, struct
file_operations * fops);
```

Arguments

name

a pointer to a string containing the name of the file to create.

mode

the permission that the file should have

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the file will be created in the root of the debugfs filesystem.

data

a pointer to something that the caller will want to get to later on. The `inode.u.generic_ip` pointer will point to this value on the `open` call.

fops

a pointer to a struct `file_operations` that should be used for this file.

Description

This is the basic “create a file” function for debugfs. It allows for a wide range of flexibility in creating a file, or a directory (if you want to create a directory, the `debugfs_create_dir` function is recommended to be used instead.)

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If debugfs is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

Description

This is the basic “create a file” function for debugfs. It allows for a wide range of flexibility in creating a file, or a directory (if you want to create a directory, the `debugfs_create_dir` function is recommended to be used instead.)

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If debugfs is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

debugfs_create_dir

Name

`debugfs_create_dir` — create a directory in the debugfs filesystem

Synopsis

```
struct dentry * debugfs_create_dir (const char * name, struct
dentry * parent);
```

Arguments

name

a pointer to a string containing the name of the directory to create.

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the directory will be created in the root of the debugfs filesystem.

Description

This function creates a directory in debugfs with the given name.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, NULL will be returned.

If debugfs is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for NULL or !NULL instead as to eliminate the need for `#ifdef` in the calling code.

Description

This function creates a directory in debugfs with the given name.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, NULL will be returned.

If *debugfs* is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

debugfs_remove

Name

`debugfs_remove` — removes a file or directory from the *debugfs* filesystem

Synopsis

```
void debugfs_remove (struct dentry * dentry);
```

Arguments

dentry

a pointer to a the dentry of the file or directory to be removed.

Description

This function removes a file or directory in *debugfs* that was previously created with a call to another *debugfs* function (like `debugfs_create_file` or variants thereof.)

This function is required to be called in order for the file to be removed, no automatic cleanup of files will happen when a module is removed, you are responsible here.

Description

This function removes a file or directory in debugfs that was previously created with a call to another debugfs function (like `debugfs_create_file` or variants thereof.)

This function is required to be called in order for the file to be removed, no automatic cleanup of files will happen when a module is removed, you are responsible here.

debugfs_create_u8

Name

`debugfs_create_u8` — create a file in the debugfs filesystem that is used to read and write a unsigned 8 bit value.

Synopsis

```
struct dentry * debugfs_create_u8 (const char * name, mode_t
mode, struct dentry * parent, u8 * value);
```

Arguments

name

a pointer to a string containing the name of the file to create.

mode

the permission that the file should have

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the file will be created in the root of the debugfs filesystem.

value

a pointer to the variable that the file should read to and write from.

Description

This function creates a file in *debugfs* with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If *debugfs* is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

Description

This function creates a file in *debugfs* with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If *debugfs* is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

debugfs_create_u16

Name

`debugfs_create_u16` — create a file in the *debugfs* filesystem that is used to read and write a unsigned 8 bit value.

Synopsis

```
struct dentry * debugfs_create_u16 (const char * name, mode_t
mode, struct dentry * parent, u16 * value);
```

Arguments

name

a pointer to a string containing the name of the file to create.

mode

the permission that the file should have

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the file will be created in the root of the debugfs filesystem.

value

a pointer to the variable that the file should read to and write from.

Description

This function creates a file in debugfs with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, NULL will be returned.

If debugfs is not enabled in the kernel, the value -ENODEV will be returned. It is not wise to check for this value, but rather, check for NULL or !NULL instead as to eliminate the need for #ifdef in the calling code.

Description

This function creates a file in *debugfs* with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If *debugfs* is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

debugfs_create_u32

Name

`debugfs_create_u32` — create a file in the *debugfs* filesystem that is used to read and write a unsigned 8 bit value.

Synopsis

```
struct dentry * debugfs_create_u32 (const char * name, mode_t  
mode, struct dentry * parent, u32 * value);
```

Arguments

name

a pointer to a string containing the name of the file to create.

mode

the permission that the file should have

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the file will be created in the root of the *debugfs* filesystem.

value

a pointer to the variable that the file should read to and write from.

Description

This function creates a file in *debugfs* with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, NULL will be returned.

If *debugfs* is not enabled in the kernel, the value -ENODEV will be returned. It is not wise to check for this value, but rather, check for NULL or !NULL instead as to eliminate the need for `#ifdef` in the calling code.

Description

This function creates a file in *debugfs* with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, NULL will be returned.

If *debugfs* is not enabled in the kernel, the value -ENODEV will be returned. It is not wise to check for this value, but rather, check for NULL or !NULL instead as to eliminate the need for `#ifdef` in the calling code.

debugfs_create_bool

Name

`debugfs_create_bool` — create a file in the debugfs filesystem that is used to read and write a boolean value.

Synopsis

```
struct dentry * debugfs_create_bool (const char * name, mode_t  
mode, struct dentry * parent, u32 * value);
```

Arguments

name

a pointer to a string containing the name of the file to create.

mode

the permission that the file should have

parent

a pointer to the parent dentry for this file. This should be a directory dentry if set. If this parameter is NULL, then the file will be created in the root of the debugfs filesystem.

value

a pointer to the variable that the file should read to and write from.

Description

This function creates a file in debugfs with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no

automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If `debugfs` is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

Description

This function creates a file in `debugfs` with the given name that contains the value of the variable *value*. If the *mode* variable is so set, it can be read from, and written to.

This function will return a pointer to a dentry if it succeeds. This pointer must be passed to the `debugfs_remove` function when the file is to be removed (no automatic cleanup happens if your module is unloaded, you are responsible here.) If an error occurs, `NULL` will be returned.

If `debugfs` is not enabled in the kernel, the value `-ENODEV` will be returned. It is not wise to check for this value, but rather, check for `NULL` or `!NULL` instead as to eliminate the need for `#ifdef` in the calling code.

Chapter 7. The Linux VFS

7.1. The Directory Cache

d_invalidate

Name

`d_invalidate` — invalidate a dentry

Synopsis

```
int d_invalidate (struct dentry * dentry);
```

Arguments

dentry

dentry to invalidate

Description

Try to invalidate the dentry if it turns out to be possible. If there are other dentries that can be reached through this one we can't delete it and we return -EBUSY. On success we return 0.

no dcache lock.

shrink_dcache_sb

Name

`shrink_dcache_sb` — shrink dcache for a superblock

Synopsis

```
void shrink_dcache_sb (struct super_block * sb);
```

Arguments

sb

superblock

Description

Shrink the dcache for the specified super block. This is used to free the dcache before unmounting a file system

have_submounts

Name

`have_submounts` — check for mounts over a dentry

Synopsis

```
int have_submounts (struct dentry * parent);
```

Arguments

parent

dentry to check.

Description

Return true if the parent or its subdirectories contain a mount point

shrink_dcache_parent

Name

`shrink_dcache_parent` — prune dcache

Synopsis

```
void shrink_dcache_parent (struct dentry * parent);
```

Arguments

parent

parent of entries to prune

Description

Prune the dcache to remove unused children of the parent dentry.

d_alloc

Name

`d_alloc` — allocate a dcache entry

Synopsis

```
struct dentry * d_alloc (struct dentry * parent, const struct  
qstr * name);
```

Arguments

parent

parent of entry to allocate

name

qstr of the name

Description

Allocates a dentry. It returns `NULL` if there is insufficient memory available. On a success the dentry is returned. The name passed in is copied and the copy passed in may be reused after this call.

d_instantiate

Name

`d_instantiate` — fill in inode information for a dentry

Synopsis

```
void d_instantiate (struct dentry * entry, struct inode *  
inode);
```

Arguments

entry

dentry to complete

inode

inode to attach to this dentry

Description

Fill in inode information in the entry.

This turns negative dentries into productive full members of society.

NOTE! This assumes that the inode count has been incremented (or otherwise set) by the caller to indicate that it is now in use by the dcache.

d_instantiate_unique

Name

`d_instantiate_unique` — instantiate a non-aliased dentry

Synopsis

```
struct dentry * d_instantiate_unique (struct dentry * entry,  
struct inode * inode);
```

Arguments

entry

dentry to instantiate

inode

inode to attach to this dentry

Description

Fill in inode information in the entry. On success, it returns NULL. If an unhashed alias of “entry” already exists, then we return the aliased dentry instead.

Note that in order to avoid conflicts with `rename` etc, the caller had better be holding the parent directory semaphore.

d_alloc_root

Name

`d_alloc_root` — allocate root dentry

Synopsis

```
struct dentry * d_alloc_root (struct inode * root_inode);
```

Arguments

root_inode

inode to allocate the root for

Description

Allocate a root (“/”) dentry for the inode given. The inode is instantiated and returned. `NULL` is returned if there is insufficient memory or the inode passed is `NULL`.

d_alloc_anon

Name

`d_alloc_anon` — allocate an anonymous dentry

Synopsis

```
struct dentry * d_alloc_anon (struct inode * inode);
```

Arguments

inode

inode to allocate the dentry for

Description

This is similar to `d_alloc_root`. It is used by filesystems when creating a dentry for a given inode, often in the process of mapping a filehandle to a dentry. The returned dentry may be anonymous, or may have a full name (if the inode was already in the cache). The file system may need to make further efforts to connect this dentry into the dcache properly.

When called on a directory inode, we must ensure that the inode only ever has one dentry. If a dentry is found, that is returned instead of allocating a new one.

On successful return, the reference to the inode has been transferred to the dentry. If `NULL` is returned (indicating `kmalloc` failure), the reference on the inode has not been released.

d_splice_alias

Name

`d_splice_alias` — splice a disconnected dentry into the tree if one exists

Synopsis

```
struct dentry * d_splice_alias (struct inode * inode, struct
dentry * dentry);
```

Arguments

inode

the inode which may have a disconnected dentry

dentry

a negative dentry which we want to point to the inode.

Description

If *inode* is a directory and has a 'disconnected' dentry (i.e. IS_ROOT and DCACHE_DISCONNECTED), then `d_move` that in place of the given dentry and return it, else simply `d_add` the inode to the dentry and return NULL.

This is needed in the lookup routine of any filesystem that is exportable (via knfsd) so that we can build dcache paths to directories effectively.

If a dentry was found and moved, then it is returned. Otherwise NULL is returned. This matches the expected return value of `->lookup`.

d_lookup

Name

`d_lookup` — search for a dentry

Synopsis

```
struct dentry * d_lookup (struct dentry * parent, struct qstr  
* name);
```

Arguments

parent

parent dentry

name

qstr of name we wish to find

Description

Searches the children of the parent dentry for the name in question. If the dentry is found its reference count is incremented and the dentry is returned. The caller must use `d_put` to free the entry when it has finished using it. `NULL` is returned on failure.

`__d_lookup` is `dcache_lock` free. The hash list is protected using RCU. Memory barriers are used while updating and doing lockless traversal. To avoid races with `d_move` while rename is happening, `d_lock` is used.

Overflows in `memcmp`, while `d_move`, are avoided by keeping the length and name pointer in one structure pointed by `d_qstr`.

`rcu_read_lock` and `rcu_read_unlock` are used to disable preemption while lookup is going on.

`dentry_unused` list is not updated even if lookup finds the required dentry in there. It is updated in places such as `prune_dcache`, `shrink_dcache_sb`, `select_parent` and `__dget_locked`. This laziness saves lookup from `dcache_lock` acquisition.

`d_lookup` is protected against the concurrent renames in some unrelated directory using the `seqlock_t` `rename_lock`.

d_validate

Name

`d_validate` — verify dentry provided from insecure source

Synopsis

```
int d_validate (struct dentry * dentry, struct dentry *  
dparent);
```

Arguments

dentry

The dentry alleged to be valid child of *dparent*

dparent

The parent dentry (known to be valid)

Description

An insecure source has sent us a dentry, here we verify it and `dget` it. This is used by `ncpfs` in its `readdir` implementation. Zero is returned if the dentry is invalid.

d_delete

Name

`d_delete` — delete a dentry

Synopsis

```
void d_delete (struct dentry * dentry);
```

Arguments

dentry

The dentry to delete

Description

Turn the dentry into a negative dentry if possible, otherwise remove it from the hash queues so it can be deleted later

d_rehash

Name

`d_rehash` — add an entry back to the hash

Synopsis

```
void d_rehash (struct dentry * entry);
```

Arguments

entry

dentry to add to the hash

Description

Adds a dentry to the hash according to its name.

d_move

Name

d_move — move a dentry

Synopsis

```
void d_move (struct dentry * dentry, struct dentry * target);
```

Arguments

dentry

entry to move

target

new dentry

Description

Update the dcache to reflect the move of a file name. Negative dcache entries should not be moved in this way.

__d_path

Name

`__d_path` — return the path of a dentry

Synopsis

```
char * __d_path (struct dentry * dentry, struct vfsmount *  
vfsmnt, struct dentry * root, struct vfsmount * rootmnt, char  
* buffer, int buflen);
```

Arguments

dentry

dentry to report

vfsmnt

vfsmnt to which the dentry belongs

root

root dentry

rootmnt

vfsmnt to which the root dentry belongs

buffer

buffer to return value in

buflen

buffer length

Description

Convert a dentry into an ASCII path name. If the entry has been deleted the string “(deleted)” is appended. Note that this is ambiguous.

Returns the buffer or an error code if the path was too long.

“buflen” should be positive. Caller holds the dcache_lock.

find_inode_number

Name

`find_inode_number` — check for dentry with name

Synopsis

```
ino_t find_inode_number (struct dentry * dir, struct qstr *
name);
```

Arguments

dir

directory to check

name

Name to find.

Description

Check whether a dentry already exists for the given name, and return the inode number if it has an inode. Otherwise 0 is returned.

This routine is used to post-process directory listings for filesystems using synthetic inode numbers, and is necessary to keep `getcwd` working.

__d_drop

Name

`__d_drop` — drop a dentry

Synopsis

```
void __d_drop (struct dentry * dentry);
```

Arguments

dentry

dentry to drop

Description

`d_drop` unhashes the entry from the parent dentry hashes, so that it won't be found through a VFS lookup any more. Note that this is different from deleting the dentry - `d_delete` will try to mark the dentry negative if possible, giving a successful `_negative_` lookup, while `d_drop` will just make the cache lookup fail.

`d_drop` is used mainly for stuff that wants to invalidate a dentry for some reason (NFS timeouts or autofs deletes).

d_add

Name

`d_add` — add dentry to hash queues

Synopsis

```
void d_add (struct dentry * entry, struct inode * inode);
```

Arguments

entry

dentry to add

inode

The inode to attach to this dentry

Description

This adds the entry to the hash queues and initializes *inode*. The entry was actually filled in earlier during `d_alloc`.

d_add_unique

Name

`d_add_unique` — add dentry to hash queues without aliasing

Synopsis

```
struct dentry * d_add_unique (struct dentry * entry, struct
inode * inode);
```

Arguments

entry

dentry to add

inode

The inode to attach to this dentry

Description

This adds the entry to the hash queues and initializes *inode*. The entry was actually filled in earlier during `d_alloc`.

dget

Name

`dget` — get a reference to a dentry

Synopsis

```
struct dentry * dget (struct dentry * dentry);
```

Arguments

dentry

dentry to get a reference to

Description

Given a *dentry* or `NULL` pointer increment the reference count if appropriate and return the *dentry*. A *dentry* will not be destroyed when it has references. `dget` should never be called for *dentries* with zero reference counter. For these cases (preferably none, functions in `dcache.c` are sufficient for normal needs and they take necessary precautions) you should hold `dcache_lock` and call `dget_locked` instead of `dget`.

d_unhashed

Name

`d_unhashed` — is *dentry* hashed

Synopsis

```
int d_unhashed (struct dentry * dentry);
```

Arguments

dentry

entry to check

Description

Returns true if the dentry passed is not currently hashed.

7.2. Inode Handling

clear_inode

Name

`clear_inode` — clear an inode

Synopsis

```
void clear_inode (struct inode * inode);
```

Arguments

inode

inode to clear

Description

This is called by the filesystem to tell us that the inode is no longer useful. We just terminate it with extreme prejudice.

invalidate_inodes

Name

`invalidate_inodes` — discard the inodes on a device

Synopsis

```
int invalidate_inodes (struct super_block * sb);
```

Arguments

sb

superblock

Description

Discard all of the inodes for a given superblock. If the discard fails because there are busy inodes then a non zero value is returned. If the discard is successful all the inodes have been discarded.

new_inode

Name

`new_inode` — obtain an inode

Synopsis

```
struct inode * new_inode (struct super_block * sb);
```

Arguments

sb

superblock

Description

Allocates a new inode for given superblock.

iunique

Name

`iunique` — get a unique inode number

Synopsis

```
ino_t iunique (struct super_block * sb, ino_t max_reserved);
```


Arguments

sb

superblock

max_reserved

highest reserved inode number

Description

Obtain an inode number that is unique on the system for a given superblock. This is used by file systems that have no natural permanent inode numbering system. An inode number is returned that is higher than the reserved limit but unique.

BUGS

With a large number of inodes live on the file system this function currently becomes quite slow.

ilookup5

Name

`ilookup5` — search for an inode in the inode cache

Synopsis

```
struct inode * ilookup5 (struct super_block * sb, unsigned
long hashval, int (*test) (struct inode *, void *), void *
data);
```

Arguments

sb

super block of file system to search

hashval

hash value (usually inode number) to search for

test

callback used for comparisons between inodes

data

opaque data pointer to pass to *test*

Description

`ilookup5` uses `ifind` to search for the inode specified by *hashval* and *data* in the inode cache. This is a generalized version of `ilookup` for file systems where the inode number is not sufficient for unique identification of an inode.

If the inode is in the cache, the inode is returned with an incremented reference count.

Otherwise NULL is returned.

Note, *test* is called with the `inode_lock` held, so can't sleep.

ilookup

Name

`ilookup` — search for an inode in the inode cache

Synopsis

```
struct inode * ilookup (struct super_block * sb, unsigned long  
ino);
```

Arguments

sb

super block of file system to search

ino

inode number to search for

Description

`ilookup` uses `ifind_fast` to search for the inode *ino* in the inode cache. This is for file systems where the inode number is sufficient for unique identification of an inode.

If the inode is in the cache, the inode is returned with an incremented reference count.

Otherwise NULL is returned.

iget5_locked

Name

`iget5_locked` — obtain an inode from a mounted file system

Synopsis

```
struct inode * iget5_locked (struct super_block * sb, unsigned  
long hashval, int (*test) (struct inode *, void *), int (*set)  
(struct inode *, void *), void * data);
```

Arguments

sb

super block of file system

hashval

hash value (usually inode number) to get

test

callback used for comparisons between inodes

set

callback used to initialize a new struct inode

data

opaque data pointer to pass to *test* and *set*

Description

This is `iget` without the `read_inode` portion of `get_new_inode`.

`iget5_locked` uses `ifind` to search for the inode specified by *hashval* and *data* in the inode cache and if present it is returned with an increased reference count. This is a generalized version of `iget_locked` for file systems where the inode number is not sufficient for unique identification of an inode.

If the inode is not in cache, `get_new_inode` is called to allocate a new inode and this is returned locked, hashed, and with the `I_NEW` flag set. The file system gets to fill it in before unlocking it via `unlock_new_inode`.

Note both *test* and *set* are called with the `inode_lock` held, so can't sleep.

iget_locked

Name

`iget_locked` — obtain an inode from a mounted file system

Synopsis

```
struct inode * iget_locked (struct super_block * sb, unsigned  
long ino);
```

Arguments

sb

super block of file system

ino

inode number to get

Description

This is `iget` without the `read_inode` portion of `get_new_inode_fast`.

`iget_locked` uses `ifind_fast` to search for the inode specified by *ino* in the inode cache and if present it is returned with an increased reference count. This is for file systems where the inode number is sufficient for unique identification of an inode.

If the inode is not in cache, `get_new_inode_fast` is called to allocate a new inode and this is returned locked, hashed, and with the `I_NEW` flag set. The file system gets to fill it in before unlocking it via `unlock_new_inode`.

__insert_inode_hash

Name

`__insert_inode_hash` — hash an inode

Synopsis

```
void __insert_inode_hash (struct inode * inode, unsigned long
hashval);
```

Arguments

inode

unhashed inode

hashval

unsigned long value used to locate this object in the `inode_hashtable`.

Description

Add an inode to the inode hash for this superblock.

remove_inode_hash

Name

`remove_inode_hash` — remove an inode from the hash

Synopsis

```
void remove_inode_hash (struct inode * inode);
```

Arguments

inode

inode to unhash

Description

Remove an inode from the superblock.

iput

Name

`iput` — put an inode

Synopsis

```
void iput (struct inode * inode);
```

Arguments

inode

inode to put

Description

Puts an inode, dropping its usage count. If the inode use count hits zero the inode is also then freed and may be destroyed.

bmap

Name

bmap — find a block number in a file

Synopsis

```
sector_t bmap (struct inode * inode, sector_t block);
```

Arguments

inode

inode of file

block

block to find

Description

Returns the block number on the device holding the inode that is the disk block number for the block of the file requested. That is, asked for block 4 of inode 1 the function will return the disk block relative to the disk start that holds that block of the file.

update_atime

Name

`update_atime` — update the access time

Synopsis

```
void update_atime (struct inode * inode);
```

Arguments

inode

inode accessed

Description

Update the accessed time on an inode and mark it for writeback. This function automatically handles read only file systems and media, as well as the “noatime” flag and inode specific “noatime” markers.

inode_update_time

Name

`inode_update_time` — update mtime and ctime time

Synopsis

```
void inode_update_time (struct inode * inode, int ctime_too);
```

Arguments

inode

inode accessed

ctime_too

update ctime too

Description

Update the mtime time on an inode and mark it for writeback. When *ctime_too* is specified update the ctime too.

make_bad_inode

Name

`make_bad_inode` — mark an inode bad due to an I/O error

Synopsis

```
void make_bad_inode (struct inode * inode);
```

Arguments

inode

Inode to mark bad

Description

When an inode cannot be read due to a media or remote network failure this function makes the inode “bad” and causes I/O operations on it to fail from this point on.

is_bad_inode

Name

`is_bad_inode` — is an inode errored

Synopsis

```
int is_bad_inode (struct inode * inode);
```

Arguments

inode

inode to test

Description

Returns true if the inode in question has been marked as bad.

7.3. Registration and Superblocks

deactivate_super

Name

`deactivate_super` — drop an active reference to superblock

Synopsis

```
void deactivate_super (struct super_block * s);
```

Arguments

s

superblock to deactivate

Description

Drops an active reference to superblock, acquiring a temporary one if there is no active references left. In that case we lock superblock, tell fs driver to shut it down and drop the temporary reference we had just acquired.

generic_shutdown_super

Name

`generic_shutdown_super` — common helper for `->kill_sb`

Synopsis

```
void generic_shutdown_super (struct super_block * sb);
```

Arguments

sb

superblock to kill

Description

`generic_shutdown_super` does all fs-independent work on superblock shutdown. Typical `->kill_sb` should pick all fs-specific objects that need destruction out of superblock, call `generic_shutdown_super` and release aforementioned objects. Note: dentries and inodes `_are_` taken care of and do not need specific handling.

sget

Name

`sget` — find or create a superblock

Synopsis

```
struct super_block * sget (struct file_system_type * type, int  
(*test) (struct super_block *,void *), int (*set) (struct  
super_block *,void *), void * data);
```

Arguments

type

filesystem type superblock should belong to

test

comparison callback

set

setup callback

data

argument to each of them

get_super

Name

`get_super` — get the superblock of a device

Synopsis

```
struct super_block * get_super (struct block_device * bdev);
```

Arguments

bdev

device to get the superblock for

Description

Scans the superblock list and finds the superblock of the file system mounted on the device given. `NULL` is returned if no match is found.

7.4. File Locks

`posix_lock_file`

Name

`posix_lock_file` — Apply a POSIX-style lock to a file

Synopsis

```
int posix_lock_file (struct file * filp, struct file_lock *  
fl);
```

Arguments

filp

The file to apply the lock to

fl

The lock to be applied

Description

Add a POSIX style lock to a file. We merge adjacent & overlapping locks whenever possible. POSIX locks are sorted by owner task, then by starting address

posix_lock_file_wait

Name

`posix_lock_file_wait` — Apply a POSIX-style lock to a file

Synopsis

```
int posix_lock_file_wait (struct file * filp, struct file_lock  
* fl);
```

Arguments

filp

The file to apply the lock to

fl

The lock to be applied

Description

Add a POSIX style lock to a file. We merge adjacent & overlapping locks whenever possible. POSIX locks are sorted by owner task, then by starting address

locks_mandatory_area

Name

`locks_mandatory_area` — Check for a conflicting lock

Synopsis

```
int locks_mandatory_area (int read_write, struct inode *
inode, struct file * filp, loff_t offset, size_t count);
```

Arguments

read_write

FLOCK_VERIFY_WRITE for exclusive access, FLOCK_VERIFY_READ for shared

inode

the file to check

filp

how the file was opened (if it was)

offset

start of area to check

count

length of area to check

Description

Searches the inode's list of locks to find any POSIX locks which conflict. This function is called from `rw_verify_area` and `locks_verify_truncate`.

`__break_lease`

Name

`__break_lease` — revoke all outstanding leases on file

Synopsis

```
int __break_lease (struct inode * inode, unsigned int mode);
```

Arguments

inode

the inode of the file to return

mode

the open mode (read or write)

Description

`break_lease` (inlined for speed) has checked there already is a lease on this file. Leases are broken on a call to `open` or `truncate`. This function can sleep unless you specified `O_NONBLOCK` to your `open`.

lease_get_mtime

Name

`lease_get_mtime` —

Synopsis

```
void lease_get_mtime (struct inode * inode, struct timespec *
time);
```

Arguments

inode

the inode

time

pointer to a timespec which will contain the last modified time

Description

This is to force NFS clients to flush their caches for files with exclusive leases. The justification is that if someone has an exclusive lease, then they could be modifying it.

flock_lock_file_wait

Name

`flock_lock_file_wait` — Apply a FLOCK-style lock to a file

Synopsis

```
int flock_lock_file_wait (struct file * filp, struct file_lock
* fl);
```

Arguments

filp

The file to apply the lock to

fl

The lock to be applied

Description

Add a FLOCK style lock to a file.

posix_block_lock

Name

posix_block_lock — blocks waiting for a file lock

Synopsis

```
void posix_block_lock (struct file_lock * blocker, struct
file_lock * waiter);
```

Arguments

blocker

the lock which is blocking

waiter

the lock which conflicts and has to wait

Description

lockd needs to block waiting for locks.

posix_unblock_lock

Name

posix_unblock_lock — stop waiting for a file lock

Synopsis

```
void posix_unblock_lock (struct file * filp, struct file_lock  
* waiter);
```

Arguments

filp

how the file was opened

waiter

the lock which was waiting

Description

lockd needs to block waiting for locks.

lock_may_read

Name

`lock_may_read` — checks that the region is free of locks

Synopsis

```
int lock_may_read (struct inode * inode, loff_t start,  
unsigned long len);
```

Arguments

inode

the inode that is being read

start

the first byte to read

len

the number of bytes to read

Description

Emulates Windows locking requirements. Whole-file mandatory locks (share modes) can prohibit a read and byte-range POSIX locks can prohibit a read if they overlap.

N.B. this function is only ever called from knfsd and ownership of locks is never checked.

lock_may_write

Name

`lock_may_write` — checks that the region is free of locks

Synopsis

```
int lock_may_write (struct inode * inode, loff_t start,  
unsigned long len);
```

Arguments

inode

the inode that is being written

start

the first byte to write

len

the number of bytes to write

Description

Emulates Windows locking requirements. Whole-file mandatory locks (share modes) can prohibit a write and byte-range POSIX locks can prohibit a write if they overlap.

N.B. this function is only ever called from knfsd and ownership of locks is never checked.

locks_mandatory_locked

Name

`locks_mandatory_locked` — Check for an active lock

Synopsis

```
int locks_mandatory_locked (struct inode * inode);
```

Arguments

inode

the file to check

Description

Searches the inode's list of locks to find any POSIX locks which conflict. This function is called from `locks_verify_locked` only.

fcntl_getlease

Name

`fcntl_getlease` — Enquire what lease is currently active

Synopsis

```
int fcntl_getlease (struct file * filp);
```

Arguments

filp

the file

Description

The value returned by this function will be one of (if no lease break is pending):

`F_RDLCK` to indicate a shared lease is held.

`F_WRLCK` to indicate an exclusive lease is held.

`F_UNLCK` to indicate no lease is held.

(if a lease break is pending):

`F_RDLCK` to indicate an exclusive lease needs to be changed to a shared lease (or removed).

`F_UNLCK` to indicate the lease needs to be removed.

XXX

sfr & willy disagree over whether `F_INPROGRESS` should be returned to userspace.

__setlease

Name

`__setlease` — sets a lease on an open file

Synopsis

```
int __setlease (struct file * filp, long arg, struct file_lock  
** flp);
```

Arguments

filp

file pointer

arg

type of lease to obtain

flp

input - file_lock to use, output - file_lock inserted

Description

The (input) `flp->fl_lmops->fl_break` function is required by `break_lease`.

Called with kernel lock held.

fcntl_setlease

Name

`fcntl_setlease` — sets a lease on an open file

Synopsis

```
int fcntl_setlease (unsigned int fd, struct file * filp, long  
arg);
```

Arguments

fd

open file descriptor

filp

file pointer

arg

type of lease to obtain

Description

Call this `fcntl` to establish a lease on the file. Note that you also need to call `F_SETSIG` to receive a signal when the lease is broken.

sys_flock

Name

`sys_flock` — flock system call.

Synopsis

```
asmlinkage long sys_flock (unsigned int fd, unsigned int cmd);
```

Arguments

fd

the file descriptor to lock.

cmd

the type of lock to apply.

Description

Apply a `FL_FLOCK` style lock to an open file descriptor. The *cmd* can be one of

`LOCK_SH` -- a shared lock.

`LOCK_EX` -- an exclusive lock.

`LOCK_UN` -- remove an existing lock.

`LOCK_MAND` -- a ‘mandatory’ flock. This exists to emulate Windows Share Modes.

`LOCK_MAND` can be combined with `LOCK_READ` or `LOCK_WRITE` to allow other processes read and write access respectively.

get_locks_status

Name

`get_locks_status` — reports lock usage in `/proc/locks`

Synopsis

```
int get_locks_status (char * buffer, char ** start, off_t  
offset, int length);
```

Arguments

buffer

address in userspace to write into

start

?

offset

how far we are through the buffer

length

how much to read

Chapter 8. Linux Networking

8.1. Socket Buffer Functions

struct sk_buff

Name

struct sk_buff — socket buffer

Synopsis

```
struct sk_buff {
    struct sk_buff * next;
    struct sk_buff * prev;
    struct sk_buff_head * list;
    struct sock * sk;
    struct timeval stamp;
    struct net_device * dev;
    struct net_device * input_dev;
    struct net_device * real_dev;
    union h;
    union nh;
    union mac;
    struct dst_entry * dst;
    char cb[40];
    unsigned int len;
    unsigned int data_len;
    unsigned int mac_len;
    unsigned int csum;
    unsigned char cloned;
    unsigned char pkt_type;
    unsigned char ip_summed;
    __u32 priority;
    unsigned short protocol;
    unsigned short security;
    void (* destructor) (struct sk_buff *skb);
#ifdef CONFIG_NETFILTER
    unsigned long nfmark;
    __u32 nfcache;
#endif
}
```

```
    __u32 nfctinfo;
    struct nf_conntrack * nfct;
#ifdef CONFIG_NETFILTER_DEBUG
    unsigned int nf_debug;
#endif
#ifdef CONFIG_BRIDGE_NETFILTER
    struct nf_bridge_info * nf_bridge;
#endif
#endif
#if defined(CONFIG_HIPPI)
    union private;
#endif
#ifdef CONFIG_NET_SCHED
    __u32 tc_index;
#endif
#ifdef CONFIG_NET_CLS_ACT
#endif
#endif
    unsigned int truesize;
    atomic_t users;
    unsigned char * head;
    unsigned char * data;
    unsigned char * tail;
    unsigned char * end;
};
```

Members

next

Next buffer in list

prev

Previous buffer in list

list

List we are on

sk

Socket we are owned by

stamp

Time we arrived

dev
Device we arrived on/are leaving by

input_dev
Device we arrived on

real_dev
The real device we are using

h
Transport layer header

nh
Network layer header

mac
Link layer header

dst
FIXME: Describe this field

cb[40]
Control buffer. Free for use by every layer. Put private vars here

len
Length of actual data

data_len
Data length

mac_len
Length of link layer header

csum
Checksum

cloned
Head may be cloned (check refcnt to be sure)

pkt_type
Packet class

ip_summed

Driver fed us an IP checksum

priority

Packet queueing priority

protocol

Packet protocol from driver

security

Security level of packet

destructor

Destruct function

nfmark

Can be used for communication between hooks

nfcache

Cache info

nfctinfo

Relationship of this skb to the connection

nfct

Associated connection, if any

nf_debug

Netfilter debugging

nf_bridge

Saved data about a bridged frame - see br_netfilter.c

private

Data which is private to the HIPPI implementation

tc_index

Traffic control index

true_size

Buffer size

users

User count - see {datagram,tcp}.c

head

Head of buffer

data

Data head pointer

tail

Tail pointer

end

End pointer

skb_queue_empty

Name

`skb_queue_empty` — check if a queue is empty

Synopsis

```
int skb_queue_empty (const struct sk_buff_head * list);
```

Arguments

list

queue head

Description

Returns true if the queue is empty, false otherwise.

skb_get

Name

`skb_get` — reference buffer

Synopsis

```
struct sk_buff * skb_get (struct sk_buff * skb);
```

Arguments

skb

buffer to reference

Description

Makes another reference to a socket buffer and returns a pointer to the buffer.

kfree_skb

Name

kfree_skb — free an sk_buff

Synopsis

```
void kfree_skb (struct sk_buff * skb);
```

Arguments

skb

buffer to free

Description

Drop a reference to the buffer and free it if the usage count has hit zero.

skb_cloned

Name

skb_cloned — is the buffer a clone

Synopsis

```
int skb_cloned (const struct sk_buff * skb);
```

Arguments

skb

buffer to check

Description

Returns true if the buffer was generated with `skb_clone` and is one of multiple shared copies of the buffer. Cloned buffers are shared data so must not be written to under normal circumstances.

skb_shared

Name

`skb_shared` — is the buffer shared

Synopsis

```
int skb_shared (const struct sk_buff * skb);
```

Arguments

skb

buffer to check

Description

Returns true if more than one person has a reference to this buffer.

skb_share_check

Name

`skb_share_check` — check if buffer is shared and if so clone it

Synopsis

```
struct sk_buff * skb_share_check (struct sk_buff * skb, int pri);
```

Arguments

skb

buffer to check

pri

priority for memory allocation

Description

If the buffer is shared the buffer is cloned and the old copy drops a reference. A new clone with a single reference is returned. If the buffer is not shared the original buffer is returned. When being called from interrupt status or with spinlocks held *pri* must be GFP_ATOMIC.

NULL is returned on a memory allocation failure.

skb_unshare

Name

`skb_unshare` — make a copy of a shared buffer

Synopsis

```
struct sk_buff * skb_unshare (struct sk_buff * skb, int pri);
```

Arguments

skb

buffer to check

pri

priority for memory allocation

Description

If the socket buffer is a clone then this function creates a new copy of the data, drops a reference count on the old copy and returns the new copy with the reference count at 1. If the buffer is not a clone the original buffer is returned. When called with a spinlock held or from interrupt state *pri* must be `GFP_ATOMIC`

`NULL` is returned on a memory allocation failure.

skb_peek

Name

skb_peek —

Synopsis

```
struct sk_buff * skb_peek (struct sk_buff_head * list_);
```

Arguments

list_

list to peek at

Description

Peek an &sk_buff. Unlike most other operations you **_MUST_** be careful with this one. A peek leaves the buffer on the list and someone else may run off with it. You must hold the appropriate locks or have a private queue to do this.

Returns `NULL` for an empty list or a pointer to the head element. The reference count is not incremented and the reference is therefore volatile. Use with caution.

skb_peek_tail

Name

skb_peek_tail —

Synopsis

```
struct sk_buff * skb_peek_tail (struct sk_buff_head * list_);
```

Arguments

list_

list to peek at

Description

Peek an &sk_buff. Unlike most other operations you **_MUST_** be careful with this one. A peek leaves the buffer on the list and someone else may run off with it. You must hold the appropriate locks or have a private queue to do this.

Returns `NULL` for an empty list or a pointer to the tail element. The reference count is not incremented and the reference is therefore volatile. Use with caution.

skb_queue_len

Name

skb_queue_len — get queue length

Synopsis

```
__u32 skb_queue_len (const struct sk_buff_head * list_);
```

Arguments

list_

list to measure

Description

Return the length of an `&sk_buff` queue.

skb_put

Name

`skb_put` — add data to a buffer

Synopsis

```
unsigned char * skb_put (struct sk_buff * skb, unsigned int  
len);
```

Arguments

skb

buffer to use

len

amount of data to add

Description

This function extends the used data area of the buffer. If this would exceed the total buffer size the kernel will panic. A pointer to the first byte of the extra data is returned.

skb_push

Name

`skb_push` — add data to the start of a buffer

Synopsis

```
unsigned char * skb_push (struct sk_buff * skb, unsigned int  
len);
```

Arguments

skb

buffer to use

len

amount of data to add

Description

This function extends the used data area of the buffer at the buffer start. If this would exceed the total buffer headroom the kernel will panic. A pointer to the first byte of the extra data is returned.

skb_pull

Name

`skb_pull` — remove data from the start of a buffer

Synopsis

```
unsigned char * skb_pull (struct sk_buff * skb, unsigned int  
len);
```

Arguments

skb

buffer to use

len

amount of data to remove

Description

This function removes data from the start of a buffer, returning the memory to the headroom. A pointer to the next data in the buffer is returned. Once the data has been pulled future pushes will overwrite the old data.

skb_headroom

Name

`skb_headroom` — bytes at buffer head

Synopsis

```
int skb_headroom (const struct sk_buff * skb);
```

Arguments

skb

buffer to check

Description

Return the number of bytes of free space at the head of an `&sk_buff`.

skb_tailroom

Name

`skb_tailroom` — bytes at buffer end

Synopsis

```
int skb_tailroom (const struct sk_buff * skb);
```

Arguments

skb

buffer to check

Description

Return the number of bytes of free space at the tail of an `sk_buff`

skb_reserve

Name

`skb_reserve` — adjust headroom

Synopsis

```
void skb_reserve (struct sk_buff * skb, unsigned int len);
```

Arguments

skb

buffer to alter

len

bytes to move

Description

Increase the headroom of an empty `&sk_buff` by reducing the tail room. This is only allowed for an empty buffer.

skb_trim

Name

`skb_trim` — remove end from a buffer

Synopsis

```
void skb_trim (struct sk_buff * skb, unsigned int len);
```

Arguments

skb

buffer to alter

len

new length

Description

Cut the length of a buffer down by removing data from the tail. If the buffer is already under the length specified it is not modified.

skb_orphan

Name

`skb_orphan` — orphan a buffer

Synopsis

```
void skb_orphan (struct sk_buff * skb);
```

Arguments

skb

buffer to orphan

Description

If a buffer currently has an owner then we call the owner's destructor function and make the *skb* unowned. The buffer continues to exist but is no longer charged to its former owner.

__dev_alloc_skb

Name

`__dev_alloc_skb` — allocate an skbuff for sending

Synopsis

```
struct sk_buff * __dev_alloc_skb (unsigned int length, int  
gfp_mask);
```

Arguments

length

length to allocate

gfp_mask

get_free_pages mask, passed to alloc_skb

Description

Allocate a new &sk_buff and assign it a usage count of one. The buffer has unspecified headroom built in. Users should allocate the headroom they think they need without accounting for the built in space. The built in space is used for optimisations.

NULL is returned if there is no free memory.

dev_alloc_skb

Name

dev_alloc_skb — allocate an skbuff for sending

Synopsis

```
struct sk_buff * dev_alloc_skb (unsigned int length);
```

Arguments

length

length to allocate

Description

Allocate a new `&sk_buff` and assign it a usage count of one. The buffer has unspecified headroom built in. Users should allocate the headroom they think they need without accounting for the built in space. The built in space is used for optimisations.

`NULL` is returned if there is no free memory. Although this function allocates memory it can be called from an interrupt.

skb_cow

Name

`skb_cow` — copy header of `skb` when it is required

Synopsis

```
int skb_cow (struct sk_buff * skb, unsigned int headroom);
```

Arguments

skb

buffer to cow

headroom

needed headroom

Description

If the `skb` passed lacks sufficient headroom or its data part is shared, data is reallocated. If reallocation fails, an error is returned and original `skb` is not changed.

The result is `skb` with writable area `skb->head...skb->tail` and at least *headroom* of space at head.

skb_padto

Name

`skb_padto` — pad an skbuff up to a minimal size

Synopsis

```
struct sk_buff * skb_padto (struct sk_buff * skb, unsigned int  
len);
```

Arguments

skb

buffer to pad

len

minimal length

Description

Pads up a buffer to ensure the trailing bytes exist and are blanked. If the buffer already contains sufficient data it is untouched. Returns the buffer, which may be a replacement for the original, or NULL for out of memory - in which case the original buffer is still freed.

skb_over_panic

Name

`skb_over_panic` — private function

Synopsis

```
void skb_over_panic (struct sk_buff * skb, int sz, void *  
here);
```

Arguments

skb

buffer

sz

size

here

address

Description

Out of line support code for `skb_put`. Not user callable.

skb_under_panic

Name

`skb_under_panic` — private function

Synopsis

```
void skb_under_panic (struct sk_buff * skb, int sz, void *  
here);
```

Arguments

skb

buffer

sz

size

here

address

Description

Out of line support code for `skb_push`. Not user callable.

alloc_skb

Name

`alloc_skb` — allocate a network buffer

Synopsis

```
struct sk_buff * alloc_skb (unsigned int size, int gfp_mask);
```

Arguments

size

size to allocate

gfp_mask

allocation mask

Description

Allocate a new `&sk_buff`. The returned buffer has no headroom and a tail room of `size` bytes. The object has a reference count of one. The return is the buffer. On a failure the return is `NULL`.

Buffers may only be allocated from interrupts using a *gfp_mask* of `GFP_ATOMIC`.

__kfree_skb

Name

`__kfree_skb` — private function

Synopsis

```
void __kfree_skb (struct sk_buff * skb);
```

Arguments

skb

buffer

Description

Free an `sk_buff`. Release anything attached to the buffer. Clean the state. This is an internal helper function. Users should always call `kfree_skb`

skb_clone

Name

`skb_clone` — duplicate an `sk_buff`

Synopsis

```
struct sk_buff * skb_clone (struct sk_buff * skb, int  
gfp_mask);
```


Arguments

skb

buffer to clone

gfp_mask

allocation priority

Description

Duplicate an `&sk_buff`. The new one is not owned by a socket. Both copies share the same packet data but not structure. The new buffer has a reference count of 1. If the allocation fails the function returns `NULL` otherwise the new buffer is returned.

If this function is called from an interrupt `gfp_mask` must be `GFP_ATOMIC`.

skb_copy

Name

`skb_copy` — create private copy of an `sk_buff`

Synopsis

```
struct sk_buff * skb_copy (const struct sk_buff * skb, int
gfp_mask);
```

Arguments

skb

buffer to copy

gfp_mask

allocation priority

Description

Make a copy of both an `&sk_buff` and its data. This is used when the caller wishes to modify the data and needs a private copy of the data to alter. Returns `NULL` on failure or the pointer to the buffer on success. The returned buffer has a reference count of 1.

As by-product this function converts non-linear `&sk_buff` to linear one, so that `&sk_buff` becomes completely private and caller is allowed to modify all the data of returned buffer. This means that this function is not recommended for use in circumstances when only header is going to be modified. Use `pskb_copy` instead.

pskb_copy

Name

`pskb_copy` — create copy of an `sk_buff` with private head.

Synopsis

```
struct sk_buff * pskb_copy (struct sk_buff * skb, int  
gfp_mask);
```

Arguments

skb

buffer to copy

gfp_mask

allocation priority

Description

Make a copy of both an `&sk_buff` and part of its data, located in header. Fragmented data remain shared. This is used when the caller wishes to modify only header of `&sk_buff` and needs private copy of the header to alter. Returns `NULL` on failure or the pointer to the buffer on success. The returned buffer has a reference count of 1.

pskb_expand_head

Name

`pskb_expand_head` — reallocate header of `sk_buff`

Synopsis

```
int pskb_expand_head (struct sk_buff * skb, int nhead, int
ntail, int gfp_mask);
```

Arguments

skb

buffer to reallocate

nhead

room to add at head

ntail

room to add at tail

gfp_mask

allocation priority

Description

Expands (or creates identical copy, if *&nhead* and *&ntail* are zero) header of *skb*. *&sk_buff* itself is not changed. *&sk_buff* MUST have reference count of 1. Returns zero in the case of success or error, if expansion failed. In the last case, *&sk_buff* is not changed.

All the pointers pointing into *skb* header may change and must be reloaded after call to this function.

skb_copy_expand

Name

skb_copy_expand — copy and expand *sk_buff*

Synopsis

```
struct sk_buff * skb_copy_expand (const struct sk_buff * skb,  
int newheadroom, int newtailroom, int gfp_mask);
```

Arguments

skb

buffer to copy

newheadroom

new free bytes at head

newtailroom

new free bytes at tail

gfp_mask

allocation priority

Description

Make a copy of both an `&sk_buff` and its data and while doing so allocate additional space.

This is used when the caller wishes to modify the data and needs a private copy of the data to alter as well as more space for new fields. Returns `NULL` on failure or the pointer to the buffer on success. The returned buffer has a reference count of 1.

You must pass `GFP_ATOMIC` as the allocation priority if this function is called from an interrupt.

BUG ALERT

`ip_summed` is not copied. Why does this work? Is it used only by netfilter in the cases when checksum is recalculated? --ANK

skb_pad

Name

`skb_pad` — zero pad the tail of an `skb`

Synopsis

```
struct sk_buff * skb_pad (struct sk_buff * skb, int pad);
```

Arguments

skb

buffer to pad

pad

space to pad

Description

Ensure that a buffer is followed by a padding area that is zero filled. Used by network drivers which may DMA or transfer data beyond the buffer end onto the wire.

May return NULL in out of memory cases.

__pskb_pull_tail

Name

`__pskb_pull_tail` — advance tail of skb header

Synopsis

```
unsigned char * __pskb_pull_tail (struct sk_buff * skb, int  
delta);
```

Arguments

skb

buffer to reallocate

delta

number of bytes to advance tail

Description

The function makes a sense only on a fragmented `&sk_buff`, it expands header moving its tail forward and copying necessary data from fragmented part.

`&sk_buff` MUST have reference count of 1.

Returns `NULL` (and `&sk_buff` does not change) if pull failed or value of new tail of `skb` in the case of success.

All the pointers pointing into `skb` header may change and must be reloaded after call to this function.

skb_dequeue

Name

`skb_dequeue` — remove from the head of the queue

Synopsis

```
struct sk_buff * skb_dequeue (struct sk_buff_head * list);
```

Arguments

list

list to dequeue from

Description

Remove the head of the list. The list lock is taken so the function may be used safely with other locking list functions. The head item is returned or `NULL` if the list is empty.

skb_dequeue_tail

Name

`skb_dequeue_tail` — remove from the tail of the queue

Synopsis

```
struct sk_buff * skb_dequeue_tail (struct sk_buff_head *  
list);
```

Arguments

list

list to dequeue from

Description

Remove the tail of the list. The list lock is taken so the function may be used safely with other locking list functions. The tail item is returned or `NULL` if the list is empty.

skb_queue_purge

Name

`skb_queue_purge` — empty a list

Synopsis

```
void skb_queue_purge (struct sk_buff_head * list);
```

Arguments

list

list to empty

Description

Delete all buffers on an `&sk_buff` list. Each buffer is removed from the list and one reference dropped. This function takes the list lock and is atomic with respect to other list locking functions.

skb_queue_head

Name

`skb_queue_head` — queue a buffer at the list head

Synopsis

```
void skb_queue_head (struct sk_buff_head * list, struct  
sk_buff * newsk);
```

Arguments

list

list to use

newsk

buffer to queue

Description

Queue a buffer at the start of the list. This function takes the list lock and can be used safely with other locking &sk_buff functions safely.

A buffer cannot be placed on two lists at the same time.

skb_queue_tail

Name

`skb_queue_tail` — queue a buffer at the list tail

Synopsis

```
void skb_queue_tail (struct sk_buff_head * list, struct  
sk_buff * newsk);
```

Arguments

list

list to use

newsk

buffer to queue

Description

Queue a buffer at the tail of the list. This function takes the list lock and can be used safely with other locking &sk_buff functions safely.

A buffer cannot be placed on two lists at the same time.

skb_unlink

Name

skb_unlink — remove a buffer from a list

Synopsis

```
void skb_unlink (struct sk_buff * skb);
```

Arguments

skb

buffer to remove

Description

Place a packet after a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls

Works even without knowing the list it is sitting on, which can be handy at times. It also means that THE LIST MUST EXIST when you unlink. Thus a list must have its contents unlinked before it is destroyed.

skb_append

Name

`skb_append` — append a buffer

Synopsis

```
void skb_append (struct sk_buff * old, struct sk_buff *  
newsk);
```

Arguments

old

buffer to insert after

newsk

buffer to insert

Description

Place a packet after a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls. A buffer cannot be placed on two lists at the same time.

skb_insert

Name

`skb_insert` — insert a buffer

Synopsis

```
void skb_insert (struct sk_buff * old, struct sk_buff *  
newsk);
```

Arguments

old

buffer to insert before

newsk

buffer to insert

Description

Place a packet before a given packet in a list. The list locks are taken and this function is atomic with respect to other list locked calls. A buffer cannot be placed on two lists at the same time.

skb_split

Name

`skb_split` — Split fragmented `skb` to two parts at length `len`.

Synopsis

```
void skb_split (struct sk_buff * skb, struct sk_buff * skb1,  
const u32 len);
```

Arguments

skb

-- undescribed --

skb1

-- undescribed --

len

-- undescribed --

8.2. Socket Filter

sk_run_filter

Name

`sk_run_filter` — run a filter on a socket

Synopsis

```
int sk_run_filter (struct sk_buff * skb, struct sock_filter *
filter, int flen);
```

Arguments

skb

buffer to run the filter on

filter

filter to apply

flen

length of filter

Description

Decode and apply filter instructions to the `skb->data`. Return length to keep, 0 for none. `skb` is the data we are filtering, `filter` is the array of filter instructions, and `len` is the number of filter blocks in the array.

sk_chk_filter

Name

`sk_chk_filter` — verify socket filter code

Synopsis

```
int sk_chk_filter (struct sock_filter * filter, int flen);
```

Arguments

filter

filter to verify

flen

length of filter

Description

Check the user's filter code. If we let some ugly filter code slip through kaboom! The filter must contain no references or jumps that are out of range, no illegal instructions and no backward jumps. It must end with a RET instruction

Returns 0 if the rule set is legal or a negative errno code if not.

8.3. Generic Network Statistics

struct gnet_stats_basic

Name

struct gnet_stats_basic — byte/packet throughput statistics

Synopsis

```
struct gnet_stats_basic {  
    __u64 bytes;  
    __u32 packets;  
};
```


Members

bytes

number of seen bytes

packets

number of seen packets

struct gnet_stats_rate_est

Name

struct gnet_stats_rate_est — rate estimator

Synopsis

```
struct gnet_stats_rate_est {  
    __u32 bps;  
    __u32 pps;  
};
```

Members

bps

current byte rate

pps

current packet rate

struct gnet_stats_queue

Name

struct gnet_stats_queue — queuing statistics

Synopsis

```
struct gnet_stats_queue {  
    __u32 qlen;  
    __u32 backlog;  
    __u32 drops;  
    __u32 requeues;  
    __u32 overlimits;  
};
```

Members

qlen

queue length

backlog

backlog size of queue

drops

number of dropped packets

requeues

number of requeues

overlimits

number of enqueues over the limit

struct gnet_estimator

Name

`struct gnet_estimator` — rate estimator configuration

Synopsis

```
struct gnet_estimator {
    signed char interval;
    unsigned char ewma_log;
};
```

Members

`interval`

sampling period

`ewma_log`

the log of measurement window weight

gnet_stats_start_copy_compat

Name

`gnet_stats_start_copy_compat` — start dumping procedure in compatibility mode

Synopsis

```
int gnet_stats_start_copy_compat (struct sk_buff * skb, int
type, int tc_stats_type, int xstats_type, spinlock_t * lock,
```

```
struct gnet_dump * d);
```

Arguments

skb

socket buffer to put statistics TLVs into

type

TLV type for top level statistic TLV

tc_stats_type

TLV type for backward compatibility struct tc_stats TLV

xstats_type

TLV type for backward compatibility xstats TLV

lock

statistics lock

d

dumping handle

Description

Initializes the dumping handle, grabs the statistic lock and appends an empty TLV header to the socket buffer for use a container for all other statistic TLVS.

The dumping handle is marked to be in backward compatibility mode telling all `gnet_stats_copy_XXX` functions to fill a local copy of struct `tc_stats`.

Returns 0 on success or -1 if the room in the socket buffer was not sufficient.

gnet_stats_start_copy

Name

`gnet_stats_start_copy` — start dumping procedure in compatibility mode

Synopsis

```
int gnet_stats_start_copy (struct sk_buff * skb, int type,  
spinlock_t * lock, struct gnet_dump * d);
```

Arguments

skb

socket buffer to put statistics TLVs into

type

TLV type for top level statistic TLV

lock

statistics lock

d

dumping handle

Description

Initializes the dumping handle, grabs the statistic lock and appends an empty TLV header to the socket buffer for use a container for all other statistic TLVS.

Returns 0 on success or -1 if the room in the socket buffer was not sufficient.

gnet_stats_copy_basic

Name

`gnet_stats_copy_basic` — copy basic statistics into statistic TLV

Synopsis

```
int gnet_stats_copy_basic (struct gnet_dump * d, struct  
gnet_stats_basic * b);
```

Arguments

d

dumping handle

b

basic statistics

Description

Appends the basic statistics to the top level TLV created by `gnet_stats_start_copy`.

Returns 0 on success or -1 with the statistic lock released if the room in the socket buffer was not sufficient.

gnet_stats_copy_rate_est

Name

`gnet_stats_copy_rate_est` — copy rate estimator statistics into statistics TLV

Synopsis

```
int gnet_stats_copy_rate_est (struct gnet_dump * d, struct  
gnet_stats_rate_est * r);
```

Arguments

d

dumping handle

r

rate estimator statistics

Description

Appends the rate estimator statistics to the top level TLV created by `gnet_stats_start_copy`.

Returns 0 on success or -1 with the statistic lock released if the room in the socket buffer was not sufficient.

gnet_stats_copy_queue

Name

`gnet_stats_copy_queue` — copy queue statistics into statistics TLV

Synopsis

```
int gnet_stats_copy_queue (struct gnet_dump * d, struct  
gnet_stats_queue * q);
```

Arguments

d

dumping handle

q

queue statistics

Description

Appends the queue statistics to the top level TLV created by `gnet_stats_start_copy`.

Returns 0 on success or -1 with the statistic lock released if the room in the socket buffer was not sufficient.

gnet_stats_copy_app

Name

`gnet_stats_copy_app` — copy application specific statistics into statistics TLV

Synopsis

```
int gnet_stats_copy_app (struct gnet_dump * d, void * st, int  
len);
```

Arguments

d

dumping handle

st

application specific statistics data

len

length of data

Description

Appends the application sepecific statistics to the top level TLV created by `gnet_stats_start_copy` and remembers the data for XSTATS if the dumping handle is in backward compatibility mode.

Returns 0 on success or -1 with the statistic lock released if the room in the socket buffer was not sufficient.

gnet_stats_finish_copy

Name

`gnet_stats_finish_copy` — finish dumping procedure

Synopsis

```
int gnet_stats_finish_copy (struct gnet_dump * d);
```

Arguments

d

dumping handle

Description

Corrects the length of the top level TLV to include all TLVs added by `gnet_stats_copy_XXX` calls. Adds the backward compatibility TLVs if `gnet_stats_start_copy_compat` was used and releases the statistics lock.

Returns 0 on success or -1 with the statistic lock released if the room in the socket buffer was not sufficient.

gen_new_estimator

Name

`gen_new_estimator` — create a new rate estimator

Synopsis

```
int gen_new_estimator (struct gnet_stats_basic * bstats,
struct gnet_stats_rate_est * rate_est, spinlock_t *
stats_lock, struct rtattr * opt);
```

Arguments

bstats

basic statistics

rate_est

rate estimator statistics

stats_lock

statistics lock

opt

rate estimator configuration TLV

Description

Creates a new rate estimator with `&bstats` as source and `&rate_est` as destination. A new timer with the interval specified in the configuration TLV is created. Upon each interval, the latest statistics will be read from `&bstats` and the estimated rate will be stored in `&rate_est` with the statistics lock grabbed during this period.

Returns 0 on success or a negative error code.

gen_kill_estimator

Name

`gen_kill_estimator` — remove a rate estimator

Synopsis

```
void gen_kill_estimator (struct gnet_stats_basic * bstats,  
struct gnet_stats_rate_est * rate_est);
```

Arguments

bstats

basic statistics

rate_est

rate estimator statistics

Description

Removes the rate estimator specified by `&bstats` and `&rate_est` and deletes the timer.

gen_replace_estimator

Name

`gen_replace_estimator` — replace rate estimator configuration

Synopsis

```
int gen_replace_estimator (struct gnet_stats_basic * bstats,  
struct gnet_stats_rate_est * rate_est, spinlock_t *  
stats_lock, struct rtattr * opt);
```

Arguments

bstats

basic statistics

rate_est

rate estimator statistics

stats_lock

statistics lock

opt

rate estimator configuration TLV

Description

Replaces the configuration of a rate estimator by calling `gen_kill_estimator` and `gen_new_estimator`.

Returns 0 on success or a negative error code.

Chapter 9. Network device support

9.1. Driver Support

dev_add_pack

Name

`dev_add_pack` — add packet handler

Synopsis

```
void dev_add_pack (struct packet_type * pt);
```

Arguments

pt

packet type declaration

Description

Add a protocol handler to the networking stack. The passed `&packet_type` is linked into kernel lists and may not be freed until it has been removed from the kernel lists.

This call does not sleep therefore it can not guarantee all CPU's that are in middle of receiving packets will see the new packet type (until the next received packet).

__dev_remove_pack

Name

`__dev_remove_pack` — remove packet handler

Synopsis

```
void __dev_remove_pack (struct packet_type * pt);
```

Arguments

pt

packet type declaration

Description

Remove a protocol handler that was previously added to the kernel protocol handlers by `dev_add_pack`. The passed `&packet_type` is removed from the kernel lists and can be freed or reused once this function returns.

The packet type might still be in use by receivers and must not be freed until after all the CPU's have gone through a quiescent state.

dev_remove_pack

Name

`dev_remove_pack` — remove packet handler

Synopsis

```
void dev_remove_pack (struct packet_type * pt);
```

Arguments

pt

packet type declaration

Description

Remove a protocol handler that was previously added to the kernel protocol handlers by `dev_add_pack`. The passed `&packet_type` is removed from the kernel lists and can be freed or reused once this function returns.

This call sleeps to guarantee that no CPU is looking at the packet type after return.

netdev_boot_setup_check

Name

`netdev_boot_setup_check` — check boot time settings

Synopsis

```
int netdev_boot_setup_check (struct net_device * dev);
```

Arguments

dev

the netdevice

Description

Check boot time settings for the device. The found settings are set for the device to be used later in the device probing. Returns 0 if no settings found, 1 if they are.

__dev_get_by_name

Name

`__dev_get_by_name` — find a device by its name

Synopsis

```
struct net_device * __dev_get_by_name (const char * name);
```

Arguments

name

name to find

Description

Find an interface by name. Must be called under RTNL semaphore or `dev_base_lock`. If the name is found a pointer to the device is returned. If the

name is not found then `NULL` is returned. The reference counters are not incremented so the caller must be careful with locks.

dev_get_by_name

Name

`dev_get_by_name` — find a device by its name

Synopsis

```
struct net_device * dev_get_by_name (const char * name);
```

Arguments

name

name to find

Description

Find an interface by name. This can be called from any context and does its own locking. The returned handle has the usage count incremented and the caller must use `dev_put` to release it when it is no longer needed. `NULL` is returned if no matching device is found.

__dev_get_by_index

Name

`__dev_get_by_index` — find a device by its ifindex

Synopsis

```
struct net_device * __dev_get_by_index (int ifindex);
```

Arguments

ifindex

index of device

Description

Search for an interface by index. Returns `NULL` if the device is not found or a pointer to the device. The device has not had its reference counter increased so the caller must be careful about locking. The caller must hold either the RTNL semaphore or *dev_base_lock*.

dev_get_by_index

Name

`dev_get_by_index` — find a device by its ifindex

Synopsis

```
struct net_device * dev_get_by_index (int ifindex);
```

Arguments

ifindex

index of device

Description

Search for an interface by index. Returns NULL if the device is not found or a pointer to the device. The device returned has had a reference added and the pointer is safe until the user calls `dev_put` to indicate they have finished with it.

`dev_get_by_flags`

Name

`dev_get_by_flags` — find any device with given flags

Synopsis

```
struct net_device * dev_get_by_flags (unsigned short if_flags,  
unsigned short mask);
```

Arguments

if_flags

IFF_* values

mask

bitmask of bits in *if_flags* to check

Description

Search for any interface with the given flags. Returns NULL if a device is not found or a pointer to the device. The device returned has had a reference added and the pointer is safe until the user calls `dev_put` to indicate they have finished with it.

dev_alloc_name

Name

`dev_alloc_name` — allocate a name for a device

Synopsis

```
int dev_alloc_name (struct net_device * dev, const char *  
name);
```

Arguments

dev

device

name

name format string

Description

Passed a format string - eg “%d” it will try and find a suitable id. Not efficient for many devices, not called a lot. The caller must hold the dev_base or rtnl lock while allocating the name and adding the device in order to avoid duplicates. Returns the number of the unit assigned or a negative errno code.

netdev_state_change

Name

netdev_state_change — device changes state

Synopsis

```
void netdev_state_change (struct net_device * dev);
```

Arguments

dev

device to cause notification

Description

Called to indicate a device has changed state. This function calls the notifier chains for netdev_chain and sends a NEWLINK message to the routing socket.

dev_load

Name

`dev_load` — load a network module

Synopsis

```
void dev_load (const char * name);
```

Arguments

name

name of interface

Description

If a network interface is not present and the process has suitable privileges this function loads the module. If module loading is not available in this kernel then it becomes a nop.

dev_open

Name

`dev_open` — prepare an interface for use.

Synopsis

```
int dev_open (struct net_device * dev);
```

Arguments

dev

device to open

Description

Takes a device from down to up state. The device's private open function is invoked and then the multicast lists are loaded. Finally the device is moved into the up state and a `NETDEV_UP` message is sent to the netdev notifier chain.

Calling this function on an active interface is a nop. On a failure a negative errno code is returned.

dev_close

Name

`dev_close` — shutdown an interface.

Synopsis

```
int dev_close (struct net_device * dev);
```

Arguments

dev

device to shutdown

Description

This function moves an active device into down state. A `NETDEV_GOING_DOWN` is sent to the netdev notifier chain. The device is then deactivated and finally a `NETDEV_DOWN` is sent to the notifier chain.

register_netdevice_notifier

Name

`register_netdevice_notifier` — register a network notifier block

Synopsis

```
int register_netdevice_notifier (struct notifier_block * nb);
```

Arguments

nb

notifier

Description

Register a notifier to be called when network device events occur. The notifier passed is linked into the kernel structures and must not be reused until it has been

unregistered. A negative errno code is returned on a failure.

When registered all registration and up events are replayed to the new notifier to allow device to have a race free view of the network device list.

unregister_netdevice_notifier

Name

`unregister_netdevice_notifier` — unregister a network notifier block

Synopsis

```
int unregister_netdevice_notifier (struct notifier_block *  
nb);
```

Arguments

nb

notifier

Description

Unregister a notifier previously registered by `register_netdevice_notifier`. The notifier is unlinked into the kernel structures and may then be reused. A negative errno code is returned on a failure.

dev_queue_xmit

Name

`dev_queue_xmit` — transmit a buffer

Synopsis

```
int dev_queue_xmit (struct sk_buff * skb);
```

Arguments

skb

buffer to transmit

Description

Queue a buffer for transmission to a network device. The caller must have set the device and priority and built the buffer before calling this function. The function can be called from an interrupt.

A negative errno code is returned on a failure. A success does not guarantee the frame will be transmitted as it may be dropped due to congestion or traffic shaping.

netif_rx

Name

`netif_rx` — post buffer to the network code

Synopsis

```
int netif_rx (struct sk_buff * skb);
```

Arguments

skb

buffer to post

Description

This function receives a packet from a device driver and queues it for the upper (protocol) levels to process. It always succeeds. The buffer may be dropped during processing for congestion control or by the protocol layers.

return values

NET_RX_SUCCESS (no congestion) NET_RX_CN_LOW (low congestion)
NET_RX_CN_MOD (moderate congestion) NET_RX_CN_HIGH (high
congestion) NET_RX_DROP (packet was dropped)

register_gifconf

Name

`register_gifconf` — register a SIOCGIF handler

Synopsis

```
int register_gifconf (unsigned int family, gifconf_func_t *  
gifconf);
```

Arguments

family

Address family

gifconf

Function handler

Description

Register protocol dependent address dumping routines. The handler that is passed must not be freed or reused until it has been replaced by another handler.

netdev_set_master

Name

`netdev_set_master` — set up master/slave pair

Synopsis

```
int netdev_set_master (struct net_device * slave, struct  
net_device * master);
```

Arguments

slave

slave device

master

new master device

Description

Changes the master device of the slave. Pass `NULL` to break the bonding. The caller must hold the RTNL semaphore. On a failure a negative `errno` code is returned. On success the reference counts are adjusted, `RTM_NEWLINK` is sent to the routing socket and the function returns zero.

dev_set_promiscuity

Name

`dev_set_promiscuity` — update promiscuity count on a device

Synopsis

```
void dev_set_promiscuity (struct net_device * dev, int inc);
```

Arguments

dev

device

inc

modifier

Description

Add or remove promiscuity from a device. While the count in the device remains above zero the interface remains promiscuous. Once it hits zero the device reverts back to normal filtering operation. A negative *inc* value is used to drop promiscuity on the device.

dev_set_allmulti

Name

`dev_set_allmulti` — update allmulti count on a device

Synopsis

```
void dev_set_allmulti (struct net_device * dev, int inc);
```

Arguments

dev

device

inc

modifier

Description

Add or remove reception of all multicast frames to a device. While the count in the device remains above zero the interface remains listening to all interfaces. Once it hits zero the device reverts back to normal filtering operation. A negative *inc* value is used to drop the counter when releasing a resource needing all multicasts.

dev_ioctl

Name

`dev_ioctl` — network device ioctl

Synopsis

```
int dev_ioctl (unsigned int cmd, void __user * arg);
```

Arguments

cmd

command to issue

arg

pointer to a struct ifreq in user space

Description

Issue ioctl functions to devices. This is normally called by the user space syscall interfaces but can sometimes be useful for other purposes. The return value is the return from the syscall if positive or a negative errno code on error.

register_netdevice

Name

`register_netdevice` — register a network device

Synopsis

```
int register_netdevice (struct net_device * dev);
```

Arguments

dev

device to register

Description

Take a completed network device structure and add it to the kernel interfaces. A `NETDEV_REGISTER` message is sent to the netdev notifier chain. 0 is returned on success. A negative errno code is returned on a failure to set up the device, or if the name is a duplicate.

Callers must hold the `rtnl` semaphore. You may want `register_netdev` instead of this.

BUGS

The locking appears insufficient to guarantee two parallel registers will not get the same name.

register_netdev

Name

`register_netdev` — register a network device

Synopsis

```
int register_netdev (struct net_device * dev);
```

Arguments

dev

device to register

Description

Take a completed network device structure and add it to the kernel interfaces. A `NETDEV_REGISTER` message is sent to the netdev notifier chain. 0 is returned on success. A negative errno code is returned on a failure to set up the device, or if the name is a duplicate.

This is a wrapper around `register_netdev` that takes the rtnl semaphore and expands the device name if you passed a format string to `alloc_netdev`.

alloc_netdev

Name

`alloc_netdev` — allocate network device

Synopsis

```
struct net_device * alloc_netdev (int sizeof_priv, const char  
* name, void (*setup) (struct net_device *));
```

Arguments

sizeof_priv

size of private data to allocate space for

name

device name format string

setup

callback to initialize device

Description

Allocates a struct net_device with private data area for driver use and performs basic initialization.

free_netdev

Name

`free_netdev` — free network device

Synopsis

```
void free_netdev (struct net_device * dev);
```

Arguments

dev

device

Description

This function does the last stage of destroying an allocated device interface. The reference to the device object is released. If this is the last reference then it will be freed.

unregister_netdevice

Name

`unregister_netdevice` — remove device from the kernel

Synopsis

```
int unregister_netdevice (struct net_device * dev);
```

Arguments

dev

device

Description

This function shuts down a device interface and removes it from the kernel tables. On success 0 is returned, on a failure a negative errno code is returned.

Callers must hold the rtnl semaphore. You may want `unregister_netdev` instead of this.

unregister_netdev

Name

`unregister_netdev` — remove device from the kernel

Synopsis

```
void unregister_netdev (struct net_device * dev);
```

Arguments

dev

device

Description

This function shuts down a device interface and removes it from the kernel tables. On success 0 is returned, on a failure a negative errno code is returned.

This is just a wrapper for `unregister_netdevice` that takes the rtnl semaphore. In general you want to use this and not `unregister_netdevice`.

9.2. 8390 Based Network Cards

ei_open

Name

`ei_open` — Open/initialize the board.

Synopsis

```
int ei_open (struct net_device * dev);
```

Arguments

dev

network device to initialize

Description

This routine goes all-out, setting everything up anew at each open, even though many of these registers should only need to be set once at boot.

ei_close

Name

`ei_close` — shut down network device

Synopsis

```
int ei_close (struct net_device * dev);
```

Arguments

dev

network device to close

Description

Opposite of `ei_open`. Only used when “ifconfig <devname> down” is done.

ei_interrupt

Name

`ei_interrupt` — handle the interrupts from an 8390

Synopsis

```
irqreturn_t ei_interrupt (int irq, void * dev_id, struct  
pt_regs * regs);
```


Arguments

irq

interrupt number

dev_id

a pointer to the `net_device`

regs

unused

Description

Handle the ether interface interrupts. We pull packets from the 8390 via the card specific functions and fire them at the networking stack. We also handle transmit completions and wake the transmit path if necessary. We also update the counters and do other housekeeping as needed.

__alloc_ei_netdev

Name

`__alloc_ei_netdev` — `alloc_etherdev` counterpart for 8390

Synopsis

```
struct net_device * __alloc_ei_netdev (int size);
```

Arguments

size

-- undescribed --

Description

Allocate 8390-specific net_device.

NS8390_init

Name

NS8390_init — initialize 8390 hardware

Synopsis

```
void NS8390_init (struct net_device * dev, int startp);
```

Arguments

dev

network device to initialize

startp

boolean. non-zero value to initiate chip processing

Description

Must be called with lock held.

9.3. Synchronous PPP

sppp_input

Name

sppp_input — receive and process a WAN PPP frame

Synopsis

```
void sppp_input (struct net_device * dev, struct sk_buff *  
skb);
```

Arguments

dev

The device it arrived on

skb

The buffer to process

Description

This can be called directly by cards that do not have timing constraints but is normally called from the network layer after interrupt servicing to process frames queued via `netif_rx`.

We process the options in the card. If the frame is destined for the protocol stacks then it requeues the frame for the upper level protocol. If it is a control from it is processed and discarded here.

sppp_close

Name

`sppp_close` — close down a synchronous PPP or Cisco HDLC link

Synopsis

```
int sppp_close (struct net_device * dev);
```

Arguments

dev

The network device to drop the link of

Description

This drops the logical interface to the channel. It is not done politely as we assume we will also be dropping DTR. Any timeouts are killed.

sppp_open

Name

sppp_open — open a synchronous PPP or Cisco HDLC link

Synopsis

```
int sppp_open (struct net_device * dev);
```

Arguments

dev

Network device to activate

Description

Close down any existing synchronous session and commence from scratch. In the PPP case this means negotiating LCP/IPCP and friends, while for Cisco HDLC we simply need to start sending keepalives

sppp_reopen

Name

sppp_reopen — notify of physical link loss

Synopsis

```
int sppp_reopen (struct net_device * dev);
```

Arguments

dev

Device that lost the link

Description

This function informs the synchronous protocol code that the underlying link died (for example a carrier drop on X.21)

We increment the magic numbers to ensure that if the other end failed to notice we will correctly start a new session. It happens do to the nature of telco circuits is that you can lose carrier on one endonly.

Having done this we go back to negotiating. This function may be called from an interrupt context.

sppp_change_mtu

Name

`sppp_change_mtu` — Change the link MTU

Synopsis

```
int sppp_change_mtu (struct net_device * dev, int new_mtu);
```

Arguments

dev

Device to change MTU on

new_mtu

New MTU

Description

Change the MTU on the link. This can only be called with the link down. It returns an error if the link is up or the mtu is out of range.

sppp_do_ioctl

Name

sppp_do_ioctl — Ioctl handler for ppp/hdlc

Synopsis

```
int sppp_do_ioctl (struct net_device * dev, struct ifreq *
ifr, int cmd);
```

Arguments

dev

Device subject to ioctl

ifr

Interface request block from the user

cmd

Command that is being issued

Description

This function handles the ioctls that may be issued by the user to control the settings of a PPP/HDLC link. It does both busy and security checks. This function is intended to be wrapped by callers who wish to add additional ioctl calls of their own.

sppp_attach

Name

`sppp_attach` — attach synchronous PPP/HDLC to a device

Synopsis

```
void sppp_attach (struct ppp_device * pd);
```

Arguments

pd

PPP device to initialise

Description

This initialises the PPP/HDLC support on an interface. At the time of calling the `dev` element must point to the network device that this interface is attached to. The interface should not yet be registered.

sppp_detach

Name

sppp_detach — release PPP resources from a device

Synopsis

```
void sppp_detach (struct net_device * dev);
```

Arguments

dev

Network device to release

Description

Stop and free up any PPP/HDLC resources used by this interface. This must be called before the device is freed.

Chapter 10. Module Support

10.1. Module Loading

request_module

Name

`request_module` — try to load a kernel module

Synopsis

```
int request_module (const char * fmt, ...);
```

Arguments

fmt

printf style format string for the name of the module

...

variable arguments

Description

Load a module using the user mode module loader. The function returns zero on success or a negative `errno` code on failure. Note that a successful module load does not mean the module did not then unload and exit on an error of its own. Callers must check that the service they requested is now available not blindly invoke it.

If module auto-loading support is disabled then this function becomes a no-operation.

call_usermodehelper

Name

`call_usermodehelper` — start a usermode application

Synopsis

```
int call_usermodehelper (char * path, char ** argv, char **  
envp, int wait);
```

Arguments

path

pathname for the application

argv

null-terminated argument list

envp

null-terminated environment list

wait

wait for the application to finish and return status.

Description

Runs a user-space application. The application is started asynchronously if `wait` is not set, and runs as a child of `keventd`. (ie. it runs with full root capabilities).

Must be called from process context. Returns a negative error code if program was not execed successfully, or 0.

10.2. Inter Module support

Refer to the file `kernel/module.c` for more information.

Chapter 11. Hardware Interfaces

11.1. Interrupt Handling

/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/irq.c

Name

/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/irq.c
— Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file

/usr/src/packages/BUILD/linux-2.6.11.4-20a/arch/i386/kernel/irq.c
at this point, but none was found. This dummy section is inserted to allow generation to continue.

11.2. MTRR Handling

mtrr_add

Name

`mtrr_add` — Add a memory type region

Synopsis

```
int mtrr_add (unsigned long base, unsigned long size, unsigned  
int type, char increment);
```

Arguments

base

Physical base address of region

size

Physical size of region

type

Type of MTRR desired

increment

If this is true do usage counting on the region

Description

Memory type region registers control the caching on newer Intel and non Intel processors. This function allows drivers to request an MTRR is added. The details and hardware specifics of each processor's implementation are hidden from the

caller, but nevertheless the caller should expect to need to provide a power of two size on an equivalent power of two boundary.

If the region cannot be added either because all regions are in use or the CPU cannot support it a negative value is returned. On success the register number for this entry is returned, but should be treated as a cookie only.

On a multiprocessor machine the changes are made to all processors. This is required on x86 by the Intel processors.

The available types are

`MTRR_TYPE_UNCACHABLE` - No caching

`MTRR_TYPE_WRBACK` - Write data back in bursts whenever

`MTRR_TYPE_WRCOMB` - Write data back soon but allow bursts

`MTRR_TYPE_WRTHROUGH` - Cache reads but not writes

BUGS

Needs a quiet flag for the cases where drivers do not mind failures and do not wish system log messages to be sent.

mtrr_del

Name

`mtrr_del` — delete a memory type region

Synopsis

```
int mtrr_del (int reg, unsigned long base, unsigned long size);
```

Arguments

reg

Register returned by `mtrr_add`

base

Physical base address

size

Size of region

Description

If register is supplied then base and size are ignored. This is how drivers should call it.

Releases an MTRR region. If the usage count drops to zero the register is freed and the region returns to default state. On success the register is returned, on failure a negative error code.

11.3. PCI Support Library

`pci_bus_max_busnr`

Name

`pci_bus_max_busnr` — returns maximum PCI bus number of given bus' children

Synopsis

```
unsigned char __devinit pci_bus_max_busnr (struct pci_bus *  
bus);
```

Arguments

bus

pointer to PCI bus structure to search

Description

Given a PCI bus, returns the highest PCI bus number present in the set including the given PCI bus and its list of child PCI buses.

pci_max_busnr

Name

`pci_max_busnr` — returns maximum PCI bus number

Synopsis

```
unsigned char __devinit pci_max_busnr ( void );
```

Arguments

void

no arguments

Description

Returns the highest PCI bus number present in the system global list of PCI buses.

pci_find_capability

Name

`pci_find_capability` — query for devices' capabilities

Synopsis

```
int pci_find_capability (struct pci_dev * dev, int cap);
```

Arguments

dev

PCI device to query

cap

capability code

Description

Tell if a device supports a given PCI capability. Returns the address of the requested capability structure within the device's PCI configuration space or 0 in case the device does not support it. Possible values for *cap*:

`PCI_CAP_ID_PM` Power Management `PCI_CAP_ID_AGP` Accelerated Graphics
Port `PCI_CAP_ID_VPD` Vital Product Data `PCI_CAP_ID_SLOTID` Slot
Identification `PCI_CAP_ID_MSI` Message Signalled Interrupts

PCI_CAP_ID_CHSWP CompactPCI HotSwap PCI_CAP_ID_PCIX PCI-X
 PCI_CAP_ID_EXP PCI Express

pci_bus_find_capability

Name

`pci_bus_find_capability` — query for devices' capabilities

Synopsis

```
int pci_bus_find_capability (struct pci_bus * bus, unsigned
int devfn, int cap);
```

Arguments

bus

the PCI bus to query

devfn

PCI device to query

cap

capability code

Description

Like `pci_find_capability` but works for pci devices that do not have a `pci_dev` structure set up yet.

Returns the address of the requested capability structure within the device's PCI configuration space or 0 in case the device does not support it.

pci_find_parent_resource

Name

`pci_find_parent_resource` — return resource region of parent bus of given region

Synopsis

```
struct resource * pci_find_parent_resource (const struct
pci_dev * dev, struct resource * res);
```

Arguments

dev

PCI device structure contains resources to be searched

res

child resource record for which parent is sought

Description

For given resource region of given device, return the resource region of parent bus the given region is contained in or where it should be allocated from.

pci_set_power_state

Name

`pci_set_power_state` — Set the power state of a PCI device

Synopsis

```
int pci_set_power_state (struct pci_dev * dev, pci_power_t  
state);
```

Arguments

dev

PCI device to be suspended

state

PCI power state (D0, D1, D2, D3hot, D3cold) we're entering

Description

Transition a device to a new power state, using the Power Management Capabilities in the device's config space.

RETURN VALUE

-EINVAL if trying to enter a lower state than we're already in. 0 if we're already in the requested state. -EIO if device does not support PCI PM. 0 if we can successfully change the power state.

pci_choose_state

Name

`pci_choose_state` — Choose the power state of a PCI device

Synopsis

```
pci_power_t pci_choose_state (struct pci_dev * dev, u32  
state);
```

Arguments

dev

PCI device to be suspended

state

target sleep state for the whole system

Description

Returns PCI power state suitable for given device and given system message.

pci_save_state

Name

`pci_save_state` — save the PCI configuration space of a device before suspending

Synopsis

```
int pci_save_state (struct pci_dev * dev);
```

Arguments

dev

- PCI device that we're dealing with

Description

buffer must be large enough to hold the entire PCI 2.2 config space (≥ 64 bytes).

pci_restore_state

Name

`pci_restore_state` — Restore the saved state of a PCI device

Synopsis

```
int pci_restore_state (struct pci_dev * dev);
```

Arguments

dev

- PCI device that we're dealing with

pci_enable_device_bars

Name

`pci_enable_device_bars` — Initialize some of a device for use

Synopsis

```
int pci_enable_device_bars (struct pci_dev * dev, int bars);
```

Arguments

dev

PCI device to be initialized

bars

bitmask of BAR's that must be configured

Description

Initialize device before it's used by a driver. Ask low-level code to enable selected I/O and memory resources. Wake up the device if it was suspended. Beware, this function can fail.

pci_enable_device

Name

`pci_enable_device` — Initialize device before it's used by a driver.

Synopsis

```
int pci_enable_device (struct pci_dev * dev);
```

Arguments

dev

PCI device to be initialized

Description

Initialize device before it's used by a driver. Ask low-level code to enable I/O and memory. Wake up the device if it was suspended. Beware, this function can fail.

pci_disable_device

Name

`pci_disable_device` — Disable PCI device after use

Synopsis

```
void pci_disable_device (struct pci_dev * dev);
```

Arguments

dev

PCI device to be disabled

Description

Signal to the system that the PCI device is not in use by the system anymore. This only involves disabling PCI bus-mastering, if active.

pci_enable_wake

Name

`pci_enable_wake` — enable device to generate PME# when suspended

Synopsis

```
int pci_enable_wake (struct pci_dev * dev, pci_power_t state,  
int enable);
```

Arguments

dev

- PCI device to operate on

state

- Current state of device.

enable

- Flag to enable or disable generation

Description

Set the bits in the device's PM Capabilities to generate PME# when the system is suspended.

-EIO is returned if device doesn't have PM Capabilities. -EINVAL is returned if device supports it, but can't generate wake events. 0 if operation is successful.

pci_release_region

Name

`pci_release_region` — Release a PCI bar

Synopsis

```
void pci_release_region (struct pci_dev * pdev, int bar);
```

Arguments

pdev

PCI device whose resources were previously reserved by `pci_request_region`

bar

BAR to release

Description

Releases the PCI I/O and memory resources previously reserved by a successful call to `pci_request_region`. Call this function only after all use of the PCI regions has ceased.

pci_request_region

Name

`pci_request_region` — Reserved PCI I/O and memory resource

Synopsis

```
int pci_request_region (struct pci_dev * pdev, int bar, char *  
res_name);
```

Arguments

pdev

PCI device whose resources are to be reserved

bar

BAR to be reserved

res_name

Name to be associated with resource.

Description

Mark the PCI region associated with PCI device *pdev* BR *bar* as being reserved by owner *res_name*. Do not access any address inside the PCI regions unless this call returns successfully.

Returns 0 on success, or `EBUSY` on error. A warning message is also printed on failure.

pci_release_regions

Name

`pci_release_regions` — Release reserved PCI I/O and memory resources

Synopsis

```
void pci_release_regions (struct pci_dev * pdev);
```

Arguments

pdev

PCI device whose resources were previously reserved by `pci_request_regions`

Description

Releases all PCI I/O and memory resources previously reserved by a successful call to `pci_request_regions`. Call this function only after all use of the PCI regions has ceased.

pci_request_regions

Name

`pci_request_regions` — Reserved PCI I/O and memory resources

Synopsis

```
int pci_request_regions (struct pci_dev * pdev, char *  
res_name);
```

Arguments

pdev

PCI device whose resources are to be reserved

res_name

Name to be associated with resource.

Description

Mark all PCI regions associated with PCI device *pdev* as being reserved by owner *res_name*. Do not access any address inside the PCI regions unless this call returns successfully.

Returns 0 on success, or `EBUSY` on error. A warning message is also printed on failure.

pci_set_master

Name

`pci_set_master` — enables bus-mastering for device `dev`

Synopsis

```
void pci_set_master (struct pci_dev * dev);
```

Arguments

dev

the PCI device to enable

Description

Enables bus-mastering on the device and calls `pcibios_set_master` to do the needed arch specific settings.

pci_set_mwi

Name

`pci_set_mwi` — enables memory-write-invalidate PCI transaction

Synopsis

```
int pci_set_mwi (struct pci_dev * dev);
```

Arguments

dev

the PCI device for which MWI is enabled

Description

Enables the Memory-Write-Invalidate transaction in `PCI_COMMAND`, and then calls `pcibios_set_mwi` to do the needed arch specific operations or a generic mwi-prep function.

RETURNS

An appropriate -ERRNO error value on error, or zero for success.

pci_clear_mwi

Name

`pci_clear_mwi` — disables Memory-Write-Invalidate for device *dev*

Synopsis

```
void pci_clear_mwi (struct pci_dev * dev);
```

Arguments

dev

the PCI device to disable

Description

Disables PCI Memory-Write-Invalidate transaction on the device

11.4. PCI Hotplug Support Library

pci_hp_register

Name

`pci_hp_register` — register a `hotplug_slot` with the PCI hotplug subsystem

Synopsis

```
int pci_hp_register (struct hotplug_slot * slot);
```

Arguments

slot

pointer to the `&struct hotplug_slot` to register

Description

Registers a hotplug slot with the pci hotplug subsystem, which will allow userspace interaction to the slot.

Returns 0 if successful, anything else for an error.

pci_hp_deregister

Name

`pci_hp_deregister` — deregister a `hotplug_slot` with the PCI hotplug subsystem

Synopsis

```
int pci_hp_deregister (struct hotplug_slot * slot);
```

Arguments

slot

pointer to the `&struct hotplug_slot` to deregister

Description

The *slot* must have been registered with the pci hotplug subsystem previously with a call to `pci_hp_register`.

Returns 0 if successful, anything else for an error.

pci_hp_change_slot_info

Name

`pci_hp_change_slot_info` — changes the slot's information structure in the core

Synopsis

```
int pci_hp_change_slot_info (struct hotplug_slot * slot,  
struct hotplug_slot_info * info);
```

Arguments

slot

pointer to the slot whose info has changed

info

pointer to the info copy into the slot's info structure

Description

slot must have been registered with the pci hotplug subsystem previously with a call to `pci_hp_register`.

Returns 0 if successful, anything else for an error.

11.5. MCA Architecture

11.5.1. MCA Device Functions

Refer to the file `arch/i386/kernel/mca.c` for more information.

11.5.2. MCA Bus DMA

`mca_enable_dma`

Name

`mca_enable_dma` — channel to enable DMA on

Synopsis

```
void mca_enable_dma (unsigned int dmanr);
```

Arguments

dmanr

DMA channel

Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.

mca_disable_dma

Name

`mca_disable_dma` — channel to disable DMA on

Synopsis

```
void mca_disable_dma (unsigned int dmanr);
```

Arguments

dmanr

DMA channel

Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.

mca_set_dma_addr

Name

`mca_set_dma_addr` — load a 24bit DMA address

Synopsis

```
void mca_set_dma_addr (unsigned int dmanr, unsigned int a);
```

Arguments

dmanr

DMA channel

a

24bit bus address

Description

Load the address register in the DMA controller. This has a 24bit limitation (16Mb).

mca_get_dma_addr

Name

`mca_get_dma_addr` — load a 24bit DMA address

Synopsis

```
unsigned int mca_get_dma_addr (unsigned int dmanr);
```

Arguments

dmanr

DMA channel

Description

Read the address register in the DMA controller. This has a 24bit limitation (16Mb). The return is a bus address.

mca_set_dma_count

Name

`mca_set_dma_count` — load a 16bit transfer count

Synopsis

```
void mca_set_dma_count (unsigned int dmanr, unsigned int
count);
```

Arguments

dmanr

DMA channel

count

count

Description

Set the DMA count for this channel. This can be up to 64Kbytes. Setting a count of zero will not do what you expect.

mca_get_dma_residue

Name

`mca_get_dma_residue` — get the remaining bytes to transfer

Synopsis

```
unsigned int mca_get_dma_residue (unsigned int dmanr);
```

Arguments

dmanr

DMA channel

Description

This function returns the number of bytes left to transfer on this DMA channel.

mca_set_dma_io

Name

`mca_set_dma_io` — set the port for an I/O transfer

Synopsis

```
void mca_set_dma_io (unsigned int dmanr, unsigned int  
io_addr);
```

Arguments

dmanr

DMA channel

io_addr

an I/O port number

Description

Unlike the ISA bus DMA controllers the DMA on MCA bus can transfer with an I/O port target.

mca_set_dma_mode

Name

`mca_set_dma_mode` — set the DMA mode

Synopsis

```
void mca_set_dma_mode (unsigned int dmanr, unsigned int mode);
```

Arguments

dmanr

DMA channel

mode

mode to set

Description

The DMA controller supports several modes. The mode values you can

set are

`MCA_DMA_MODE_READ` when reading from the DMA device.

`MCA_DMA_MODE_WRITE` to writing to the DMA device.

`MCA_DMA_MODE_IO` to do DMA to or from an I/O port.

`MCA_DMA_MODE_16` to do 16bit transfers.

Chapter 12. The Device File System

devfs_mk_dir

Name

`devfs_mk_dir` — Create a directory in the devfs namespace.

Synopsis

```
int devfs_mk_dir (const char * fmt,  ...);
```

Arguments

fmt

The name of the entry.

...

variable arguments

Description

Use of this function is optional. The `devfs_register` function will automatically create intermediate directories as needed. This function is provided for efficiency reasons, as it provides a handle to a directory. On failure `NULL` is returned.

Description

Use of this function is optional. The `devfs_register` function will automatically create intermediate directories as needed. This function is provided for efficiency reasons, as it provides a handle to a directory. On failure `NULL` is returned.

Chapter 13. Security Framework

register_security

Name

`register_security` — registers a security framework with the kernel

Synopsis

```
int register_security (struct security_operations * ops);
```

Arguments

ops

a pointer to the struct `security_options` that is to be registered

Description

This function is to allow a security module to register itself with the kernel security subsystem. Some rudimentary checking is done on the *ops* value passed to this function. A call to `unregister_security` should be done to remove this `security_options` structure from the kernel.

If there is already a security module registered with the kernel, an error will be returned. Otherwise 0 is returned on success.

unregister_security

Name

`unregister_security` — unregisters a security framework with the kernel

Synopsis

```
int unregister_security (struct security_operations * ops);
```

Arguments

ops

a pointer to the struct `security_options` that is to be registered

Description

This function removes a struct `security_operations` variable that had previously been registered with a successful call to `register_security`.

If *ops* does not match the valued previously passed to `register_security` an error is returned. Otherwise the default security options is set to the the `capability_security_ops` structure, and 0 is returned.

mod_reg_security

Name

`mod_reg_security` — allows security modules to be “stacked”

Synopsis

```
int mod_reg_security (const char * name, struct
security_operations * ops);
```

Arguments

name

a pointer to a string with the name of the security_options to be registered

ops

a pointer to the struct security_options that is to be registered

Description

This function allows security modules to be stacked if the currently loaded security module allows this to happen. It passes the *name* and *ops* to the register_security function of the currently loaded security module.

The return value depends on the currently loaded security module, with 0 as success.

mod_unreg_security

Name

mod_unreg_security — allows a security module registered with mod_reg_security to be unloaded

Synopsis

```
int mod_unreg_security (const char * name, struct  
security_operations * ops);
```

Arguments

name

a pointer to a string with the name of the security_options to be removed

ops

a pointer to the struct security_options that is to be removed

Description

This function allows security modules that have been successfully registered with a call to `mod_reg_security` to be unloaded from the system. This calls the currently loaded security module's `unregister_security` call with the *name* and *ops* variables.

The return value depends on the currently loaded security module, with 0 as success.

capable

Name

`capable` — calls the currently loaded security module's `capable` function with the specified capability

Synopsis

```
int capable (int cap);
```

Arguments

cap

the requested capability level.

Description

This function calls the currently loaded security module's `capable` function with a pointer to the current task and the specified *cap* value.

This allows the security module to implement the `capable` function call however it chooses to.

Chapter 14. Power Management

pm_register

Name

`pm_register` — register a device with power management

Synopsis

```
struct pm_dev * pm_register (pm_dev_t type, unsigned long id,  
pm_callback callback);
```

Arguments

type

device type

id

device ID

callback

callback function

Description

Add a device to the list of devices that wish to be notified about power management events. A `&pm_dev` structure is returned on success, on failure the return is `NULL`.

The callback function will be called in process context and it may sleep.

pm_unregister

Name

`pm_unregister` — unregister a device with power management

Synopsis

```
void pm_unregister (struct pm_dev * dev);
```

Arguments

dev

device to unregister

Description

Remove a device from the power management notification lists. The `dev` passed must be a handle previously returned by `pm_register`.

pm_unregister_all

Name

`pm_unregister_all` — unregister all devices with matching callback

Synopsis

```
void pm_unregister_all (pm_callback callback);
```

Arguments

callback

callback function pointer

Description

Unregister every device that would call the callback passed. This is primarily meant as a helper function for loadable modules. It enables a module to give up all its managed devices without keeping its own private list.

pm_send_all

Name

`pm_send_all` — send request to all managed devices

Synopsis

```
int pm_send_all (pm_request_t rqst, void * data);
```

Arguments

rqst

power management request

data

data for the callback

Description

Issue a power management request to all devices. The `PM_SUSPEND` events are handled specially. Any device is permitted to fail a suspend by returning a non zero (error) value from its callback function. If any device vetoes a suspend request then all other devices that have suspended during the processing of this request are restored to their previous state.

WARNING

This function takes the `pm_devs_lock`. The lock is not dropped until the callbacks have completed. This prevents races against pm locking functions, races against module unload `pm_unregister` code. It does mean however that you must not issue `pm_` functions within the callback or you will deadlock and users will hate you.

Zero is returned on success. If a suspend fails then the status from the device that vetoes the suspend is returned.

BUGS

what stops two power management requests occurring in parallel and conflicting.

Chapter 15. Block Devices

blk_get_backing_dev_info

Name

`blk_get_backing_dev_info` — get the address of a queue's backing_dev_info

Synopsis

```
struct backing_dev_info * blk_get_backing_dev_info (struct  
block_device * bdev);
```

Arguments

bdev
device

Description

Locates the passed device's request queue and returns the address of its backing_dev_info

Will return NULL if the request queue cannot be located.

blk_queue_prep_rq

Name

`blk_queue_prep_rq` — set a `prepare_request` function for queue

Synopsis

```
void blk_queue_prep_rq (request_queue_t * q, prep_rq_fn *  
pfn);
```

Arguments

q

queue

pfn

`prepare_request` function

Description

It's possible for a queue to register a `prepare_request` callback which is invoked before the request is handed to the `request_fn`. The goal of the function is to prepare a request for I/O, it can be used to build a `cdb` from the request data for instance.

blk_queue_merge_bvec

Name

`blk_queue_merge_bvec` — set a `merge_bvec` function for queue

Synopsis

```
void blk_queue_merge_bvec (request_queue_t * q, merge_bvec_fn
* mbfn);
```

Arguments

q

queue

mbfn

merge_bvec_fn

Description

Usually queues have static limitations on the max sectors or segments that we can put in a request. Stacking drivers may have some settings that are dynamic, and thus we have to query the queue whether it is ok to add a new bio_vec to a bio at a given offset or not. If the block device has such limitations, it needs to register a merge_bvec_fn to control the size of bio's sent to it. Note that a block device *must* allow a single page to be added to an empty bio. The block device driver may want to use the bio_split function to deal with these bio's. By default no merge_bvec_fn is defined for a queue, and only the fixed limits are honored.

blk_queue_make_request

Name

blk_queue_make_request — define an alternate make_request function for a device

Synopsis

```
void blk_queue_make_request (request_queue_t * q,  
make_request_fn * mfn);
```

Arguments

q

the request queue for the device to be affected

mfn

the alternate make_request function

Description

The normal way for `&struct bios` to be passed to a device driver is for them to be collected into requests on a request queue, and then to allow the device driver to select requests off that queue when it is ready. This works well for many block devices. However some block devices (typically virtual devices such as `md` or `lvm`) do not benefit from the processing on the request queue, and are served best by having the requests passed directly to them. This can be achieved by providing a function to `blk_queue_make_request`.

Caveat

The driver that does this *must* be able to deal appropriately with buffers in “highmemory”. This can be accomplished by either calling `__bio_kmap_atomic` to get a temporary kernel mapping, or by calling `blk_queue_bounce` to create a buffer in normal memory.

blk_queue_ordered

Name

`blk_queue_ordered` — does this queue support ordered writes

Synopsis

```
void blk_queue_ordered (request_queue_t * q, int flag);
```

Arguments

q

the request queue

flag

see below

Description

For journalled file systems, doing ordered writes on a commit block instead of explicitly doing `wait_on_buffer` (which is bad for performance) can be a big win. Block drivers supporting this feature should call this function and indicate so.

blk_queue_issue_flush_fn

Name

`blk_queue_issue_flush_fn` — set function for issuing a flush

Synopsis

```
void blk_queue_issue_flush_fn (request_queue_t * q,  
issue_flush_fn * iff);
```

Arguments

q

the request queue

iff

the function to be called issuing the flush

Description

If a driver supports issuing a flush command, the support is notified to the block layer by defining it through this call.

blk_complete_barrier_rq

Name

blk_complete_barrier_rq — complete possible barrier request

Synopsis

```
int blk_complete_barrier_rq (request_queue_t * q, struct  
request * rq, int sectors);
```

Arguments

<i>q</i>	the request queue for the device
<i>rq</i>	the request
<i>sectors</i>	number of sectors to complete

Description

Used in driver `end_io` handling to determine whether to postpone completion of a barrier request until a post flush has been done. This is the unlocked variant, used if the caller doesn't already hold the queue lock.

blk_complete_barrier_rq_locked

Name

`blk_complete_barrier_rq_locked` — complete possible barrier request

Synopsis

```
int blk_complete_barrier_rq_locked (request_queue_t * q,
struct request * rq, int sectors);
```

Arguments

<i>q</i>	the request queue for the device
<i>rq</i>	the request
<i>sectors</i>	number of sectors to complete

Description

See `blk_complete_barrier_rq`. This variant must be used if the caller holds the queue lock.

blk_queue_bounce_limit

Name

`blk_queue_bounce_limit` — set bounce buffer limit for queue

Synopsis

```
void blk_queue_bounce_limit (request_queue_t * q, u64  
dma_addr);
```

Arguments

<i>q</i>	the request queue for the device
----------	----------------------------------

dma_addr

bus address limit

Description

Different hardware can have different requirements as to what pages it can do I/O directly to. A low level driver can call `blk_queue_bounce_limit` to have lower memory pages allocated as bounce buffers for doing I/O to pages residing above *page*. By default the block layer sets this to the highest numbered “low” memory page.

blk_queue_max_sectors

Name

`blk_queue_max_sectors` — set max sectors for a request for this queue

Synopsis

```
void blk_queue_max_sectors (request_queue_t * q, unsigned
short max_sectors);
```

Arguments

q

the request queue for the device

max_sectors

max sectors in the usual 512b unit

Description

Enables a low level driver to set an upper limit on the size of received requests.

blk_queue_max_phys_segments

Name

`blk_queue_max_phys_segments` — set max phys segments for a request for this queue

Synopsis

```
void blk_queue_max_phys_segments (request_queue_t * q,  
unsigned short max_segments);
```

Arguments

q

the request queue for the device

max_segments

max number of segments

Description

Enables a low level driver to set an upper limit on the number of physical data segments in a request. This would be the largest sized scatter list the driver could handle.

blk_queue_max_hw_segments

Name

`blk_queue_max_hw_segments` — set max hw segments for a request for this queue

Synopsis

```
void blk_queue_max_hw_segments (request_queue_t * q, unsigned short max_segments);
```

Arguments

q

the request queue for the device

max_segments

max number of segments

Description

Enables a low level driver to set an upper limit on the number of hw data segments in a request. This would be the largest number of address/length pairs the host adapter can actually give as once to the device.

blk_queue_max_segment_size

Name

`blk_queue_max_segment_size` — set max segment size for `blk_rq_map_sg`

Synopsis

```
void blk_queue_max_segment_size (request_queue_t * q, unsigned  
int max_size);
```

Arguments

q

the request queue for the device

max_size

max size of segment in bytes

Description

Enables a low level driver to set an upper limit on the size of a coalesced segment

blk_queue_hardsect_size

Name

`blk_queue_hardsect_size` — set hardware sector size for the queue

Synopsis

```
void blk_queue_hardsect_size (request_queue_t * q, unsigned
short size);
```

Arguments

q

the request queue for the device

size

the hardware sector size, in bytes

Description

This should typically be set to the lowest possible sector size that the hardware can operate on (possible without reverting to even internal read-modify-write operations). Usually the default of 512 covers most hardware.

blk_queue_stack_limits

Name

`blk_queue_stack_limits` — inherit underlying queue limits for stacked drivers

Synopsis

```
void blk_queue_stack_limits (request_queue_t * t,
request_queue_t * b);
```

Arguments

t

the stacking driver (top)

b

the underlying device (bottom)

blk_queue_segment_boundary

Name

`blk_queue_segment_boundary` — set boundary rules for segment merging

Synopsis

```
void blk_queue_segment_boundary (request_queue_t * q, unsigned  
long mask);
```

Arguments

q

the request queue for the device

mask

the memory boundary mask

blk_queue_dma_alignment

Name

`blk_queue_dma_alignment` — set dma length and memory alignment

Synopsis

```
void blk_queue_dma_alignment (request_queue_t * q, int mask);
```

Arguments

q

the request queue for the device

mask

alignment mask

description

set required memory and length alignment for direct dma transactions. this is used when building direct io requests for the queue.

blk_queue_find_tag

Name

`blk_queue_find_tag` — find a request by its tag and queue

Synopsis

```
struct request * blk_queue_find_tag (request_queue_t * q, int
tag);
```

Arguments

q

The request queue for the device

tag

The tag of the request

Description

Notes

Should be used when a device returns a tag and you want to match it with a request.
no locks need be held.

blk_queue_free_tags

Name

blk_queue_free_tags — release tag maintenance info

Synopsis

```
void blk_queue_free_tags (request_queue_t * q);
```

Arguments

q

the request queue for the device

Notes

This is used to disabled tagged queuing to a device, yet leave queue in function.

blk_queue_init_tags

Name

blk_queue_init_tags — initialize the queue tag info

Synopsis

```
int blk_queue_init_tags (request_queue_t * q, int depth,  
struct blk_queue_tag * tags);
```

Arguments

q
the request queue for the device

depth
the maximum queue depth supported

tags
-- undescribed --

blk_queue_resize_tags

Name

`blk_queue_resize_tags` — change the queueing depth

Synopsis

```
int blk_queue_resize_tags (request_queue_t * q, int  
new_depth);
```

Arguments

q
the request queue for the device

new_depth
the new max command queueing depth

Notes

Must be called with the queue lock held.

blk_queue_end_tag

Name

`blk_queue_end_tag` — end tag operations for a request

Synopsis

```
void blk_queue_end_tag (request_queue_t * q, struct request *  
rq);
```

Arguments

q
the request queue for the device

rq
the request that has completed

Description

Typically called when `end_that_request_first` returns 0, meaning all transfers have been done for a request. It's important to call this function before `end_that_request_last`, as that will put the request back on the free list thus corrupting the internal tag list.

Notes

queue lock must be held.

blk_queue_start_tag

Name

`blk_queue_start_tag` — find a free tag and assign it

Synopsis

```
int blk_queue_start_tag (request_queue_t * q, struct request *  
rq);
```

Arguments

q

the request queue for the device

rq

the block request that needs tagging

Description

This can either be used as a stand-alone helper, or possibly be assigned as the queue `&prep_rq_fn` (in which case `&struct request` automatically gets a tag assigned).

Note that this function assumes that any type of request can be queued! if this is not true for your device, you must check the request type before calling this function.

The request will also be removed from the request queue, so it's the drivers responsibility to read it if it should need to be restarted for some reason.

Notes

queue lock must be held.

blk_queue_invalidate_tags

Name

`blk_queue_invalidate_tags` — invalidate all pending tags

Synopsis

```
void blk_queue_invalidate_tags (request_queue_t * q);
```

Arguments

q

the request queue for the device

Description

Hardware conditions may dictate a need to stop all pending requests. In this case, we will safely clear the block side of the tag queue and readd all requests to the request queue in the right order.

Notes

queue lock must be held.

generic_unplug_device

Name

`generic_unplug_device` — fire a request queue

Synopsis

```
void generic_unplug_device (request_queue_t * q);
```

Arguments

q

The `&request_queue_t` in question

Description

Linux uses plugging to build bigger requests queues before letting the device have at them. If a queue is plugged, the I/O scheduler is still adding and merging requests on the queue. Once the queue gets unplugged, the `request_fn` defined for the queue is invoked and transfers started.

blk_start_queue

Name

`blk_start_queue` — restart a previously stopped queue

Synopsis

```
void blk_start_queue (request_queue_t * q);
```

Arguments

q

The &request_queue_t in question

Description

`blk_start_queue` will clear the stop flag on the queue, and call the `request_fn` for the queue if it was in a stopped state when entered. Also see `blk_stop_queue`. Queue lock must be held.

blk_stop_queue

Name

`blk_stop_queue` — stop a queue

Synopsis

```
void blk_stop_queue (request_queue_t * q);
```

Arguments

q

The `&request_queue_t` in question

Description

The Linux block layer assumes that a block driver will consume all entries on the request queue when the `request_fn` strategy is called. Often this will not happen, because of hardware limitations (queue depth settings). If a device driver gets a 'queue full' response, or if it simply chooses not to queue more I/O at one point, it can call this function to prevent the `request_fn` from being called until the driver has signalled it's ready to go again. This happens by calling `blk_start_queue` to restart queue operations. Queue lock must be held.

blk_sync_queue

Name

`blk_sync_queue` — cancel any pending callbacks on a queue

Synopsis

```
void blk_sync_queue (struct request_queue * q);
```

Arguments

q

the queue

Description

The block layer may perform asynchronous callback activity on a queue, such as calling the unplug function after a timeout. A block device may call `blk_sync_queue` to ensure that any such activity is cancelled, thus allowing it to release resources the the callbacks might use. The caller must already have made sure that its `->make_request_fn` will not re-add plugging prior to calling this function.

blk_run_queue

Name

`blk_run_queue` — run a single device queue

Synopsis

```
void blk_run_queue (struct request_queue * q);
```

Arguments

q

The queue to run

blk_cleanup_queue

Name

`blk_cleanup_queue` — release a `request_queue_t` when it is no longer needed

Synopsis

```
void blk_cleanup_queue (request_queue_t * q);
```

Arguments

q

the request queue to be released

Description

`blk_cleanup_queue` is the pair to `blk_init_queue` or `blk_queue_make_request`. It should be called when a request queue is being released; typically when a block device is being de-registered. Currently, its primary task is to free all the `&struct request` structures that were allocated to the queue and the queue itself.

Caveat

Hopefully the low level driver will have finished any outstanding requests first...

blk_init_queue

Name

`blk_init_queue` — prepare a request queue for use with a block device

Synopsis

```
request_queue_t * blk_init_queue (request_fn_proc * rfn,
spinlock_t * lock);
```

Arguments

rfn

The function to be called to process requests that have been placed on the queue.

lock

Request queue spin lock

Description

If a block device wishes to use the standard request handling procedures, which sorts requests and coalesces adjacent requests, then it must call `blk_init_queue`. The function *rfn* will be called when there are requests on the queue that need to be processed. If the device supports plugging, then *rfn* may not be called immediately when requests are available on the queue, but may be called at some time later instead. Plugged queues are generally unplugged when a buffer belonging to one of the requests on the queue is needed, or due to memory pressure.

rfn is not required, or even expected, to remove all requests off the queue, but only as many as it can handle at a time. If it does leave requests on the queue, it is responsible for arranging that the requests get dealt with eventually.

The queue spin lock must be held while manipulating the requests on the request queue.

Function returns a pointer to the initialized request queue, or NULL if it didn't succeed.

Note

`blk_init_queue` must be paired with a `blk_cleanup_queue` call when the block device is deactivated (such as at module unload).

blk_requeue_request

Name

`blk_requeue_request` — put a request back on queue

Synopsis

```
void blk_requeue_request (request_queue_t * q, struct request  
* rq);
```

Arguments

q

request queue where request should be inserted

rq

request to be inserted

Description

Drivers often keep queueing requests until the hardware cannot accept more, when that condition happens we need to put the request back on the queue. Must be called with queue lock held.

blk_insert_request

Name

`blk_insert_request` — insert a special request in to a request queue

Synopsis

```
void blk_insert_request (request_queue_t * q, struct request *  
rq, int at_head, void * data, int reinsert);
```

Arguments

q

request queue where request should be inserted

rq

request to be inserted

at_head

insert request at head or tail of queue

data

private data

reinsert

true if request it a reinsertion of previously processed one

Description

Many block devices need to execute commands asynchronously, so they don't block the whole kernel from preemption during request execution. This is accomplished normally by inserting artificial requests tagged as `REQ_SPECIAL` in to the corresponding request queue, and letting them be scheduled for actual execution by the request queue.

We have the option of inserting the head or the tail of the queue. Typically we use the tail for new ioctls and so forth. We use the head of the queue for things like a QUEUE_FULL message from a device, or a host that is unable to accept a particular command.

blk_rq_map_user

Name

`blk_rq_map_user` — map user data to a request, for REQ_BLOCK_PC usage

Synopsis

```
struct request * blk_rq_map_user (request_queue_t * q, int
rw, void __user * ubuf, unsigned int len);
```

Arguments

q

request queue where request should be inserted

rw

READ or WRITE data

ubuf

the user buffer

len

length of user data

Description

Data will be mapped directly for zero copy io, if possible. Otherwise a kernel bounce buffer is used.

A matching `blk_rq_unmap_user` must be issued at the end of io, while still in process context.

Note

The mapped bio may need to be bounced through `blk_queue_bounce` before being submitted to the device, as pages mapped may be out of reach. It's the callers responsibility to make sure this happens. The original bio must be passed back in to `blk_rq_unmap_user` for proper unmapping.

blk_rq_unmap_user

Name

`blk_rq_unmap_user` — unmap a request with user data

Synopsis

```
int blk_rq_unmap_user (struct request * rq, struct bio * bio,
unsigned int ulen);
```

Arguments

rq

request to be unmapped

bio

-- undescribed --

ulen

length of user buffer

Description

Unmap a request previously mapped by `blk_rq_map_user`.

blk_execute_rq

Name

`blk_execute_rq` — insert a request into queue for execution

Synopsis

```
int blk_execute_rq (request_queue_t * q, struct gendisk *  
bd_disk, struct request * rq);
```

Arguments

q

queue to insert the request in

bd_disk

matching gendisk

rq

request to insert

Description

Insert a fully prepared request at the back of the io scheduler queue for execution.

blkdev_issue_flush

Name

`blkdev_issue_flush` — queue a flush

Synopsis

```
int blkdev_issue_flush (struct block_device * bdev, sector_t *  
error_sector);
```

Arguments

bdev

blockdev to issue flush for

error_sector

error sector

Description

Issue a flush for the block device in question. Caller can supply room for storing the error offset in case of a flush error, if they wish to. Caller must run

`wait_for_completion` on its own.

blkdev_scsi_issue_flush_fn

Name

`blkdev_scsi_issue_flush_fn` — issue flush for SCSI devices

Synopsis

```
int blkdev_scsi_issue_flush_fn (request_queue_t * q, struct  
gendisk * disk, sector_t * error_sector);
```

Arguments

q
device queue

disk
gendisk

error_sector
error offset

Description

Devices understanding the SCSI command set, can use this function as a helper for issuing a cache flush. Note: driver is required to store the error offset (in case of error flushing) in `->sector` of struct request.

blk_end_sync_rq

Name

`blk_end_sync_rq` — executes a completion event on a request

Synopsis

```
void blk_end_sync_rq (struct request * rq);
```

Arguments

rq

request to complete

blk_congestion_wait

Name

`blk_congestion_wait` — wait for a queue to become uncongested

Synopsis

```
long blk_congestion_wait (int rw, long timeout);
```

Arguments

rw

READ or WRITE

timeout

timeout in jiffies

Description

Waits for up to *timeout* jiffies for a queue (any queue) to exit congestion. If no queues are congested then just wait for the next request to be returned.

blk_attempt_remerge

Name

`blk_attempt_remerge` — attempt to remerge active head with next request

Synopsis

```
void blk_attempt_remerge (request_queue_t * q, struct request  
* rq);
```

Arguments

q

The `&request_queue_t` belonging to the device

rq

The head request (usually)

Description

For head-active devices, the queue can easily be unplugged so quickly that proper merging is not done on the front request. This may hurt performance greatly for some devices. The block layer cannot safely do merging on that first request for these queues, but the driver can call this function and make it happen any way. Only the driver knows when it is safe to do so.

generic_make_request

Name

`generic_make_request` —

Synopsis

```
void generic_make_request (struct bio * bio);
```

Arguments

bio

The bio describing the location in memory and on the device.

Description

`generic_make_request` is used to make I/O requests of block devices. It is passed a `&struct bio`, which describes the I/O that needs to be done.

`generic_make_request` does not return any status. The success/failure status of the request, along with notification of completion, is delivered asynchronously through the `bio->bi_end_io` function described (one day) else where.

The caller of `generic_make_request` must make sure that `bi_io_vec` are set to describe the memory buffer, and that `bi_dev` and `bi_sector` are set to describe the device address, and the `bi_end_io` and optionally `bi_private` are set to describe how completion notification should be signaled.

`generic_make_request` and the drivers it calls may use `bi_next` if this bio happens to be merged with someone else, and may change `bi_dev` and `bi_sector` for remaps as it sees fit. So the values of these fields should NOT be depended on after the call to `generic_make_request`.

submit_bio

Name

`submit_bio` —

Synopsis

```
void submit_bio (int rw, struct bio * bio);
```

Arguments

rw

whether to READ or WRITE, or maybe to READA (read ahead)

bio

The `&struct bio` which describes the I/O

Description

`submit_bio` is very similar in purpose to `generic_make_request`, and uses that function to do most of the work. Both are fairly rough interfaces, *bio* must be

presetup and ready for I/O.

end_that_request_first

Name

`end_that_request_first` — end I/O on a request

Synopsis

```
int end_that_request_first (struct request * req, int  
uptodate, int nr_sectors);
```

Arguments

req

the request being processed

uptodate

1 for success, 0 for I/O error, < 0 for specific error

nr_sectors

number of sectors to end I/O on

Description

Ends I/O on a number of sectors attached to *req*, and sets it up for the next range of segments (if any) in the cluster.

Return

0 - we are done with this request, call `end_that_request_last` 1 - still buffers pending for this request

end_that_request_chunk

Name

`end_that_request_chunk` — end I/O on a request

Synopsis

```
int end_that_request_chunk (struct request * req, int
uptodate, int nr_bytes);
```

Arguments

req

the request being processed

uptodate

1 for success, 0 for I/O error, < 0 for specific error

nr_bytes

number of bytes to complete

Description

Ends I/O on a number of bytes attached to *req*, and sets it up for the next range of segments (if any). Like `end_that_request_first`, but deals with bytes instead of sectors.

Return

0 - we are done with this request, call `end_that_request_last` 1 - still buffers pending for this request

Chapter 16. Miscellaneous Devices

misc_register

Name

`misc_register` — register a miscellaneous device

Synopsis

```
int misc_register (struct miscdevice * misc);
```

Arguments

misc

device structure

Description

Register a miscellaneous device with the kernel. If the minor number is set to `MISC_DYNAMIC_MINOR` a minor number is assigned and placed in the minor field of the structure. For other cases the minor number requested is used.

The structure passed is linked into the kernel and may not be destroyed until it has been unregistered.

A zero is returned on success and a negative `errno` code for failure.

misc_deregister

Name

`misc_deregister` — unregister a miscellaneous device

Synopsis

```
int misc_deregister (struct miscdevice * misc);
```

Arguments

misc

device to unregister

Description

Unregister a miscellaneous device that was previously successfully registered with `misc_register`. Success is indicated by a zero return, a negative `errno` code indicates an error.

Chapter 17. Video4Linux

video_register_device

Name

`video_register_device` — register video4linux devices

Synopsis

```
int video_register_device (struct video_device * vfd, int
type, int nr);
```

Arguments

vfd

video device structure we want to register

type

type of device to register

nr

which device number (0 == /dev/video0, 1 == /dev/video1, ... -1 == first free)

Description

The registration code assigns minor numbers based on the type requested. -ENFILE is returned in all the device slots for this category are full. If not then the minor field is set and the driver initialize function is called (if non NULL).

Zero is returned on success.

Valid types are

VFL_TYPE_GRABBER - A frame grabber

VFL_TYPE_VTX - A teletext device

VFL_TYPE_VBI - Vertical blank data (undecoded)

VFL_TYPE_RADIO - A radio card

video_unregister_device

Name

`video_unregister_device` — unregister a video4linux device

Synopsis

```
void video_unregister_device (struct video_device * vfd);
```

Arguments

vfd

the device to unregister

Description

This unregisters the passed device and deassigns the minor number. Future open calls will be met with errors.

Chapter 18. Sound Devices

register_sound_special

Name

`register_sound_special` — register a special sound node

Synopsis

```
int register_sound_special (struct file_operations * fops, int  
unit);
```

Arguments

fops

File operations for the driver

unit

Unit number to allocate

Description

Allocate a special sound device by minor number from the sound subsystem. The allocated number is returned on succes. On failure a negative error code is returned.

register_sound_mixer

Name

`register_sound_mixer` — register a mixer device

Synopsis

```
int register_sound_mixer (struct file_operations * fops, int  
dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a mixer device. Unit is the number of the mixer requested. Pass -1 to request the next free mixer unit. On success the allocated number is returned, on failure a negative error code is returned.

register_sound_midi

Name

`register_sound_midi` — register a midi device

Synopsis

```
int register_sound_midi (struct file_operations * fops, int
dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a midi device. Unit is the number of the midi device requested. Pass -1 to request the next free midi unit. On success the allocated number is returned, on failure a negative error code is returned.

register_sound_dsp

Name

`register_sound_dsp` — register a DSP device

Synopsis

```
int register_sound_dsp (struct file_operations * fops, int
dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a DSP device. Unit is the number of the DSP requested. Pass -1 to request the next free DSP unit. On success the allocated number is returned, on failure a negative error code is returned.

This function allocates both the audio and dsp device entries together and will always allocate them as a matching pair - eg dsp3/audio3

register_sound_synth

Name

`register_sound_synth` — register a synth device

Synopsis

```
int register_sound_synth (struct file_operations * fops, int  
dev);
```

Arguments

fops

File operations for the driver

dev

Unit number to allocate

Description

Allocate a synth device. Unit is the number of the synth device requested. Pass -1 to request the next free synth unit. On success the allocated number is returned, on failure a negative error code is returned.

unregister_sound_special

Name

`unregister_sound_special` — unregister a special sound device

Synopsis

```
void unregister_sound_special (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_special`. The unit passed is the return value from the register function.

unregister_sound_mixer

Name

`unregister_sound_mixer` — unregister a mixer

Synopsis

```
void unregister_sound_mixer (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_mixer`. The `unit` passed is the return value from the register function.

unregister_sound_midi

Name

`unregister_sound_midi` — unregister a midi device

Synopsis

```
void unregister_sound_midi (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_midi`. The `unit` passed is the return value from the register function.

unregister_sound_dsp

Name

`unregister_sound_dsp` — unregister a DSP device

Synopsis

```
void unregister_sound_dsp (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_dsp`. The unit passed is the return value from the register function.

Both of the allocated units are released together automatically.

unregister_sound_synth

Name

`unregister_sound_synth` — unregister a synth device

Synopsis

```
void unregister_sound_synth (int unit);
```

Arguments

unit

unit number to allocate

Description

Release a sound device that was allocated with `register_sound_synth`. The unit passed is the return value from the register function.

Chapter 19. 16x50 UART Driver

uart_update_timeout

Name

`uart_update_timeout` — update per-port FIFO timeout.

Synopsis

```
void uart_update_timeout (struct uart_port * port, unsigned  
int cflag, unsigned int baud);
```

Arguments

port

uart_port structure describing the port

cflag

termios cflag value

baud

speed of the port

Description

Set the port FIFO timeout value. The *cflag* value should reflect the actual hardware settings.

uart_get_baud_rate

Name

`uart_get_baud_rate` — return baud rate for a particular port

Synopsis

```
unsigned int uart_get_baud_rate (struct uart_port * port,  
struct termios * termios, struct termios * old, unsigned int  
min, unsigned int max);
```

Arguments

port

uart_port structure describing the port in question.

termios

desired termios settings.

old

old termios (or NULL)

min

minimum acceptable baud rate

max

maximum acceptable baud rate

Description

Decode the termios structure into a numeric baud rate, taking account of the magic 38400 baud rate (with `spd_*` flags), and mapping the `B0` rate to 9600 baud.

If the new baud rate is invalid, try the old `termios` setting. If it's still invalid, we try 9600 baud.

Update the `termios` structure to reflect the baud rate we're actually going to be using.

uart_get_divisor

Name

`uart_get_divisor` — return uart clock divisor

Synopsis

```
unsigned int uart_get_divisor (struct uart_port * port,  
unsigned int baud);
```

Arguments

port

uart_port structure describing the port.

baud

desired baud rate

Description

Calculate the uart clock divisor for the port.

uart_register_driver

Name

`uart_register_driver` — register a driver with the uart core layer

Synopsis

```
int uart_register_driver (struct uart_driver * drv);
```

Arguments

drv

low level driver structure

Description

Register a uart driver with the core driver. We in turn register with the tty layer, and initialise the core driver per-port state.

We have a proc file in `/proc/tty/driver` which is named after the normal driver.

`drv->port` should be NULL, and the per-port structures should be registered using `uart_add_one_port` after this call has succeeded.

uart_unregister_driver

Name

`uart_unregister_driver` — remove a driver from the uart core layer

Synopsis

```
void uart_unregister_driver (struct uart_driver * drv);
```

Arguments

drv

low level driver structure

Description

Remove all references to a driver from the core driver. The low level driver must have removed all its ports via the `uart_remove_one_port` if it registered them with `uart_add_one_port`. (ie, `drv->port == NULL`)

uart_add_one_port

Name

`uart_add_one_port` — attach a driver-defined port structure

Synopsis

```
int uart_add_one_port (struct uart_driver * drv, struct  
uart_port * port);
```

Arguments

drv

pointer to the uart low level driver structure for this port

port

uart port structure to use for this port.

Description

This allows the driver to register its own `uart_port` structure with the core driver. The main purpose is to allow the low level uart drivers to expand `uart_port`, rather than having yet more levels of structures.

uart_remove_one_port

Name

`uart_remove_one_port` — detach a driver defined port structure

Synopsis

```
int uart_remove_one_port (struct uart_driver * drv, struct
uart_port * port);
```

Arguments

drv

pointer to the uart low level driver structure for this port

port

uart port structure for this port

Description

This unhooks (and hangs up) the specified port structure from the core driver. No further calls will be made to the low-level code for this port.

uart_register_port

Name

uart_register_port —

Synopsis

```
int uart_register_port (struct uart_driver * drv, struct
uart_port * port);
```

Arguments

drv

pointer to the uart low level driver structure for this port

port

uart port structure describing the port

Description

Register UART settings with the specified low level driver. Detect the type of the port if UPF_BOOT_AUTOCONF is set, and detect the IRQ if UPF_AUTO_IRQ is set.

We try to pick the same port for the same IO base address, so that when a modem is plugged in, unplugged and plugged back in, it gets allocated the same port.

Returns negative error, or positive line number.

uart_unregister_port

Name

`uart_unregister_port` — de-allocate a port

Synopsis

```
void uart_unregister_port (struct uart_driver * drv, int  
line);
```

Arguments

drv

pointer to the uart low level driver structure for this port

line

line index previously returned from `uart_register_port`

Description

Hang up the specified line associated with the low level driver, and mark the port as unused.

serial8250_suspend_port

Name

`serial8250_suspend_port` — suspend one serial port

Synopsis

```
void serial8250_suspend_port (int line);
```

Arguments

line

serial line number

Description

Suspend one serial port.

serial8250_resume_port

Name

`serial8250_resume_port` — resume one serial port

Synopsis

```
void serial8250_resume_port (int line);
```

Arguments

line

serial line number

Description

Resume one serial port.

serial8250_register_port

Name

`serial8250_register_port` — register a serial port

Synopsis

```
int serial8250_register_port (struct uart_port * port);
```

Arguments

port

serial port template

Description

Configure the serial port specified by the request. If the port exists and is in use, it is hung up and unregistered first.

The port is then probed and if necessary the IRQ is autodetected. If this fails an error is returned.

On success the port is ready to use and the line number is returned.

serial8250_unregister_port

Name

`serial8250_unregister_port` — remove a 16x50 serial port at runtime

Synopsis

```
void serial8250_unregister_port (int line);
```

Arguments

line

serial line number

Description

Remove one serial port. This may not be called from interrupt context. We hand the port back to the our control.

register_serial

Name

`register_serial` — configure a 16x50 serial port at runtime

Synopsis

```
int register_serial (struct serial_struct * req);
```

Arguments

req

request structure

Description

Configure the serial port specified by the request. If the port exists and is in use an error is returned. If the port is not currently in the table it is added.

The port is then probed and if necessary the IRQ is autodetected If this fails an error is returned.

On success the port is ready to use and the line number is returned.

unregister_serial

Name

`unregister_serial` — remove a 16x50 serial port at runtime

Synopsis

```
void unregister_serial (int line);
```

Arguments

line

serial line number

Description

Remove one serial port. This may not be called from interrupt context. We hand the port back to our local PM control.

Chapter 20. Z85230 Support Library

z8530_interrupt

Name

`z8530_interrupt` — Handle an interrupt from a Z8530

Synopsis

```
irqreturn_t z8530_interrupt (int irq, void * dev_id, struct  
pt_regs * regs);
```

Arguments

irq

Interrupt number

dev_id

The Z8530 device that is interrupting.

regs

unused

Description

A Z85[2]30 device has stuck its hand in the air for attention. We scan both the channels on the chip for events and then call the channel specific call backs for each channel that has events. We have to use callback functions because the two channels can be in different modes.

Locking is done for the handlers. Note that locking is done at the chip level (the 5uS delay issue is per chip not per channel). `c->lock` for both channels points to `dev->lock`

z8530_sync_open

Name

`z8530_sync_open` — Open a Z8530 channel for PIO

Synopsis

```
int z8530_sync_open (struct net_device * dev, struct
z8530_channel * c);
```

Arguments

dev

The network interface we are using

c

The Z8530 channel to open in synchronous PIO mode

Description

Switch a Z8530 into synchronous mode without DMA assist. We raise the RTS/DTR and commence network operation.

z8530_sync_close

Name

`z8530_sync_close` — Close a PIO Z8530 channel

Synopsis

```
int z8530_sync_close (struct net_device * dev, struct
z8530_channel * c);
```

Arguments

dev

Network device to close

c

Z8530 channel to disassociate and move to idle

Description

Close down a Z8530 interface and switch its interrupt handlers to discard future events.

z8530_sync_dma_open

Name

`z8530_sync_dma_open` — Open a Z8530 for DMA I/O

Synopsis

```
int z8530_sync_dma_open (struct net_device * dev, struct  
z8530_channel * c);
```

Arguments

dev

The network device to attach

c

The Z8530 channel to configure in sync DMA mode.

Description

Set up a Z85x30 device for synchronous DMA in both directions. Two ISA DMA channels must be available for this to work. We assume ISA DMA driven I/O and PC limits on access.

z8530_sync_dma_close

Name

`z8530_sync_dma_close` — Close down DMA I/O

Synopsis

```
int z8530_sync_dma_close (struct net_device * dev, struct  
z8530_channel * c);
```

Arguments

dev

Network device to detach

c

Z8530 channel to move into discard mode

Description

Shut down a DMA mode synchronous interface. Halt the DMA, and free the buffers.

z8530_sync_txdma_open

Name

`z8530_sync_txdma_open` — Open a Z8530 for TX driven DMA

Synopsis

```
int z8530_sync_txdma_open (struct net_device * dev, struct  
z8530_channel * c);
```

Arguments

dev

The network device to attach

c

The Z8530 channel to configure in sync DMA mode.

Description

Set up a Z85x30 device for synchronous DMA transmission. One ISA DMA channel must be available for this to work. The receive side is run in PIO mode, but then it has the bigger FIFO.

z8530_sync_txdma_close

Name

`z8530_sync_txdma_close` — Close down a TX driven DMA channel

Synopsis

```
int z8530_sync_txdma_close (struct net_device * dev, struct
z8530_channel * c);
```

Arguments

dev

Network device to detach

c

Z8530 channel to move into discard mode

Description

Shut down a DMA/PIO split mode synchronous interface. Halt the DMA, and free the buffers.

z8530_describe

Name

z8530_describe — Uniformly describe a Z8530 port

Synopsis

```
void z8530_describe (struct z8530_dev * dev, char * mapping,  
unsigned long io);
```

Arguments

dev

Z8530 device to describe

mapping

string holding mapping type (eg “I/O” or “Mem”)

io

the port value in question

Description

Describe a Z8530 in a standard format. We must pass the I/O as the port offset isn't predictable. The main reason for this function is to try and get a common format of report.

z8530_init

Name

`z8530_init` — Initialise a Z8530 device

Synopsis

```
int z8530_init (struct z8530_dev * dev);
```

Arguments

dev

Z8530 device to initialise.

Description

Configure up a Z8530/Z85C30 or Z85230 chip. We check the device is present, identify the type and then program it to hopefully keep quite and behave. This matters a lot, a Z8530 in the wrong state will sometimes get into stupid modes generating 10Khz interrupt streams and the like.

We set the interrupt handler up to discard any events, in case we get them during reset or setp.

Return 0 for success, or a negative value indicating the problem in errno form.

z8530_shutdown

Name

`z8530_shutdown` — Shutdown a Z8530 device

Synopsis

```
int z8530_shutdown (struct z8530_dev * dev);
```

Arguments

dev

The Z8530 chip to shutdown

Description

We set the interrupt handlers to silence any interrupts. We then reset the chip and wait 100uS to be sure the reset completed. Just in case the caller then tries to do stuff.

This is called without the lock held

z8530_channel_load

Name

`z8530_channel_load` — Load channel data

Synopsis

```
int z8530_channel_load (struct z8530_channel * c, u8 *  
rtable);
```

Arguments

c

Z8530 channel to configure

rtable

table of register, value pairs

FIXME

ioctl to allow user uploaded tables

Load a Z8530 channel up from the system data. We use +16 to indicate the “prime” registers. The value 255 terminates the table.

z8530_null_rx

Name

`z8530_null_rx` — Discard a packet

Synopsis

```
void z8530_null_rx (struct z8530_channel * c, struct sk_buff *  
skb);
```


Arguments

c

The channel the packet arrived on

skb

The buffer

Description

We point the receive handler at this function when idle. Instead of syncppp processing the frames we get to throw them away.

z8530_queue_xmit

Name

`z8530_queue_xmit` — Queue a packet

Synopsis

```
int z8530_queue_xmit (struct z8530_channel * c, struct sk_buff
* skb);
```

Arguments

c

The channel to use

skb

The packet to kick down the channel

Description

Queue a packet for transmission. Because we have rather hard to hit interrupt latencies for the Z85230 per packet even in DMA mode we do the flip to DMA buffer if needed here not in the IRQ.

Called from the network code. The lock is not held at this point.

z8530_get_stats

Name

`z8530_get_stats` — Get network statistics

Synopsis

```
struct net_device_stats * z8530_get_stats (struct  
z8530_channel * c);
```

Arguments

c

The channel to use

Description

Get the statistics block. We keep the statistics in software as the chip doesn't do it for us.

Locking is ignored here - we could lock for a copy but its not likely to be that big an issue

Chapter 21. Frame Buffer Library

The frame buffer drivers depend heavily on four data structures. These structures are declared in `include/linux/fb.h`. They are `fb_info`, `fb_var_screeninfo`, `fb_fix_screeninfo` and `fb_monospecs`. The last three can be made available to and from userland.

`fb_info` defines the current state of a particular video card. Inside `fb_info`, there exists a `fb_ops` structure which is a collection of needed functions to make `fbdev` and `fbcon` work. `fb_info` is only visible to the kernel.

`fb_var_screeninfo` is used to describe the features of a video card that are user defined. With `fb_var_screeninfo`, things such as depth and the resolution may be defined.

The next structure is `fb_fix_screeninfo`. This defines the properties of a card that are created when a mode is set and can't be changed otherwise. A good example of this is the start of the frame buffer memory. This "locks" the address of the frame buffer memory, so that it cannot be changed or moved.

The last structure is `fb_monospecs`. In the old API, there was little importance for `fb_monospecs`. This allowed for forbidden things such as setting a mode of 800x600 on a fix frequency monitor. With the new API, `fb_monospecs` prevents such things, and if used correctly, can prevent a monitor from being cooked. `fb_monospecs` will not be useful until kernels 2.5.x.

21.1. Frame Buffer Memory

`register_framebuffer`

Name

`register_framebuffer` — registers a frame buffer device

Synopsis

```
int register_framebuffer (struct fb_info * fb_info);
```

Arguments

fb_info

frame buffer info structure

Description

Registers a frame buffer device *fb_info*.

Returns negative `errno` on error, or zero for success.

unregister_framebuffer

Name

`unregister_framebuffer` — releases a frame buffer device

Synopsis

```
int unregister_framebuffer (struct fb_info * fb_info);
```

Arguments

fb_info

frame buffer info structure

Description

Unregisters a frame buffer device *fb_info*.

Returns negative `errno` on error, or zero for success.

fb_register_client

Name

`fb_register_client` — register a client notifier

Synopsis

```
int fb_register_client (struct notifier_block * nb);
```

Arguments

nb

notifier block to callback on events

fb_unregister_client

Name

`fb_unregister_client` — unregister a client notifier

Synopsis

```
int fb_unregister_client (struct notifier_block * nb);
```

Arguments

nb

notifier block to callback on events

fb_set_suspend

Name

`fb_set_suspend` — low level driver signals suspend

Synopsis

```
void fb_set_suspend (struct fb_info * info, int state);
```

Arguments

info

framebuffer affected

state

0 = resuming, !=0 = suspending

Description

This is meant to be used by low level drivers to signal suspend/resume to the core & clients. It must be called with the console semaphore held

fb_get_options

Name

`fb_get_options` — get kernel boot parameters

Synopsis

```
int fb_get_options (char * name, char ** option);
```

Arguments

name

-- undescribed --

option

-- undescribed --

Description

name - framebuffer name as it would appear in the boot parameter line
(video=<name>:<options>)

NOTE

Needed to maintain backwards compatibility

21.2. Frame Buffer Console

/usr/src/packages/BUILD/linux-2.6.11.4-20a/drivers/video/console/fbcon.c

Name

/usr/src/packages/BUILD/linux-2.6.11.4-20a/drivers/video/console/f
— Document generation inconsistency

Oops

Warning

The template for this document tried to insert the structured comment from the file

/usr/src/packages/BUILD/linux-2.6.11.4-20a/drivers/video/console/fbcon.c
at this point, but none was found. This dummy section is inserted to allow generation to continue.

21.3. Frame Buffer Colormap

fb_alloc_cmap

Name

fb_alloc_cmap — allocate a colormap

Synopsis

```
int fb_alloc_cmap (struct fb_cmap * cmap, int len, int
transp);
```

Arguments

cmap

frame buffer colormap structure

len

length of *cmap*

transp

boolean, 1 if there is transparency, 0 otherwise

Description

Allocates memory for a colormap *cmap*. *len* is the number of entries in the palette.

Returns -1 `errno` on error, or zero on success.

fb_dealloc_cmap

Name

`fb_dealloc_cmap` — deallocate a colormap

Synopsis

```
void fb_dealloc_cmap (struct fb_cmap * cmap);
```

Arguments

cmap

frame buffer colormap structure

Description

Deallocates a colormap that was previously allocated with `fb_alloc_cmap`.

fb_copy_cmap

Name

`fb_copy_cmap` — copy a colormap

Synopsis

```
int fb_copy_cmap (struct fb_cmap * from, struct fb_cmap * to);
```

Arguments

from

frame buffer colormap structure

to

frame buffer colormap structure

Description

Copy contents of colormap from *from* to *to*.

fb_set_cmap

Name

`fb_set_cmap` — set the colormap

Synopsis

```
int fb_set_cmap (struct fb_cmap * cmap, struct fb_info *  
info);
```

Arguments

cmap

frame buffer colormap structure

info

frame buffer info structure

Description

Sets the colormap *cmap* for a screen of device *info*.

Returns negative `errno` on error, or zero on success.

fb_default_cmap

Name

`fb_default_cmap` — get default colormap

Synopsis

```
struct fb_cmap * fb_default_cmap (int len);
```

Arguments

len

size of palette for a depth

Description

Gets the default colormap for a specific screen depth. *len* is the size of the palette for a particular screen depth.

Returns pointer to a frame buffer colormap structure.

fb_invert_cmaps

Name

`fb_invert_cmaps` — invert all defaults colormaps

Synopsis

```
void fb_invert_cmaps ( void );
```

Arguments

void

no arguments

Description

Invert all default colormaps.

21.4. Frame Buffer Video Mode Database

fb_try_mode

Name

`fb_try_mode` — test a video mode

Synopsis

```
int fb_try_mode (struct fb_var_screeninfo * var, struct  
fb_info * info, const struct fb_videomode * mode, unsigned int  
bpp);
```

Arguments

<i>var</i>	frame buffer user defined part of display
<i>info</i>	frame buffer info structure
<i>mode</i>	frame buffer video mode structure
<i>bpp</i>	color depth in bits per pixel

Description

Tries a video mode to test it's validity for device *info*.

Returns 1 on success.

fb_find_mode

Name

`fb_find_mode` — finds a valid video mode

Synopsis

```
int fb_find_mode (struct fb_var_screeninfo * var, struct
fb_info * info, const char * mode_option, const struct
fb_videomode * db, unsigned int dbsize, const struct
fb_videomode * default_mode, unsigned int default_bpp);
```

Arguments

var

frame buffer user defined part of display

info

frame buffer info structure

mode_option

string video mode to find

db

video mode database

dbsize

size of *db*

default_mode

default video mode to fall back to

default_bpp

default color depth in bits per pixel

Description

Finds a suitable video mode, starting with the specified mode in *mode_option* with fallback to *default_mode*. If *default_mode* fails, all modes in the video mode database will be tried.

Valid mode specifiers for *mode_option*:

<xres>x<yres>[-<bpp>][@<refresh>] or <name>[-<bpp>][@<refresh>]

with <xres>, <yres>, <bpp> and <refresh> decimal numbers and <name> a string.

NOTE

The passed struct *var* is *_not_* cleared! This allows you to supply values for e.g. the grayscale and accel_flags fields.

Returns zero for failure, 1 if using specified *mode_option*, 2 if using specified *mode_option* with an ignored refresh rate, 3 if default mode is used, 4 if fall back to any valid mode.

fb_var_to_videomode

Name

`fb_var_to_videomode` — convert `fb_var_screeninfo` to `fb_videomode`

Synopsis

```
void fb_var_to_videomode (struct fb_videomode * mode, struct
fb_var_screeninfo * var);
```

Arguments

mode

pointer to struct `fb_videomode`

var

pointer to struct `fb_var_screeninfo`

fb_videomode_to_var

Name

`fb_videomode_to_var` — convert `fb_videomode` to `fb_var_screeninfo`

Synopsis

```
void fb_videomode_to_var (struct fb_var_screeninfo * var,
struct fb_videomode * mode);
```

Arguments

var

pointer to struct fb_var_screeninfo

mode

pointer to struct fb_videomode

fb_mode_is_equal

Name

`fb_mode_is_equal` — compare 2 videomodes

Synopsis

```
int fb_mode_is_equal (struct fb_videomode * model, struct
fb_videomode * mode2);
```

Arguments

model

first videomode

mode2

second videomode

RETURNS

1 if equal, 0 if not

fb_find_best_mode

Name

`fb_find_best_mode` — find best matching videomode

Synopsis

```
struct fb_videomode * fb_find_best_mode (struct  
fb_var_screeninfo * var, struct list_head * head);
```

Arguments

var

pointer to struct `fb_var_screeninfo`

head

pointer to struct `list_head` of modelist

RETURNS

struct `fb_videomode`, NULL if none found

IMPORTANT

This function assumes that all modelist entries in info->modelist are valid.

NOTES

Finds best matching videomode which has an equal or greater dimension than var->xres and var->yres. If more than 1 videomode is found, will return the videomode with the highest refresh rate

fb_match_mode

Name

`fb_match_mode` — find a videomode which exactly matches the timings in var

Synopsis

```
struct fb_videomode * fb_match_mode (struct fb_var_screeninfo
* var, struct list_head * head);
```

Arguments

var

pointer to struct fb_var_screeninfo

head

pointer to struct list_head of modelist

RETURNS

struct fb_videomode, NULL if none found

fb_add_videomode

Name

fb_add_videomode —

Synopsis

```
int fb_add_videomode (struct fb_videomode * mode, struct  
list_head * head);
```

Arguments

mode

videomode to add

head

struct list_head of modelist

NOTES

Will only add unmatched mode entries

fb_delete_videomode

Name

`fb_delete_videomode` —

Synopsis

```
void fb_delete_videomode (struct fb_videomode * mode, struct  
list_head * head);
```

Arguments

mode

videomode to remove

head

struct list_head of modelist

NOTES

Will remove all matching mode entries

fb_destroy_modelist

Name

`fb_destroy_modelist` —

Synopsis

```
void fb_destroy_modelist (struct list_head * head);
```

Arguments

head

struct list_head of modelist

fb_videomode_to_modelist

Name

fb_videomode_to_modelist —

Synopsis

```
void fb_videomode_to_modelist (struct fb_videomode * modedb,  
int num, struct list_head * head);
```

Arguments

modedb

array of struct fb_videomode

num

number of entries in array

head

struct list_head of modelist

21.5. Frame Buffer Macintosh Video Mode Database

mac_vmode_to_var

Name

`mac_vmode_to_var` — converts vmode/cmode pair to var structure

Synopsis

```
int mac_vmode_to_var (int vmode, int cmode, struct
fb_var_screeninfo * var);
```

Arguments

vmode

MacOS video mode

cmode

MacOS color mode

var

frame buffer video mode structure

Description

Converts a MacOS vmode/cmode pair to a frame buffer video mode structure.

Returns negative errno on error, or zero for success.

mac_var_to_vmode

Name

`mac_var_to_vmode` — convert var structure to MacOS vmode/cmode pair

Synopsis

```
int mac_var_to_vmode (const struct fb_var_screeninfo * var,  
int * vmode, int * cmode);
```

Arguments

var

frame buffer video mode structure

vmode

MacOS video mode

cmode

MacOS color mode

Description

Converts a frame buffer video mode structure to a MacOS vmode/cmode pair.

Returns negative errno on error, or zero for success.

mac_map_monitor_sense

Name

`mac_map_monitor_sense` — Convert monitor sense to vmode

Synopsis

```
int mac_map_monitor_sense (int sense);
```

Arguments

sense

Macintosh monitor sense number

Description

Converts a Macintosh monitor sense number to a MacOS vmode number.

Returns MacOS vmode video mode number.

mac_find_mode

Name

`mac_find_mode` — find a video mode

Synopsis

```
int __init mac_find_mode (struct fb_var_screeninfo * var,  
struct fb_info * info, const char * mode_option, unsigned int  
default_bpp);
```

Arguments

var

frame buffer user defined part of display

info

frame buffer info structure

mode_option

video mode name (see `mac_modesdb[]`)

default_bpp

default color depth in bits per pixel

Description

Finds a suitable video mode. Tries to set mode specified by *mode_option*. If the name of the wanted mode begins with 'mac', the Mac video mode database will be used, otherwise it will fall back to the standard video mode database.

Note

Function marked as `__init` and can only be used during system boot.

Returns error code from `fb_find_mode` (see `fb_find_mode` function).

21.6. Frame Buffer Fonts

Refer to the file `drivers/video/console/fonts.c` for more information.

