

# 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

<b>1. Data Types .....</b>	<b>1</b>
1.1. Doubly Linked Lists .....	1
list_add .....	1
list_add_tail .....	2
__list_del_entry .....	2
list_replace .....	3
list_del_init .....	4
list_move .....	5
list_move_tail .....	5
list_is_last .....	6
list_empty .....	7
list_empty_careful .....	8
list_rotate_left .....	9
list_is_singular .....	9
list_cut_position .....	10
list_splice .....	11
list_splice_tail .....	12
list_splice_init .....	12
list_splice_tail_init .....	13
list_entry .....	14
list_first_entry .....	15
list_for_each .....	16
__list_for_each .....	17
list_for_each_prev .....	18
list_for_each_safe .....	18
list_for_each_prev_safe .....	19
list_for_each_entry .....	20
list_for_each_entry_reverse .....	21
list_prepare_entry .....	22
list_for_each_entry_continue .....	23
list_for_each_entry_continue_reverse .....	24
list_for_each_entry_from .....	25
list_for_each_entry_safe .....	26
list_for_each_entry_safe_continue .....	27
list_for_each_entry_safe_from .....	28
list_for_each_entry_safe_reverse .....	29
list_safe_reset_next .....	30
hlist_for_each_entry .....	31
hlist_for_each_entry_continue .....	31
hlist_for_each_entry_from .....	32
hlist_for_each_entry_safe .....	33

<b>2. Basic C Library Functions .....</b>	<b>35</b>
2.1. String Conversions .....	35
simple_strtoul .....	35
simple_strtoul .....	36
simple_strtol .....	36
simple_strtoll .....	37
vsprintf .....	38
vsnprintf .....	40
snprintf .....	41
scnprintf .....	42
vsprintf .....	43
sprintf .....	44
vbin_printf .....	45
bstr_printf .....	46
bprintf .....	47
vsscanf .....	48
sscanf .....	49
2.2. String Manipulation .....	50
strnicmp .....	50
strcpy .....	51
strncpy .....	52
strncpy .....	53
strcat .....	54
strncat .....	55
strlcat .....	56
strcmp .....	56
strncmp .....	57
strchr .....	58
strchr .....	59
strnchr .....	59
skip_spaces .....	60
strim .....	61
strlen .....	62
strnlen .....	63
strspn .....	63
strcspn .....	64
strpbrk .....	65
strsep .....	66
sysfs_streq .....	67
strtobool .....	68
memset .....	68
memcpy .....	69
memmove .....	70

memcmp.....	71
memscan .....	72
strstr.....	73
strnstr.....	74
memchr .....	75
2.3. Bit Operations .....	76
set_bit.....	76
__set_bit.....	77
clear_bit.....	78
__change_bit.....	79
change_bit.....	80
test_and_set_bit.....	81
test_and_set_bit_lock.....	81
__test_and_set_bit.....	82
test_and_clear_bit .....	83
__test_and_clear_bit .....	84
test_and_change_bit.....	85
test_bit.....	86
__ffs .....	87
ffz .....	88
ffs .....	88
fls.....	89
<b>3. Basic Kernel Library Functions .....</b>	<b>91</b>
3.1. Bitmap Operations .....	91
__bitmap_shift_right.....	91
__bitmap_shift_left.....	92
bitmap_scnprintf .....	93
__bitmap_parse .....	94
bitmap_parse_user .....	95
bitmap_scnlistprintf .....	96
bitmap_parselist_user .....	97
bitmap_remap .....	99
bitmap_bitremap .....	100
bitmap_onto .....	101
bitmap_fold.....	104
bitmap_find_free_region.....	105
bitmap_release_region .....	106
bitmap_allocate_region.....	107
bitmap_copy_le.....	108
__bitmap_parselist.....	109
bitmap_pos_to_ord .....	110
bitmap_ord_to_pos .....	111
3.2. Command-line Parsing.....	112

get_option .....	113
get_options .....	114
memparse .....	115
3.3. CRC Functions .....	116
crc7 .....	116
crc16 .....	117
crc_itu_t .....	118
/usr/src/linux-3.0.76-0.11//lib/crc32.c .....	118
crc_ccitt .....	119
3.4. idr/ida Functions .....	120
idr_pre_get .....	120
idr_get_new_above .....	121
idr_get_new .....	123
idr_remove .....	123
idr_remove_all .....	124
idr_destroy .....	125
idr_find .....	126
idr_for_each .....	127
idr_get_next .....	128
idr_replace .....	129
idr_init .....	130
ida_pre_get .....	130
ida_get_new_above .....	131
ida_get_new .....	132
ida_remove .....	133
ida_destroy .....	134
ida_simple_get .....	135
ida_simple_remove .....	136
ida_init .....	137
<b>4. Memory Management in Linux .....</b>	<b>139</b>
4.1. The Slab Cache .....	139
kcalloc .....	139
kmallocc_node .....	140
kzalloc .....	141
kzalloc_node .....	142
kmem_cache_create .....	143
kmem_cache_shrink .....	144
kmem_cache_destroy .....	145
kmem_cache_alloc .....	146
kmem_cache_free .....	147
kfree .....	148
ksize .....	149
4.2. User Space Memory Access .....	150

__copy_to_user_inatomic .....	150
__copy_to_user .....	151
__copy_from_user .....	152
strlen_user .....	154
__strncpy_from_user .....	155
strncpy_from_user .....	156
clear_user .....	157
__clear_user .....	158
strnlen_user .....	159
copy_to_user .....	160
__copy_from_user .....	161
4.3. More Memory Management Functions .....	162
read_cache_pages .....	162
page_cache_sync_readahead .....	163
page_cache_async_readahead.....	164
delete_from_page_cache.....	165
filemap_flush.....	166
filemap_fdatawait_range.....	167
filemap_fdatawait.....	168
filemap_write_and_wait_range.....	169
replace_page_cache_page.....	170
add_to_page_cache_locked .....	171
add_page_wait_queue.....	172
unlock_page .....	173
end_page_writeback .....	174
__lock_page .....	175
find_get_page.....	175
find_lock_page.....	176
find_or_create_page.....	177
find_get_pages_contig .....	178
find_get_pages_tag .....	180
grab_cache_page_nowait .....	181
generic_file_aio_read.....	182
filemap_fault .....	183
read_cache_page_async .....	184
read_cache_page_gfp.....	185
read_cache_page .....	186
__generic_file_aio_write.....	187
generic_file_aio_write.....	188
try_to_release_page .....	189
zap_page_range.....	190
zap_vma_ptes.....	191
__get_user_pages.....	192

vm_insert_page .....	194
vm_insert_pfn .....	195
remap_pfn_range .....	196
vm_iomap_memory .....	198
unmap_mapping_range .....	199
follow_pfn .....	200
vm_unmap_aliases .....	201
vm_unmap_ram .....	202
vm_map_ram .....	202
unmap_kernel_range_noflush .....	203
vfree .....	204
vunmap .....	205
vmap .....	206
vmalloc .....	207
vzalloc .....	208
vmalloc_user .....	209
vmalloc_node .....	210
vzalloc_node .....	211
vmalloc_32 .....	212
vmalloc_32_user .....	212
remap_vmalloc_range .....	213
alloc_vm_area .....	214
find_next_best_node .....	215
free_bootmem_with_active_regions .....	216
sparse_memory_present_with_active_regions .....	217
get_pfn_range_for_nid .....	218
absent_pages_in_range .....	219
add_active_range .....	220
remove_active_range .....	221
remove_all_active_ranges .....	222
find_min_pfn_with_active_regions .....	222
free_area_init_nodes .....	223
set_dma_reserve .....	224
setup_per_zone_wmarks .....	225
get_pageblock_flags_group .....	226
set_pageblock_flags_group .....	227
mempool_create .....	228
mempool_resize .....	229
mempool_destroy .....	230
mempool_alloc .....	231
mempool_free .....	232
dma_pool_create .....	233
dma_pool_destroy .....	234



dma_pool_alloc .....	235
dma_pool_free .....	236
dmam_pool_create .....	237
dmam_pool_destroy .....	238
balance_dirty_pages_ratelimited_nr .....	239
tag_pages_for_writeback .....	240
write_cache_pages .....	241
generic_writepages .....	242
write_one_page .....	243
wait_for_stable_page .....	244
truncate_inode_pages_range .....	245
truncate_inode_pages .....	246
invalidate_mapping_pages .....	247
invalidate_inode_pages2_range .....	248
invalidate_inode_pages2 .....	249
truncate_pagecache .....	250
truncate_setsize .....	251
vmtruncate .....	252
<b>5. Kernel IPC facilities .....</b>	<b>255</b>
5.1. IPC utilities .....	255
ipc_init .....	255
ipc_init_ids .....	256
ipc_init_proc_interface .....	256
ipc_findkey .....	257
ipc_get_maxid .....	258
ipc_addid .....	259
ipcget_new .....	260
ipc_check_perms .....	261
ipcget_public .....	262
ipc_rmid .....	263
ipc_alloc .....	264
ipc_free .....	265
ipc_rcu_alloc .....	266
ipc_schedule_free .....	267
ipc_immediate_free .....	268
ipcperms .....	268
kernel_to_ipc64_perm .....	269
ipc64_perm_to_ipc_perm .....	270
ipc_lock .....	271
ipcget .....	272
ipc_update_perm .....	273
ipcctl_pre_down .....	274
ipc_parse_version .....	275

<b>6. FIFO Buffer .....</b>	<b>277</b>
6.1. kfifo interface .....	277
DECLARE_KFIFO_PTR .....	277
DECLARE_KFIFO .....	277
INIT_KFIFO .....	278
DEFINE_KFIFO .....	279
kfifo_initialized .....	280
kfifo_esize .....	281
kfifo_recsz .....	281
kfifo_size .....	282
kfifo_reset .....	283
kfifo_reset_out .....	283
kfifo_len .....	284
kfifo_is_empty .....	285
kfifo_is_full .....	286
kfifo_avail .....	286
kfifo_skip .....	287
kfifo_peek_len .....	288
kfifo_alloc .....	288
kfifo_free .....	289
kfifo_init .....	290
kfifo_put .....	291
kfifo_get .....	292
kfifo_peek .....	293
kfifo_in .....	294
kfifo_in_spinlocked .....	295
kfifo_out .....	296
kfifo_out_spinlocked .....	297
kfifo_from_user .....	298
kfifo_to_user .....	299
kfifo_dma_in_prepare .....	300
kfifo_dma_in_finish .....	302
kfifo_dma_out_prepare .....	302
kfifo_dma_out_finish .....	304
kfifo_out_peek .....	304
<b>7. relay interface support .....</b>	<b>307</b>
7.1. relay interface .....	307
relay_buf_full .....	307
relay_reset .....	307
relay_open .....	308
relay_switch_subbuf .....	310
relay_subbufs_consumed .....	311
relay_close .....	312

relay_flush.....	312
relay_mmap_buf .....	313
relay_alloc_buf .....	314
relay_create_buf.....	315
relay_destroy_channel .....	316
relay_destroy_buf.....	317
relay_remove_buf .....	317
relay_buf_empty .....	318
wakeup_readers.....	319
__relay_reset.....	320
relay_close_buf .....	321
relay_hotcpu_callback .....	321
relay_late_setup_files.....	322
relay_file_open.....	323
relay_file_mmap .....	324
relay_file_poll .....	325
relay_file_release .....	326
relay_file_read_subbuf_avail .....	327
relay_file_read_start_pos .....	328
relay_file_read_end_pos .....	329
<b>8. Module Support .....</b>	<b>331</b>
8.1. Module Loading.....	331
__request_module .....	331
call_usermodehelper_setup.....	332
call_usermodehelper_setfns .....	333
call_usermodehelper_exec .....	334
8.2. Inter Module support.....	335
<b>9. Hardware Interfaces .....</b>	<b>337</b>
9.1. Interrupt Handling.....	337
synchronize_irq.....	337
irq_set_affinity_notifier.....	337
disable_irq_nosync .....	338
disable_irq.....	339
enable_irq.....	340
irq_set_irq_wake.....	341
setup_irq.....	342
remove_irq .....	343
free_irq.....	344
request_threaded_irq.....	345
request_any_context_irq .....	346
9.2. DMA Channels .....	348
request_dma .....	348

free_dma .....	349
9.3. Resources Management .....	349
request_resource_conflict .....	349
reallocate_resource .....	350
insert_resource_conflict .....	351
insert_resource .....	352
insert_resource_expand_to_fit .....	353
resource_alignment .....	354
request_resource .....	355
release_resource .....	356
allocate_resource .....	357
adjust_resource .....	358
__request_region .....	359
__check_region .....	360
__release_region .....	361
9.4. MTRR Handling .....	362
mtrr_add .....	362
mtrr_del .....	364
9.5. PCI Support Library .....	365
pci_bus_max_busnr .....	365
pci_find_capability .....	366
pci_bus_find_capability .....	367
pci_find_ext_capability .....	368
pci_find_next_ht_capability .....	369
pci_find_ht_capability .....	370
pci_find_parent_resource .....	370
__pci_complete_power_transition .....	371
pci_set_power_state .....	372
pci_choose_state .....	373
pci_save_state .....	374
pci_restore_state .....	375
pci_store_saved_state .....	376
pci_load_saved_state .....	376
pci_load_and_free_saved_state .....	377
pci_reenable_device .....	378
pci_enable_device_io .....	379
pci_enable_device_mem .....	380
pci_enable_device .....	380
pcim_enable_device .....	381
pcim_pin_device .....	382
pci_disable_device .....	383
pci_set_pcie_reset_state .....	384
pci_pme_capable .....	385

pci_pme_active .....	385
__pci_enable_wake.....	386
pci_wake_from_d3 .....	388
pci_target_state .....	389
pci_prepare_to_sleep .....	390
pci_back_from_sleep .....	390
pci_dev_run_wake .....	391
pci_enable_ido .....	392
pci_disable_ido .....	393
pci_enable_obff.....	394
pci_disable_obff.....	395
pci_ltr_supported .....	396
pci_enable_ltr.....	397
pci_disable_ltr.....	397
pci_set_ltr.....	398
pci_release_region .....	399
pci_request_region.....	400
pci_request_region_exclusive .....	401
pci_release_selected_regions .....	402
pci_request_selected_regions .....	403
pci_release_regions.....	404
pci_request_regions .....	405
pci_request_regions_exclusive .....	406
pci_set_master.....	407
pci_clear_master .....	408
pci_set_cacheline_size.....	408
pci_set_mwi .....	409
pci_try_set_mwi.....	410
pci_clear_mwi.....	411
pci_intx .....	412
pci_msi_off .....	413
__pci_reset_function.....	414
pci_reset_function.....	415
pcix_get_max_mmrbc.....	416
pcix_get_mmrbc .....	417
pcix_set_mmrbc .....	417
pcie_get_readrq.....	418
pcie_set_readrq .....	419
pci_selectBars .....	420
pci_add_dynid.....	421
pci_match_id.....	423
__pci_register_driver .....	424
pci_unregister_driver .....	424

pci_dev_driver.....	425
pci_dev_get.....	426
pci_dev_put.....	427
pci_remove_bus_device.....	428
pci_remove_behind_bridge.....	429
pci_stop_bus_device.....	430
pci_find_bus.....	430
pci_find_next_bus.....	431
pci_get_slot.....	432
pci_get_domain_bus_and_slot.....	433
pci_get_subsys.....	434
pci_get_device.....	436
pci_get_class.....	437
pci_dev_present.....	438
pci_enable_msi_block.....	439
pci_enable_msix.....	440
pci_msi_enabled.....	441
pci_bus_alloc_resource.....	442
pci_bus_add_device.....	443
pci_bus_add_devices.....	444
pci_bus_set_ops.....	445
pci_read_vpd.....	445
pci_write_vpd.....	446
pci_vpd_truncate.....	447
pci_block_user_cfg_access.....	448
pci_unblock_user_cfg_access.....	449
pci_lost_interrupt.....	450
__ht_create_irq.....	451
ht_create_irq.....	452
ht_destroy_irq.....	453
pci_scan_slot.....	453
pci_rescan_bus.....	454
pci_create_slot.....	455
pci_renumber_slot.....	457
pci_destroy_slot.....	458
pci_hp_create_module_link.....	459
pci_hp_remove_module_link.....	459
pci_enable_rom.....	460
pci_disable_rom.....	461
pci_map_rom.....	462
pci_unmap_rom.....	463
pci_enable_sriov.....	464
pci_disable_sriov.....	465

pci_sriov_migration .....	465
pci_num_vf .....	466
pci_read_legacy_io .....	467
pci_write_legacy_io .....	468
pci_mmap_legacy_mem .....	470
pci_mmap_legacy_io .....	471
pci_adjust_legacy_attr .....	472
pci_create_legacy_files .....	473
pci_mmap_resource .....	473
pci_remove_resource_files .....	474
pci_create_resource_files .....	475
pci_write_rom .....	476
pci_read_rom .....	477
pci_remove_sysfs_dev_files .....	479
9.6. PCI Hotplug Support Library .....	479
__pci_hp_register .....	480
pci_hp_deregister .....	481
pci_hp_change_slot_info .....	482
9.7. MCA Architecture .....	483
9.7.1. MCA Device Functions .....	483
9.7.2. MCA Bus DMA .....	483
mca_enable_dma .....	483
mca_disable_dma .....	484
mca_set_dma_addr .....	485
mca_get_dma_addr .....	485
mca_set_dma_count .....	486
mca_get_dma_residue .....	487
mca_set_dma_io .....	488
mca_set_dma_mode .....	489
<b>10. Firmware Interfaces .....</b>	<b>491</b>
10.1. DMI Interfaces .....	491
dmi_check_system .....	491
dmi_first_match .....	492
dmi_get_system_info .....	492
dmi_name_in_vendors .....	493
dmi_find_device .....	494
dmi_get_date .....	495
dmi_walk .....	496
dmi_match .....	497
10.2. EDD Interfaces .....	498
edd_show_raw_data .....	498
edd_release .....	499
edd_dev_is_type .....	500

edd_get_pci_dev .....	501
edd_init .....	501
<b>11. Security Framework .....</b>	<b>503</b>
security_init.....	503
security_module_enable .....	503
register_security .....	504
securityfs_create_file .....	505
securityfs_create_dir.....	507
securityfs_remove .....	508
<b>12. Audit Interfaces.....</b>	<b>511</b>
audit_log_start.....	511
audit_log_format.....	512
audit_log_untrustedstring .....	513
audit_log_end.....	514
audit_log .....	514
audit_alloc.....	515
audit_free .....	516
audit_syscall_entry .....	517
audit_syscall_exit.....	518
__audit_getname.....	519
__audit_inode.....	520
auditsc_get_stamp.....	521
audit_set_loginuid.....	522
__audit_mq_open.....	523
__audit_mq_sendrecv .....	524
__audit_mq_notify .....	525
__audit_mq_getsetattr.....	526
__audit_ipc_obj.....	526
__audit_ipc_set_perm.....	527
audit_socketcall.....	528
__audit_fd_pair.....	529
audit_sockaddr .....	529
__audit_signal_info.....	530
__audit_log_bprm_fcaps.....	531
__audit_log_capset .....	532
audit_core_dumps.....	533
audit_receive_filter.....	534
<b>13. Accounting Framework.....</b>	<b>537</b>
sys_acct.....	537
acct_auto_close_mnt.....	537
acct_auto_close .....	538
acct_collect .....	539



acct_process .....	540
<b>14. Block Devices .....</b>	<b>543</b>
blk_get_backing_dev_info .....	543
blk_delay_queue .....	543
blk_start_queue .....	544
blk_stop_queue .....	545
blk_sync_queue .....	546
__blk_run_queue .....	547
blk_run_queue_async .....	548
blk_run_queue .....	549
blk_cleanup_queue .....	549
blk_init_queue .....	550
blk_make_request .....	552
blk_requeue_request .....	553
blk_insert_request .....	554
part_round_stats .....	555
blk_add_request_payload .....	556
submit_bio .....	557
blk_rq_check_limits .....	558
blk_insert_cloned_request .....	559
blk_rq_err_bytes .....	560
blk_peek_request .....	561
blk_start_request .....	562
blk_fetch_request .....	563
blk_update_request .....	564
blk_unprep_request .....	565
blk_end_request .....	566
blk_end_request_all .....	567
blk_end_request_cur .....	568
blk_end_request_err .....	569
__blk_end_request .....	570
__blk_end_request_all .....	571
__blk_end_request_cur .....	572
__blk_end_request_err .....	573
rq_flush_dcache_pages .....	574
blk_lld_busy .....	575
blk_rq_unprep_clone .....	576
blk_rq_prep_clone .....	577
__blk_run_queue_uncond .....	579
__blk_drain_queue .....	579
get_request .....	580
get_request_wait .....	581
attempt_plug_merge .....	582

__generic_make_request.....	584
blk_end_bidi_request.....	585
__blk_end_bidi_request.....	586
blk_rq_map_user.....	587
blk_rq_map_user_iov .....	589
blk_rq_unmap_user.....	590
blk_rq_map_kern .....	591
blk_release_queue.....	592
blk_queue_prep_rq .....	593
blk_queue_unprep_rq .....	594
blk_queue_merge_bvec .....	595
blk_set_default_limits.....	596
blk_set_stacking_limits .....	597
blk_queue_make_request.....	598
blk_queue_bounce_limit.....	599
blk_limits_max_hw_sectors .....	600
blk_queue_max_hw_sectors.....	601
blk_queue_max_discard_sectors .....	602
blk_queue_max_segments .....	603
blk_queue_max_segment_size .....	604
blk_queue_logical_block_size.....	604
blk_queue_physical_block_size .....	605
blk_queue_alignment_offset.....	606
blk_limits_io_min.....	607
blk_queue_io_min.....	608
blk_limits_io_opt.....	609
blk_queue_io_opt.....	610
blk_queue_stack_limits.....	611
blk_stack_limits .....	612
bdev_stack_limits .....	613
disk_stack_limits .....	614
blk_queue_dma_pad .....	615
blk_queue_update_dma_pad.....	616
blk_queue_dma_drain.....	617
blk_queue_segment_boundary .....	619
blk_queue_dma_alignment.....	619
blk_queue_update_dma_alignment .....	620
blk_queue_flush.....	621
blk_execute_rq_nowait .....	622
blk_execute_rq .....	624
blkdev_issue_flush.....	625
blkdev_issue_discard .....	626
blkdev_issue_zeroout.....	627

blk_queue_find_tag.....	628
blk_free_tags.....	629
blk_queue_free_tags .....	629
blk_init_tags.....	630
blk_queue_init_tags .....	631
blk_queue_resize_tags .....	632
blk_queue_end_tag .....	633
blk_queue_start_tag .....	634
blk_queue_invalidate_tags.....	635
__blk_free_tags.....	636
__blk_queue_free_tags .....	636
blk_rq_count_integrity_sg .....	637
blk_rq_map_integrity_sg .....	638
blk_integrity_compare .....	639
blk_integrity_register .....	640
blk_integrity_unregister .....	641
blk_trace_ioctl.....	642
blk_trace_shutdown .....	643
blk_add_trace_rq.....	644
blk_add_trace_bio.....	645
blk_add_trace_bio_remap.....	646
blk_add_trace_rq_remap.....	647
blk_mangle_minor .....	648
blk_alloc_devt.....	649
blk_free_devt .....	650
disk_replace_part_tbl.....	651
disk_expand_part_tbl .....	652
disk_block_events .....	653
disk_unblock_events .....	654
disk_clear_events .....	655
disk_get_part.....	656
disk_part_iter_init.....	657
disk_part_iter_next .....	658
disk_part_iter_exit .....	659
disk_map_sector_rcu .....	660
register_blkdev .....	661
add_disk .....	662
get_gendisk .....	663
bdget_disk.....	664
disk_check_events .....	665

<b>15. Char devices .....</b>	<b>667</b>
register_chrdev_region.....	667
alloc_chrdev_region.....	668
__register_chrdev .....	669
unregister_chrdev_region.....	670
__unregister_chrdev .....	671
cdev_add .....	672
cdev_del .....	673
cdev_alloc .....	674
cdev_init.....	675
<b>16. Miscellaneous Devices.....</b>	<b>677</b>
misc_register .....	677
misc_deregister .....	677
<b>17. Clock Framework .....</b>	<b>679</b>
clk_get.....	679
clk_enable .....	680
clk_disable .....	681
clk_get_rate.....	682
clk_put.....	683
clk_round_rate .....	683
clk_set_rate .....	684
clk_set_parent .....	685
clk_get_parent.....	686
clk_get_sys.....	687
clk_add_alias.....	688

# Chapter 1. Data Types

## 1.1. Doubly Linked Lists

### list\_add

**LINUX**

Kernel Hackers Manual June 2013

#### 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

**LINUX**

Kernel Hackers Manual June 2013

## 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\_del\_entry

**LINUX**

## Name

`__list_del_entry` — deletes entry from list.

## Synopsis

```
void __list_del_entry (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\_replace

## LINUX

## Name

`list_replace` — replace old entry by new one

## Synopsis

```
void list_replace (struct list_head * old, struct list_head *  
new);
```

## Arguments

*old*

the element to be replaced

*new*

the new element to insert

## Description

If *old* was empty, it will be overwritten.

## list\_del\_init

### LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

### 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\_is\_last

## LINUX

Kernel Hackers Manual June 2013

### Name

`list_is_last` — tests whether *list* is the last entry in list *head*

## Synopsis

```
int list_is_last (const struct list_head * list, const struct  
list_head * head);
```

## Arguments

*list*

the entry to test

*head*

the head of the list

## list\_empty

### LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_empty_careful` — tests whether a list is empty and not being modified

## Synopsis

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

## Arguments

*head*

the list to test

## Description

tests whether a list is empty \_and\_ checks that no other CPU might be in the process of modifying either member (next or prev)

## 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\_rotate\_left

### LINUX

Kernel Hackers Manual June 2013

### Name

`list_rotate_left` — rotate the list to the left

### Synopsis

```
void list_rotate_left (struct list_head * head);
```

### Arguments

*head*

the head of the list

## list\_is\_singular

### LINUX

## Name

`list_is_singular` — tests whether a list has just one entry.

## Synopsis

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

## Arguments

*head*

the list to test.

# list\_cut\_position

## LINUX

## Name

`list_cut_position` — cut a list into two

## Synopsis

```
void list_cut_position (struct list_head * list, struct  
list_head * head, struct list_head * entry);
```

## Arguments

*list*

a new list to add all removed entries

*head*

a list with entries

*entry*

an entry within head, could be the head itself and if so we won't cut the list

## Description

This helper moves the initial part of *head*, up to and including *entry*, from *head* to *list*. You should pass on *entry* an element you know is on *head*. *list* should be an empty list or a list you do not care about losing its data.

## list\_splice

### LINUX

Kernel Hackers Manual June 2013

### Name

`list_splice` — join two lists, this is designed for stacks

### Synopsis

```
void list_splice (const 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\_tail

### LINUX

Kernel Hackers Manual June 2013

## Name

`list_splice_tail` — join two lists, each list being a queue

## Synopsis

```
void list_splice_tail (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

## LINUX

Kernel Hackers Manual June 2013

### 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\_splice\_tail\_init

## LINUX

## Name

`list_splice_tail_init` — join two lists and reinitialise the emptied list

## Synopsis

```
void list_splice_tail_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

Each of the lists is a queue. The list at *list* is reinitialised

# list\_entry

## LINUX

## 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\_first\_entry

### LINUX

Kernel Hackers Manual June 2013

## Name

list\_first\_entry — get the first element from a list

## Synopsis

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

## Arguments

*ptr*

the list head to take the element from.

*type*

the type of the struct this is embedded in.

*member*

the name of the list\_struct within the struct.

## Description

Note, that list is expected to be not empty.

# list\_for\_each

## LINUX

Kernel Hackers Manual June 2013

## 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 cursor.

*head*

the head for your list.

## `__list_for_each`

### LINUX

Kernel Hackers Manual June 2013

## 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 cursor.

*head*

the head for your list.

## Description

This variant doesn't differ from `list_for_each` any more. We don't do prefetching in either case.

# list\_for\_each\_prev

## LINUX

Kernel Hackers Manual June 2013

## 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 cursor.

*head*

the head for your list.

# list\_for\_each\_safe

## LINUX

Kernel Hackers Manual June 2013

### 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 cursor.

*n*

another struct `list_head` to use as temporary storage

*head*

the head for your list.

# list\_for\_each\_prev\_safe

## LINUX

## Name

`list_for_each_prev_safe` — iterate over a list backwards safe against removal of list entry

## Synopsis

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

## Arguments

*pos*

the struct `list_head` to use as a loop cursor.

*n*

another struct `list_head` to use as temporary storage

*head*

the head for your list.

# list\_for\_each\_entry

## LINUX

## 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 cursor.

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## list\_for\_each\_entry\_reverse

### LINUX

Kernel Hackers Manual June 2013

## 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 cursor.

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## list\_prepare\_entry

**LINUX**

Kernel Hackers Manual June 2013

### Name

`list_prepare_entry` — prepare a pos entry for use in  
`list_for_each_entry_continue`

### 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

Prepares a pos entry for use as a start point in `list_for_each_entry_continue`.

# list\_for\_each\_entry\_continue

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_for_each_entry_continue` — continue iteration over list of given type

## Synopsis

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

## Arguments

*pos*

the type \* to use as a loop cursor.

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## Description

Continue to iterate over list of given type, continuing after the current position.

# list\_for\_each\_entry\_continue\_reverse

**LINUX**

Kernel Hackers Manual June 2013

## Name

`list_for_each_entry_continue_reverse` — iterate backwards from the given point

## Synopsis

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

## Arguments

*pos*

the type \* to use as a loop cursor.

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## Description

Start to iterate over list of given type backwards, continuing after the current position.

# list\_for\_each\_entry\_from

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_for_each_entry_from` — iterate over list of given type from the current point

## Synopsis

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

## Arguments

*pos*

the type \* to use as a loop cursor.

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## Description

Iterate over list of given type, continuing from current position.

# list\_for\_each\_entry\_safe

## LINUX

Kernel Hackers Manual June 2013

## 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 cursor.

*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\_entry\_safe\_continue

## LINUX

Kernel Hackers Manual June 2013

### Name

`list_for_each_entry_safe_continue` — continue list iteration safe against removal

### Synopsis

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

### Arguments

*pos*

the type \* to use as a loop cursor.

*n*

another type \* to use as temporary storage

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## Description

Iterate over list of given type, continuing after current point, safe against removal of list entry.

# list\_for\_each\_entry\_safe\_from

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_for_each_entry_safe_from` — iterate over list from current point safe against removal

## Synopsis

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

## Arguments

*pos*

the type \* to use as a loop cursor.

*n*

another type \* to use as temporary storage

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.



## Description

Iterate over list of given type from current point, safe against removal of list entry.

# list\_for\_each\_entry\_safe\_reverse

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_for_each_entry_safe_reverse` — iterate backwards over list safe against removal

## Synopsis

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

## Arguments

*pos*

the type \* to use as a loop cursor.

*n*

another type \* to use as temporary storage

*head*

the head for your list.

*member*

the name of the list\_struct within the struct.

## Description

Iterate backwards over list of given type, safe against removal of list entry.

# list\_safe\_reset\_next

## LINUX

Kernel Hackers Manual June 2013

## Name

`list_safe_reset_next` — reset a stale `list_for_each_entry_safe` loop

## Synopsis

```
list_safe_reset_next ( pos, n, member );
```

## Arguments

*pos*

the loop cursor used in the `list_for_each_entry_safe` loop

*n*

temporary storage used in `list_for_each_entry_safe`

*member*

the name of the `list_struct` within the struct.

## Description

`list_safe_reset_next` is not safe to use in general if the list may be modified concurrently (eg. the lock is dropped in the loop body). An exception to this is if the

cursor element (pos) is pinned in the list, and list\_safe\_reset\_next is called after re-taking the lock and before completing the current iteration of the loop body.

## hlist\_for\_each\_entry

### LINUX

Kernel Hackers Manual June 2013

### 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 cursor.

*pos*

the struct `hlist_node` to use as a loop cursor.

*head*

the head for your list.

*member*

the name of the `hlist_node` within the struct.

# hlist\_for\_each\_entry\_continue

## LINUX

Kernel Hackers Manual June 2013

### Name

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

### Synopsis

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

### Arguments

*tpos*

the type \* to use as a loop cursor.

*pos*

the struct `hlist_node` to use as a loop cursor.

*member*

the name of the `hlist_node` within the struct.

# hlist\_for\_each\_entry\_from

## LINUX

## Name

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

## Synopsis

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

## Arguments

*tpos*

the type \* to use as a loop cursor.

*pos*

the struct `hlist_node` to use as a loop cursor.

*member*

the name of the `hlist_node` within the struct.

# `hlist_for_each_entry_safe`

## LINUX

## 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 cursor.

*pos*

the struct hlist\_node to use as a loop cursor.

*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.

# Chapter 2. 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.

## 2.1. String Conversions

### simple\_strtoul

**LINUX**

Kernel Hackers Manual June 2013

#### 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

## simple\_strtoul

### LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

Kernel Hackers Manual June 2013

### 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\_strtoll

## LINUX

## 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

# vsnprintf

## LINUX

## 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

This function follows C99 `vsnprintf`, but has some extensions: `pS` output the name of a text symbol with offset `pS` output the name of a text symbol without offset `pF` output the name of a function pointer with its offset `pF` output the name of a function pointer without its offset `pB` output the name of a backtrace symbol with its offset `pR` output the address range in a struct resource with decoded flags `pR` output the address range in a struct resource with raw flags `pM` output a 6-byte MAC address with colons `pm` output a 6-byte MAC address without colons `pI4` print an IPv4 address without leading zeros `pi4` print an IPv4 address with leading zeros `pI6` print an IPv6 address with colons `pi6` print an IPv6 address without colons `pI6c` print an IPv6 address as specified by

## http

`//tools.ietf.org/html/draft-ietf-6man-text-addr-representation-00` `pU[bBIL]` print a UUID/GUID in big or little endian using lower or upper case. `n` is ignored

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 `vsnprintf`. If the return is greater than or equal to *size*, the resulting string is truncated.

If you're not already dealing with a `va_list` consider using `snprintf`.

## vsnprintf

### LINUX

Kernel Hackers Manual June 2013

### 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 have been written into the *buf* not including the trailing `'\0'`. If *size* is `== 0` the function returns 0.

If you're not already dealing with a `va_list` consider using `snprintf`.

See the `vsnprintf` documentation for format string extensions over C99.

# snprintf

## LINUX

Kernel Hackers Manual June 2013

## 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.

See the `vsnprintf` documentation for format string extensions over C99.

# scanf

## LINUX

Kernel Hackers Manual June 2013

## Name

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

## Synopsis

```
int scanf (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.

# vsprintf

**LINUX**

Kernel Hackers Manual June 2013

## 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.

If you're not already dealing with a `va_list` consider using `sprintf`.

See the `vsnprintf` documentation for format string extensions over C99.

# sprintf

## LINUX

Kernel Hackers Manual June 2013

## 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.

See the `vsnprintf` documentation for format string extensions over C99.

## vbin\_printf

### LINUX

Kernel Hackers Manual June 2013

### Name

`vbin_printf` — Parse a format string and place args' binary value in a buffer

### Synopsis

```
int vbin_printf (u32 * bin_buf, size_t size, const char * fmt,
va_list args);
```

## Arguments

*bin\_buf*

The buffer to place args' binary value

*size*

The size of the buffer (by words (32bits), not characters)

*fmt*

The format string to use

*args*

Arguments for the format string

## Description

The format follows C99 `vsnprintf`, except `n` is ignored, and its argument is skipped.

The return value is the number of words (32bits) which would be generated for the given input.

## NOTE

If the return value is greater than *size*, the resulting *bin\_buf* is NOT valid for `bstr_printf`.

# bstr\_printf

## LINUX

Kernel Hackers Manual June 2013

## Name

`bstr_printf` — Format a string from binary arguments and place it in a buffer

## Synopsis

```
int bstr_printf (char * buf, size_t size, const char * fmt,  
const u32 * bin_buf);
```

## 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

*bin\_buf*

Binary arguments for the format string

## Description

This function like C99 `vsnprintf`, but the difference is that `vsnprintf` gets arguments from stack, and `bstr_printf` gets arguments from *bin\_buf* which is a binary buffer that generated by `vbin_printf`.

The format follows C99 `vsnprintf`, but has some extensions: see `vsnprintf` comment for details.

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.

# bprintf

## LINUX

Kernel Hackers Manual June 2013

### Name

`bprintf` — Parse a format string and place args' binary value in a buffer

### Synopsis

```
int bprintf (u32 * bin_buf, size_t size, const char * fmt,  
...);
```

### Arguments

*bin\_buf*

The buffer to place args' binary value

*size*

The size of the buffer(by words(32bits), not characters)

*fmt*

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

...

variable arguments

### Description

The function returns the number of words(u32) written into *bin\_buf*.

# vsscanf

## LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

## 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

## 2.2. String Manipulation

### strnicmp

**LINUX**

## 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

### LINUX

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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.

In the case where the length of *src* is less than that of *count*, the remainder of *dest* will be padded with NUL.

# strncpy

## LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

### Name

`strcmp` — Compare two strings

### Synopsis

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

### Arguments

*cs*

One string

*ct*

Another string

# strncmp

## LINUX

Kernel Hackers Manual June 2013

### 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

### LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

**LINUX**

Kernel Hackers Manual June 2013

## 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

# skip\_spaces

**LINUX**



## Name

`skip_spaces` — Removes leading whitespace from *str*.

## Synopsis

```
char * skip_spaces (const char * str);
```

## Arguments

*str*

The string to be stripped.

## Description

Returns a pointer to the first non-whitespace character in *str*.

# strim

## LINUX

## Name

`strim` — Removes leading and trailing whitespace from *s*.

## Synopsis

```
char * strim (char * s);
```

## Arguments

*s*

The string to be stripped.

## Description

Note that the first trailing whitespace is replaced with a NUL-terminator in the given string *s*. Returns a pointer to the first non-whitespace character in *s*.

# strlen

## LINUX

Kernel Hackers Manual June 2013

## Name

`strlen` — Find the length of a string

## Synopsis

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

## Arguments

*s*

The string to be sized

## strnlen

### LINUX

Kernel Hackers Manual June 2013

## Name

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

## Synopsis

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

## Arguments

*s*

The string to be sized

*count*

The maximum number of bytes to search

# strspn

## LINUX

Kernel Hackers Manual June 2013

### Name

`strspn` — Calculate the length of the initial substring of *s* which only contain letters in *accept*

### Synopsis

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

### Arguments

*s*

The string to be searched

*accept*

The string to search for

# strcspn

## LINUX

Kernel Hackers Manual June 2013

### Name

`strcspn` — Calculate the length of the initial substring of *s* which does not contain letters in *reject*

## Synopsis

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

## Arguments

*s*

The string to be searched

*reject*

The string to avoid

## strpbrk

### LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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. :)

## sysfs\_streq

### LINUX

Kernel Hackers Manual June 2013

## Name

`sysfs_streq` — return true if strings are equal, modulo trailing newline

## Synopsis

```
bool sysfs_streq (const char * s1, const char * s2);
```

## Arguments

*s1*

one string

*s2*

another string

## Description

This routine returns true iff two strings are equal, treating both NUL and newline-then-NUL as equivalent string terminations. It's geared for use with `sysfs`

input strings, which generally terminate with newlines but are compared against values without newlines.

# strtobool

## LINUX

Kernel Hackers Manual June 2013

### Name

`strtobool` — convert common user inputs into boolean values

### Synopsis

```
int strtobool (const char * s, bool * res);
```

### Arguments

*s*

input string

*res*

result

### Description

This routine returns 0 iff the first character is one of 'Yy1Nn0'. Otherwise it will return -EINVAL. Value pointed to by *res* is updated upon finding a match.



# memset

## LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

Kernel Hackers Manual June 2013

### 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.

# memmove

## LINUX

Kernel Hackers Manual June 2013

### Name

`memmove` — Copy one area of memory to another

### Synopsis

```
void * memmove (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

## LINUX

## 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

### LINUX

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

## strnstr

### LINUX

Kernel Hackers Manual June 2013

## Name

`strnstr` — Find the first substring in a length-limited string

## Synopsis

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

## Arguments

*s1*

The string to be searched

*s2*

The string to search for

*len*

the maximum number of characters to search

## memchr

### LINUX

Kernel Hackers Manual June 2013

## 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

## 2.3. Bit Operations

### set\_bit

**LINUX**

Kernel Hackers Manual June 2013

## Name

`set_bit` — Atomically set a bit in memory

## Synopsis

```
void set_bit (unsigned 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`

### LINUX

Kernel Hackers Manual June 2013

## 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

**LINUX**

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

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

# test\_and\_set\_bit

## LINUX

Kernel Hackers Manual June 2013

### 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 also implies a memory barrier.

# test\_and\_set\_bit\_lock

## LINUX

Kernel Hackers Manual June 2013

### Name

`test_and_set_bit_lock` — Set a bit and return its old value for lock

### Synopsis

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

### Arguments

*nr*

Bit to set

*addr*

Address to count from

### Description

This is the same as `test_and_set_bit` on x86.

# \_\_test\_and\_set\_bit

## LINUX

## 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`

**LINUX**

## 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 also implies a memory barrier.

# \_\_test\_and\_clear\_bit

## LINUX

## 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

## LINUX

Kernel Hackers Manual June 2013

## 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

### LINUX

Kernel Hackers Manual June 2013

## Name

`test_bit` — Determine whether a bit is set

## Synopsis

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

## Arguments

*nr*

bit number to test

*addr*

Address to start counting from

## \_\_ffs

### LINUX

Kernel Hackers Manual June 2013

## Name

\_\_ffs — find first set 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.

## ffz

### LINUX

Kernel Hackers Manual June 2013

#### Name

`ffz` — find first zero bit 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

### LINUX

Kernel Hackers Manual June 2013

#### Name

`ffs` — find first set bit in word

## 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 other bitops.

ffs(value) returns 0 if value is 0 or the position of the first set bit if value is nonzero. The first (least significant) bit is at position 1.

## fls

### LINUX

Kernel Hackers Manual June 2013

## Name

fls — find last set bit in word

## Synopsis

```
int fls (int x);
```

## **Arguments**

`x`

the word to search

## **Description**

This is defined in a similar way as the libc and compiler builtin `ffs`, but returns the position of the most significant set bit.

`fls(value)` returns 0 if value is 0 or the position of the last set bit if value is nonzero. The last (most significant) bit is at position 32.

# Chapter 3. Basic Kernel Library Functions

The Linux kernel provides more basic utility functions.

## 3.1. Bitmap Operations

### `__bitmap_shift_right`

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`__bitmap_shift_right` — logical right shift of the bits in a bitmap

#### Synopsis

```
void __bitmap_shift_right (unsigned long * dst, const unsigned  
long * src, int shift, int bits);
```

#### Arguments

*dst*

destination bitmap

*src*

source bitmap

*shift*

shift by this many bits

*bits*

bitmap size, in bits

## Description

Shifting right (dividing) means moving bits in the MS -> LS bit direction. Zeros are fed into the vacated MS positions and the LS bits shifted off the bottom are lost.

# \_\_bitmap\_shift\_left

**LINUX**

Kernel Hackers Manual June 2013

## Name

`__bitmap_shift_left` — logical left shift of the bits in a bitmap

## Synopsis

```
void __bitmap_shift_left (unsigned long * dst, const unsigned  
long * src, int shift, int bits);
```

## Arguments

*dst*

destination bitmap

*src*

source bitmap



*shift*

shift by this many bits

*bits*

bitmap size, in bits

## Description

Shifting left (multiplying) means moving bits in the LS -> MS direction. Zeros are fed into the vacated LS bit positions and those MS bits shifted off the top are lost.

# bitmap\_scnprintf

**LINUX**

Kernel Hackers Manual June 2013

## Name

`bitmap_scnprintf` — convert bitmap to an ASCII hex string.

## Synopsis

```
int bitmap_scnprintf (char * buf, unsigned int buflen, const  
unsigned long * maskp, int nmaskbits);
```

## Arguments

*buf*

byte buffer into which string is placed

*buflen*

reserved size of *buf*, in bytes

*maskp*

pointer to bitmap to convert

*nmaskbits*

size of bitmap, in bits

## Description

Exactly *nmaskbits* bits are displayed. Hex digits are grouped into comma-separated sets of eight digits per set.

# \_\_bitmap\_parse

**LINUX**

Kernel Hackers Manual June 2013

## Name

`__bitmap_parse` — convert an ASCII hex string into a bitmap.

## Synopsis

```
int __bitmap_parse (const char * buf, unsigned int buflen, int  
is_user, unsigned long * maskp, int nmaskbits);
```

## Arguments

*buf*

pointer to buffer containing string.

*buflen*

buffer size in bytes. If string is smaller than this then it must be terminated with a `\0`.

*is\_user*

location of buffer, 0 indicates kernel space

*maskp*

pointer to bitmap array that will contain result.

*nmaskbits*

size of bitmap, in bits.

## Description

Commas group hex digits into chunks. Each chunk defines exactly 32 bits of the resultant bitmask. No chunk may specify a value larger than 32 bits (`-Eoverflow`), and if a chunk specifies a smaller value then leading 0-bits are prepended. `-EINVAL` is returned for illegal characters and for grouping errors such as “1,,5”, “,44”, “,” and “”. Leading and trailing whitespace accepted, but not embedded whitespace.

## bitmap\_parse\_user

**LINUX**

Kernel Hackers Manual June 2013

### Name

`bitmap_parse_user` — convert an ASCII hex string in a user buffer into a bitmap

## Synopsis

```
int bitmap_parse_user (const char __user * ubuf, unsigned int  
ulen, unsigned long * maskp, int nmaskbits);
```

## Arguments

*ubuf*

pointer to user buffer containing string.

*ulen*

buffer size in bytes. If string is smaller than this then it must be terminated with a `\0`.

*maskp*

pointer to bitmap array that will contain result.

*nmaskbits*

size of bitmap, in bits.

## Description

Wrapper for `__bitmap_parse`, providing it with user buffer.

We cannot have this as an inline function in `bitmap.h` because it needs `linux/uaccess.h` to get the `access_ok` declaration and this causes cyclic dependencies.

## bitmap\_scnlistprintf

**LINUX**

## Name

`bitmap_scnlistprintf` — convert bitmap to list format ASCII string

## Synopsis

```
int bitmap_scnlistprintf (char * buf, unsigned int buflen,  
const unsigned long * maskp, int nmaskbits);
```

## Arguments

*buf*

byte buffer into which string is placed

*buflen*

reserved size of *buf*, in bytes

*maskp*

pointer to bitmap to convert

*nmaskbits*

size of bitmap, in bits

## Description

Output format is a comma-separated list of decimal numbers and ranges.

Consecutively set bits are shown as two hyphen-separated decimal numbers, the smallest and largest bit numbers set in the range. Output format is compatible with the format accepted as input by `bitmap_parselist`.

The return value is the number of characters which would be generated for the given input, excluding the trailing `'\0'`, as per ISO C99.

# bitmap\_parselist\_user

## LINUX

Kernel Hackers Manual June 2013

### Name

`bitmap_parselist_user` —

### Synopsis

```
int bitmap_parselist_user (const char __user * ubuf, unsigned
int ulen, unsigned long * maskp, int nmaskbits);
```

### Arguments

*ubuf*

pointer to user buffer containing string.

*ulen*

buffer size in bytes. If string is smaller than this then it must be terminated with a `\0`.

*maskp*

pointer to bitmap array that will contain result.

*nmaskbits*

size of bitmap, in bits.

### Description

Wrapper for `bitmap_parselist`, providing it with user buffer.

We cannot have this as an inline function in `bitmap.h` because it needs `linux/uaccess.h` to get the `access_ok` declaration and this causes cyclic dependencies.

## bitmap\_remap

### LINUX

Kernel Hackers Manual June 2013

### Name

`bitmap_remap` — Apply map defined by a pair of bitmaps to another bitmap

### Synopsis

```
void bitmap_remap (unsigned long * dst, const unsigned long *  
src, const unsigned long * old, const unsigned long * new,  
int bits);
```

### Arguments

*dst*

remapped result

*src*

subset to be remapped

*old*

defines domain of map

*new*

defines range of map

*bits*

number of bits in each of these bitmaps

## Description

Let *old* and *new* define a mapping of bit positions, such that whatever position is held by the *n*-th set bit in *old* is mapped to the *n*-th set bit in *new*. In the more general case, allowing for the possibility that the weight 'w' of *new* is less than the weight of *old*, map the position of the *n*-th set bit in *old* to the position of the *m*-th set bit in *new*, where  $m == n \% w$ .

If either of the *old* and *new* bitmaps are empty, or if *src* and *dst* point to the same location, then this routine copies *src* to *dst*.

The positions of unset bits in *old* are mapped to themselves (the identify map).

Apply the above specified mapping to *src*, placing the result in *dst*, clearing any bits previously set in *dst*.

For example, lets say that *old* has bits 4 through 7 set, and *new* has bits 12 through 15 set. This defines the mapping of bit position 4 to 12, 5 to 13, 6 to 14 and 7 to 15, and of all other bit positions unchanged. So if say *src* comes into this routine with bits 1, 5 and 7 set, then *dst* should leave with bits 1, 13 and 15 set.

## bitmap\_bitremap

**LINUX**

Kernel Hackers Manual June 2013

### Name

`bitmap_bitremap` — Apply map defined by a pair of bitmaps to a single bit

### Synopsis

```
int bitmap_bitremap (int oldbit, const unsigned long * old,  
const unsigned long * new, int bits);
```



## Arguments

*oldbit*

bit position to be mapped

*old*

defines domain of map

*new*

defines range of map

*bits*

number of bits in each of these bitmaps

## Description

Let *old* and *new* define a mapping of bit positions, such that whatever position is held by the *n*-th set bit in *old* is mapped to the *n*-th set bit in *new*. In the more general case, allowing for the possibility that the weight 'w' of *new* is less than the weight of *old*, map the position of the *n*-th set bit in *old* to the position of the *m*-th set bit in *new*, where  $m == n \% w$ .

The positions of unset bits in *old* are mapped to themselves (the identify map).

Apply the above specified mapping to bit position *oldbit*, returning the new bit position.

For example, lets say that *old* has bits 4 through 7 set, and *new* has bits 12 through 15 set. This defines the mapping of bit position 4 to 12, 5 to 13, 6 to 14 and 7 to 15, and of all other bit positions unchanged. So if say *oldbit* is 5, then this routine returns 13.

## bitmap\_onto

**LINUX**

## Name

`bitmap_onto` — translate one bitmap relative to another

## Synopsis

```
void bitmap_onto (unsigned long * dst, const unsigned long *  
orig, const unsigned long * relmap, int bits);
```

## Arguments

*dst*

resulting translated bitmap

*orig*

original untranslated bitmap

*relmap*

bitmap relative to which translated

*bits*

number of bits in each of these bitmaps

## Description

Set the *n*-th bit of *dst* iff there exists some *m* such that the *n*-th bit of *relmap* is set, the *m*-th bit of *orig* is set, and the *n*-th bit of *relmap* is also the *m*-th `_set_` bit of *relmap*. (If you understood the previous sentence the first time you read it, you're overqualified for your current job.)

In other words, *orig* is mapped onto (surjectively) *dst*, using the the map { <*n*, *m*> | the *n*-th bit of *relmap* is the *m*-th set bit of *relmap* }.

Any set bits in *orig* above bit number *W*, where *W* is the weight of (number of set bits in) *relmap* are mapped nowhere. In particular, if for all bits *m* set in *orig*, *m*

$\geq W$ , then *dst* will end up empty. In situations where the possibility of such an empty result is not desired, one way to avoid it is to use the `bitmap_fold` operator, below, to first fold the *orig* bitmap over itself so that all its set bits *x* are in the range  $0 \leq x < W$ . The `bitmap_fold` operator does this by setting the bit  $(m \% W)$  in *dst*, for each bit (*m*) set in *orig*.

Example [1] for `bitmap_onto`: Let's say *relmap* has bits 30-39 set, and *orig* has bits 1, 3, 5, 7, 9 and 11 set. Then on return from this routine, *dst* will have bits 31, 33, 35, 37 and 39 set.

When bit 0 is set in *orig*, it means turn on the bit in *dst* corresponding to whatever is the first bit (if any) that is turned on in *relmap*. Since bit 0 was off in the above example, we leave off that bit (bit 30) in *dst*.

When bit 1 is set in *orig* (as in the above example), it means turn on the bit in *dst* corresponding to whatever is the second bit that is turned on in *relmap*. The second bit in *relmap* that was turned on in the above example was bit 31, so we turned on bit 31 in *dst*.

Similarly, we turned on bits 33, 35, 37 and 39 in *dst*, because they were the 4th, 6th, 8th and 10th set bits set in *relmap*, and the 4th, 6th, 8th and 10th bits of *orig* (i.e. bits 3, 5, 7 and 9) were also set.

When bit 11 is set in *orig*, it means turn on the bit in *dst* corresponding to whatever is the twelfth bit that is turned on in *relmap*. In the above example, there were only ten bits turned on in *relmap* (30..39), so that bit 11 was set in *orig* had no affect on *dst*.

Example [2] for `bitmap_fold + bitmap_onto`: Let's say *relmap* has these ten bits set: 40 41 42 43 45 48 53 61 74 95 (for the curious, that's 40 plus the first ten terms of the Fibonacci sequence.)

Further lets say we use the following code, invoking `bitmap_fold` then `bitmap_onto`, as suggested above to avoid the possitility of an empty *dst* result:

```
unsigned long *tmp; // a temporary bitmap's bits

bitmap_fold(tmp, orig, bitmap_weight(relmap, bits), bits); bitmap_onto(dst, tmp,
relmap, bits);
```

Then this table shows what various values of *dst* would be, for various *orig*'s. I list the zero-based positions of each set bit. The tmp column shows the intermediate result, as computed by using `bitmap_fold` to fold the *orig* bitmap modulo ten (the weight of *relmap*).

<i>orig</i>	<i>tmp</i>	<i>dst</i>	0 0 40 1 1 41 9 9 95 10 0 40 (*)	1 3 5 7 1 3 5 7 41 43 48 61 0 1 2 3 4
0 1 2 3 4 40 41 42 43 45 0 9 18 27 0 9 8 7 40 61 74 95 0 10 20 30 0 40 0 11 22 33 0				
1 2 3 40 41 42 43 0 12 24 36 0 2 4 6 40 42 45 53 78 102 211 1 2 8 41 42 74 (*)				

(\*) For these marked lines, if we hadn't first done `bitmap_fold` into `tmp`, then the `dst` result would have been empty.

If either of `orig` or `relmap` is empty (no set bits), then `dst` will be returned empty.

If (as explained above) the only set bits in `orig` are in positions `m` where `m >= W`, (where `W` is the weight of `relmap`) then `dst` will once again be returned empty.

All bits in `dst` not set by the above rule are cleared.

## bitmap\_fold

### LINUX

Kernel Hackers Manual June 2013

### Name

`bitmap_fold` — fold larger bitmap into smaller, modulo specified size

### Synopsis

```
void bitmap_fold (unsigned long * dst, const unsigned long *  
orig, int sz, int bits);
```

### Arguments

*dst*

resulting smaller bitmap

*orig*

original larger bitmap

*sz*

specified size

*bits*

number of bits in each of these bitmaps

## Description

For each bit *oldbit* in *orig*, set bit *oldbit mod sz* in *dst*. Clear all other bits in *dst*. See further the comment and Example [2] for `bitmap_onto` for why and how to use this.

# bitmap\_find\_free\_region

## LINUX

Kernel Hackers Manual June 2013

## Name

`bitmap_find_free_region` — find a contiguous aligned mem region

## Synopsis

```
int bitmap_find_free_region (unsigned long * bitmap, int bits,
int order);
```

## Arguments

*bitmap*

array of unsigned longs corresponding to the bitmap

*bits*

number of bits in the bitmap

*order*

region size (log base 2 of number of bits) to find

## Description

Find a region of free (zero) bits in a *bitmap* of *bits* bits and allocate them (set them to one). Only consider regions of length a power (*order*) of two, aligned to that power of two, which makes the search algorithm much faster.

Return the bit offset in bitmap of the allocated region, or `-errno` on failure.

# bitmap\_release\_region

**LINUX**

Kernel Hackers Manual June 2013

## Name

`bitmap_release_region` — release allocated bitmap region

## Synopsis

```
void bitmap_release_region (unsigned long * bitmap, int pos,  
int order);
```

## Arguments

*bitmap*

array of unsigned longs corresponding to the bitmap

*pos*

beginning of bit region to release

*order*

region size (log base 2 of number of bits) to release

## Description

This is the complement to `__bitmap_find_free_region` and releases the found region (by clearing it in the bitmap).

No return value.

# bitmap\_allocate\_region

## LINUX

Kernel Hackers Manual June 2013

## Name

`bitmap_allocate_region` — allocate bitmap region

## Synopsis

```
int bitmap_allocate_region (unsigned long * bitmap, int pos,  
int order);
```

## Arguments

*bitmap*

array of unsigned longs corresponding to the bitmap

*pos*

beginning of bit region to allocate

*order*

region size (log base 2 of number of bits) to allocate

## Description

Allocate (set bits in) a specified region of a bitmap.

Return 0 on success, or `-EBUSY` if specified region wasn't free (not all bits were zero).

# bitmap\_copy\_le

## LINUX

Kernel Hackers Manual June 2013

## Name

`bitmap_copy_le` — copy a bitmap, putting the bits into little-endian order.

## Synopsis

```
void bitmap_copy_le (void * dst, const unsigned long * src,  
int nbits);
```

## Arguments

*dst*

destination buffer

*src*

bitmap to copy



*nbits*

number of bits in the bitmap

## Description

Require `nbits % BITS_PER_LONG == 0`.

# \_\_bitmap\_parselist

## LINUX

Kernel Hackers Manual June 2013

## Name

`__bitmap_parselist` — convert list format ASCII string to bitmap

## Synopsis

```
int __bitmap_parselist (const char * buf, unsigned int buflen,  
int is_user, unsigned long * maskp, int nmaskbits);
```

## Arguments

*buf*

read nul-terminated user string from this buffer

*buflen*

buffer size in bytes. If string is smaller than this then it must be terminated with a `\0`.

*is\_user*

location of buffer, 0 indicates kernel space

*maskp*

write resulting mask here

*nmaskbits*

number of bits in mask to be written

## Description

Input format is a comma-separated list of decimal numbers and ranges. Consecutively set bits are shown as two hyphen-separated decimal numbers, the smallest and largest bit numbers set in the range.

Returns 0 on success, -errno on invalid input strings.

## Error values

-EINVAL: second number in range smaller than first -EINVAL: invalid character in string -ERANGE: bit number specified too large for mask

# bitmap\_pos\_to\_ord

**LINUX**

Kernel Hackers Manual June 2013

## Name

`bitmap_pos_to_ord` — find ordinal of set bit at given position in bitmap

## Synopsis

```
int bitmap_pos_to_ord (const unsigned long * buf, int pos, int bits);
```

## Arguments

*buf*

pointer to a bitmap

*pos*

a bit position in *buf* ( $0 \leq pos < bits$ )

*bits*

number of valid bit positions in *buf*

## Description

Map the bit at position *pos* in *buf* (of length *bits*) to the ordinal of which set bit it is. If it is not set or if *pos* is not a valid bit position, map to -1.

If for example, just bits 4 through 7 are set in *buf*, then *pos* values 4 through 7 will get mapped to 0 through 3, respectively, and other *pos* values will get mapped to 0. When *pos* value 7 gets mapped to (returns) *ord* value 3 in this example, that means that bit 7 is the 3rd (starting with 0th) set bit in *buf*.

The bit positions 0 through *bits* are valid positions in *buf*.

## bitmap\_ord\_to\_pos

**LINUX**

## Name

`bitmap_ord_to_pos` — find position of n-th set bit in bitmap

## Synopsis

```
int bitmap_ord_to_pos (const unsigned long * buf, int ord, int bits);
```

## Arguments

*buf*

pointer to bitmap

*ord*

ordinal bit position (n-th set bit,  $n \geq 0$ )

*bits*

number of valid bit positions in *buf*

## Description

Map the ordinal offset of bit *ord* in *buf* to its position in *buf*. Value of *ord* should be in range  $0 \leq ord < \text{weight}(\text{buf})$ , else results are undefined.

If for example, just bits 4 through 7 are set in *buf*, then *ord* values 0 through 3 will get mapped to 4 through 7, respectively, and all other *ord* values return undefined values. When *ord* value 3 gets mapped to (returns) *pos* value 7 in this example, that means that the 3rd set bit (starting with 0th) is at position 7 in *buf*.

The bit positions 0 through *bits* are valid positions in *buf*.

## 3.2. Command-line Parsing

### get\_option

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`get_option` — Parse integer from an option string

#### Synopsis

```
int get_option (char ** str, int * pint);
```

#### Arguments

*str*

option string

*pint*

(output) integer value parsed from *str*

#### Description

Read an int from an option string; if available accept a subsequent comma as well.

#### Return values

0 - no int in string 1 - int found, no subsequent comma 2 - int found including a subsequent comma 3 - hyphen found to denote a range

# get\_options

## LINUX

Kernel Hackers Manual June 2013

### Name

`get_options` — Parse a string into a list of integers

### Synopsis

```
char * get_options (const char * str, int nints, int * ints);
```

### Arguments

*str*

String to be parsed

*nints*

size of integer array

*ints*

integer array

### Description

This function parses a string containing a comma-separated list of integers, a hyphen-separated range of `_positive_` integers, or a combination of both. The parse halts when the array is full, or when no more numbers can be retrieved from the string.

Return value is the character in the string which caused the parse to end (typically a null terminator, if *str* is completely parseable).

# memparse

## LINUX

Kernel Hackers Manual June 2013

### Name

`memparse` — parse a string with mem suffixes into a number

### Synopsis

```
unsigned long long memparse (const char * ptr, char **  
retptr);
```

### Arguments

*ptr*

Where parse begins

*retptr*

(output) Optional pointer to next char after parse completes

### Description

Parses a string into a number. The number stored at *ptr* is potentially suffixed with *K* (for kilobytes, or 1024 bytes), *M* (for megabytes, or 1048576 bytes), or *G* (for gigabytes, or 1073741824). If the number is suffixed with *K*, *M*, or *G*, then the return value is the number multiplied by one kilobyte, one megabyte, or one gigabyte, respectively.

## 3.3. CRC Functions

### crc7

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`crc7` — update the CRC7 for the data buffer

#### Synopsis

```
u8 crc7 (u8 crc, const u8 * buffer, size_t len);
```

#### Arguments

*crc*

previous CRC7 value

*buffer*

data pointer

*len*

number of bytes in the buffer

#### Context

any



## Description

Returns the updated CRC7 value.

# crc16

## LINUX

Kernel Hackers Manual June 2013

## Name

`crc16` — compute the CRC-16 for the data buffer

## Synopsis

```
u16 crc16 (u16 crc, u8 const * buffer, size_t len);
```

## Arguments

*crc*

previous CRC value

*buffer*

data pointer

*len*

number of bytes in the buffer

## Description

Returns the updated CRC value.

# crc\_itu\_t

## LINUX

Kernel Hackers Manual June 2013

### Name

`crc_itu_t` — Compute the CRC-ITU-T for the data buffer

### Synopsis

```
u16 crc_itu_t (u16 crc, const u8 * buffer, size_t len);
```

### Arguments

*crc*

previous CRC value

*buffer*

data pointer

*len*

number of bytes in the buffer

### Description

Returns the updated CRC value

## **/usr/src/linux-3.0.76-0.11//lib/crc32.c**

### **Name**

/usr/src/linux-3.0.76-0.11//lib/crc32.c — Document generation inconsistency

### **Oops**

#### **Warning**

The template for this document tried to insert the structured comment from the file /usr/src/linux-3.0.76-0.11//lib/crc32.c at this point, but none was found. This dummy section is inserted to allow generation to continue.

## **crc\_ccitt**

### **LINUX**

Kernel Hackers Manual June 2013

### **Name**

crc\_ccitt — recompute the CRC for the data buffer

### **Synopsis**

```
u16 crc_ccitt (u16 crc, u8 const * buffer, size_t len);
```

## Arguments

*crc*

previous CRC value

*buffer*

data pointer

*len*

number of bytes in the buffer

## 3.4. idr/ida Functions

idr synchronization (stolen from radix-tree.h)

`idr_find` is able to be called locklessly, using RCU. The caller must ensure calls to this function are made within `rcu_read_lock` regions. Other readers (lock-free or otherwise) and modifications may be running concurrently.

It is still required that the caller manage the synchronization and lifetimes of the items. So if RCU lock-free lookups are used, typically this would mean that the items have their own locks, or are amenable to lock-free access; and that the items are freed by RCU (or only freed after having been deleted from the idr tree \*and\* a `synchronize_rcu` grace period).

IDA - IDR based ID allocator

This is id allocator without id -> pointer translation. Memory usage is much lower than full blown idr because each id only occupies a bit. ida uses a custom leaf node which contains `IDA_BITMAP_BITS` slots.

2007-04-25 written by Tejun Heo <htejun@gmail.com>

## idr\_pre\_get

**LINUX**

## Name

`idr_pre_get` — reserve resources for idr allocation

## Synopsis

```
int idr_pre_get (struct idr * idp, gfp_t gfp_mask);
```

## Arguments

*idp*

idr handle

*gfp\_mask*

memory allocation flags

## Description

This function should be called prior to calling the `idr_get_new*` functions. It preallocates enough memory to satisfy the worst possible allocation. The caller should pass in `GFP_KERNEL` if possible. This of course requires that no spinning locks be held.

If the system is REALLY out of memory this function returns 0, otherwise 1.

## `idr_get_new_above`

**LINUX**

## Name

`idr_get_new_above` — allocate new idr entry above or equal to a start id

## Synopsis

```
int idr_get_new_above (struct idr * idp, void * ptr, int
starting_id, int * id);
```

## Arguments

*idp*

idr handle

*ptr*

pointer you want associated with the id

*starting\_id*

id to start search at

*id*

pointer to the allocated handle

## Description

This is the allocate id function. It should be called with any required locks.

If allocation from IDR's private freelist fails, `idr_get_new_above` will return `-EAGAIN`. The caller should retry the `idr_pre_get` call to refill IDR's preallocation and then retry the `idr_get_new_above` call.

If the idr is full `idr_get_new_above` will return `-ENOSPC`.

*id* returns a value in the range *starting\_id* ... 0x7fffffff

# idr\_get\_new

## LINUX

Kernel Hackers Manual June 2013

### Name

`idr_get_new` — allocate new idr entry

### Synopsis

```
int idr_get_new (struct idr * idp, void * ptr, int * id);
```

### Arguments

*idp*

idr handle

*ptr*

pointer you want associated with the id

*id*

pointer to the allocated handle

### Description

If allocation from IDR's private freelist fails, `idr_get_new_above` will return `-EAGAIN`. The caller should retry the `idr_pre_get` call to refill IDR's preallocation and then retry the `idr_get_new_above` call.

If the idr is full `idr_get_new_above` will return `-ENOSPC`.

*id* returns a value in the range 0 ... 0x7fffffff

# idr\_remove

## LINUX

Kernel Hackers Manual June 2013

### Name

`idr_remove` — remove the given id and free its slot

### Synopsis

```
void idr_remove (struct idr * idp, int id);
```

### Arguments

*idp*

idr handle

*id*

unique key

# idr\_remove\_all

## LINUX

Kernel Hackers Manual June 2013

### Name

`idr_remove_all` — remove all ids from the given idr tree



## Synopsis

```
void idr_remove_all (struct idr * idp);
```

## Arguments

*idp*

idr handle

## Description

`idr_destroy` only frees up unused, cached `idp_layers`, but this function will remove all id mappings and leave all `idp_layers` unused.

A typical clean-up sequence for objects stored in an idr tree will use `idr_for_each` to free all objects, if necessary, then `idr_remove_all` to remove all ids, and `idr_destroy` to free up the cached `idr_layers`.

## idr\_destroy

### LINUX

Kernel Hackers Manual June 2013

## Name

`idr_destroy` — release all cached layers within an idr tree

## Synopsis

```
void idr_destroy (struct idr * idp);
```

## Arguments

*idp*

idr handle

## idr\_find

### LINUX

Kernel Hackers Manual June 2013

## Name

`idr_find` — return pointer for given id

## Synopsis

```
void * idr_find (struct idr * idp, int id);
```

## Arguments

*idp*

idr handle

*id*

lookup key

## Description

Return the pointer given the id it has been registered with. A `NULL` return indicates that *id* is not valid or you passed `NULL` in `idr_get_new`.

This function can be called under `rcu_read_lock`, given that the leaf pointers lifetimes are correctly managed.

## idr\_for\_each

### LINUX

Kernel Hackers Manual June 2013

### Name

`idr_for_each` — iterate through all stored pointers

### Synopsis

```
int idr_for_each (struct idr * idp, int (*fn) (int id, void  
*p, void *data), void * data);
```

### Arguments

*idp*

idr handle

*fn*

function to be called for each pointer

*data*

data passed back to callback function

## Description

Iterate over the pointers registered with the given `idr`. The callback function will be called for each pointer currently registered, passing the `id`, the pointer and the data pointer passed to this function. It is not safe to modify the `idr` tree while in the callback, so functions such as `idr_get_new` and `idr_remove` are not allowed.

We check the return of `fn` each time. If it returns anything other than 0, we break out and return that value.

The caller must serialize `idr_for_each` vs `idr_get_new` and `idr_remove`.

## idr\_get\_next

### LINUX

Kernel Hackers Manual June 2013

### Name

`idr_get_next` — lookup next object of `id` to given `id`.

### Synopsis

```
void * idr_get_next (struct idr * idp, int * nextidp);
```

### Arguments

*idp*

idr handle

*nextidp*

pointer to lookup key

## Description

Returns pointer to registered object with id, which is next number to given id. After being looked up, *\*nextidp* will be updated for the next iteration.

## idr\_replace

### LINUX

Kernel Hackers Manual June 2013

## Name

`idr_replace` — replace pointer for given id

## Synopsis

```
void * idr_replace (struct idr * idp, void * ptr, int id);
```

## Arguments

*idp*

idr handle

*ptr*

pointer you want associated with the id

*id*

lookup key

## Description

Replace the pointer registered with an *id* and return the old value. A `-ENOENT` return indicates that *id* was not found. A `-EINVAL` return indicates that *id* was not within valid constraints.

The caller must serialize with writers.

## idr\_init

### LINUX

Kernel Hackers Manual June 2013

## Name

`idr_init` — initialize idr handle

## Synopsis

```
void idr_init (struct idr * idp);
```

## Arguments

*idp*

idr handle

## Description

This function is use to set up the handle (*idp*) that you will pass to the rest of the functions.

# ida\_pre\_get

## LINUX

Kernel Hackers Manual June 2013

### Name

`ida_pre_get` — reserve resources for ida allocation

### Synopsis

```
int ida_pre_get (struct ida * ida, gfp_t gfp_mask);
```

### Arguments

*ida*

ida handle

*gfp\_mask*

memory allocation flag

### Description

This function should be called prior to locking and calling the following function. It preallocates enough memory to satisfy the worst possible allocation.

If the system is REALLY out of memory this function returns 0, otherwise 1.

# ida\_get\_new\_above

## LINUX

## Name

`ida_get_new_above` — allocate new ID above or equal to a start id

## Synopsis

```
int ida_get_new_above (struct ida * ida, int starting_id, int  
* p_id);
```

## Arguments

*ida*

ida handle

*starting\_id*

id to start search at

*p\_id*

pointer to the allocated handle

## Description

Allocate new ID above or equal to *ida*. It should be called with any required locks.

If memory is required, it will return `-EAGAIN`, you should unlock and go back to the `ida_pre_get` call. If the ida is full, it will return `-ENOSPC`.

*p\_id* returns a value in the range *starting\_id* ... 0x7fffffff.



# ida\_get\_new

## LINUX

Kernel Hackers Manual June 2013

### Name

`ida_get_new` — allocate new ID

### Synopsis

```
int ida_get_new (struct ida * ida, int * p_id);
```

### Arguments

*ida*

idr handle

*p\_id*

pointer to the allocated handle

### Description

Allocate new ID. It should be called with any required locks.

If memory is required, it will return `-EAGAIN`, you should unlock and go back to the `idr_pre_get` call. If the idr is full, it will return `-ENOSPC`.

*id* returns a value in the range 0 ... 0x7fffffff.

# ida\_remove

## LINUX

Kernel Hackers Manual June 2013

### Name

`ida_remove` — remove the given ID

### Synopsis

```
void ida_remove (struct ida * ida, int id);
```

### Arguments

*ida*

ida handle

*id*

ID to free

# ida\_destroy

## LINUX

Kernel Hackers Manual June 2013

### Name

`ida_destroy` — release all cached layers within an ida tree

## Synopsis

```
void ida_destroy (struct ida * ida);
```

## Arguments

*ida*

ida handle

## ida\_simple\_get

### LINUX

Kernel Hackers Manual June 2013

## Name

`ida_simple_get` — get a new id.

## Synopsis

```
int ida_simple_get (struct ida * ida, unsigned int start,  
unsigned int end, gfp_t gfp_mask);
```

## Arguments

*ida*

the (initialized) ida.

*start*

the minimum id (inclusive, < 0x8000000)

*end*

the maximum id (exclusive, < 0x8000000 or 0)

*gfp\_mask*

memory allocation flags

## Description

Allocates an id in the range `start <= id < end`, or returns `-ENOSPC`. On memory allocation failure, returns `-ENOMEM`.

Use `ida_simple_remove` to get rid of an id.

# ida\_simple\_remove

## LINUX

Kernel Hackers Manual June 2013

## Name

`ida_simple_remove` — remove an allocated id.

## Synopsis

```
void ida_simple_remove (struct ida * ida, unsigned int id);
```

## Arguments

*ida*

the (initialized) ida.

*id*

the id returned by `ida_simple_get`.

## ida\_init

### LINUX

Kernel Hackers Manual June 2013

## Name

`ida_init` — initialize ida handle

## Synopsis

```
void ida_init (struct ida * ida);
```

## Arguments

*ida*

ida handle

## Description

This function is use to set up the handle (*ida*) that you will pass to the rest of the functions.



# Chapter 4. Memory Management in Linux

## 4.1. The Slab Cache

### kcalloc

#### LINUX

Kernel Hackers Manual June 2013

#### Name

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

#### Synopsis

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

#### Arguments

*n*

number of elements.

*size*

element size.

*flags*

the type of memory to allocate.

## Description

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. May use emergency pools. For example, use this inside interrupt handlers.

GFP\_HIGHUSER - Allocate pages from high memory.

GFP\_NOIO - Do not do any I/O at all while trying to get memory.

GFP\_NOFS - Do not make any fs calls while trying to get memory.

GFP\_NOWAIT - Allocation will not sleep.

GFP\_THISNODE - Allocate node-local memory only.

GFP\_DMA - Allocation suitable for DMA. Should only be used for `kmalloc` caches. Otherwise, use a slab created with `SLAB_DMA`.

Also it is possible to set different flags by OR'ing in one or more of the following additional *flags*:

\_\_GFP\_COLD - Request cache-cold pages instead of trying to return cache-warm pages.

\_\_GFP\_HIGH - This allocation has high priority and may use emergency pools.

\_\_GFP\_NOFAIL - Indicate that this allocation is in no way allowed to fail (think twice before using).

\_\_GFP\_NORETRY - If memory is not immediately available, then give up at once.

\_\_GFP\_NOWARN - If allocation fails, don't issue any warnings.

\_\_GFP\_REPEAT - If allocation fails initially, try once more before failing.

There are other flags available as well, but these are not intended for general use, and so are not documented here. For a full list of potential flags, always refer to `linux/gfp.h`.

## kmalloc\_node

**LINUX**



## Name

`kmalloc_node` — allocate memory from a specific node

## Synopsis

```
void * kmalloc_node (size_t size, gfp_t flags, int node);
```

## Arguments

*size*

how many bytes of memory are required.

*flags*

the type of memory to allocate (see `kcalloc`).

*node*

node to allocate from.

## Description

`kmalloc` for non-local nodes, used to allocate from a specific node if available. Equivalent to `kmalloc` in the non-NUMA single-node case.

## kzalloc

**LINUX**

## Name

`kzalloc` — allocate memory. The memory is set to zero.

## Synopsis

```
void * kzalloc (size_t size, gfp_t flags);
```

## Arguments

*size*

how many bytes of memory are required.

*flags*

the type of memory to allocate (see `kmalloc`).

## `kzalloc_node`

### LINUX

## Name

`kzalloc_node` — allocate zeroed memory from a particular memory node.

## Synopsis

```
void * kzalloc_node (size_t size, gfp_t flags, int node);
```

## Arguments

*size*

how many bytes of memory are required.

*flags*

the type of memory to allocate (see `kmalloc`).

*node*

memory node from which to allocate

## `kmem_cache_create`

**LINUX**

Kernel Hackers Manual June 2013

### Name

`kmem_cache_create` — Create a cache.

### Synopsis

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

## 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.

## 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.

*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\_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

## LINUX

Kernel Hackers Manual June 2013

### Name

`kmem_cache_shrink` — Shrink a cache.

### Synopsis

```
int kmem_cache_shrink (struct kmem_cache * 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

## LINUX

## Name

`kmem_cache_destroy` — delete a cache

## Synopsis

```
void kmem_cache_destroy (struct kmem_cache * cachep);
```

## Arguments

*cachep*

the cache to destroy

## Description

Remove a struct `kmem_cache` object from the slab cache.

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 no one will allocate memory from the cache during the `kmem_cache_destroy`.

## `kmem_cache_alloc`

**LINUX**

## Name

`kmem_cache_alloc` — Allocate an object

## Synopsis

```
void * kmem_cache_alloc (struct kmem_cache * cachep, gfp_t  
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_free`

**LINUX**

## Name

`kmem_cache_free` — Deallocate an object

## Synopsis

```
void kmem_cache_free (struct kmem_cache * 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.

# kfree

## LINUX

## Name

`kfree` — free previously allocated memory



## Synopsis

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

## Arguments

*objp*

pointer returned by `kmalloc`.

## Description

If *objp* is `NULL`, no operation is performed.

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

## ksize

### LINUX

Kernel Hackers Manual June 2013

## Name

`ksize` — get the actual amount of memory allocated for a given object

## Synopsis

```
size_t ksize (const void * objp);
```

## Arguments

*objp*

Pointer to the object

## Description

`kmalloc` may internally round up allocations and return more memory than requested. `ksize` can be used to determine the actual amount of memory allocated. The caller may use this additional memory, even though a smaller amount of memory was initially specified with the `kmalloc` call. The caller must guarantee that `objp` points to a valid object previously allocated with either `kmalloc` or `kmem_cache_alloc`. The object must not be freed during the duration of the call.

## 4.2. User Space Memory Access

### `__copy_to_user_inatomic`

**LINUX**

Kernel Hackers Manual June 2013

### 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.

## Description

Copy data from kernel space to user space. Caller must check the specified block with `access_ok` before calling this function. The caller should also make sure he pins the user space address so that we don't result in page fault and sleep.

Here we special-case 1, 2 and 4-byte `copy_*_user` invocations. On a fault we return the initial request size (1, 2 or 4), as `copy_*_user` should do. If a store crosses a page boundary and gets a fault, the x86 will not write anything, so this is accurate.

## `__copy_to_user`

### LINUX

Kernel Hackers Manual June 2013

## Name

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

## Synopsis

```
unsigned long __must_check __copy_to_user (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**

**LINUX**

## Name

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

## Synopsis

```
unsigned long __copy_from_user (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.

An alternate version - `__copy_from_user_inatomic` - may be called from atomic context and will fail rather than sleep. In this case the uncopied bytes will \*NOT\* be padded with zeros. See `fs/filemap.h` for explanation of why this is needed.

## strlen\_user

### LINUX

Kernel Hackers Manual June 2013

### 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**

### **LINUX**

Kernel Hackers Manual June 2013

### **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

### LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

## 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**

### **LINUX**

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

## 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*.

## copy\_to\_user

### LINUX

Kernel Hackers Manual June 2013

### Name

`copy_to_user` — Copy a block of data into user space.

### Synopsis

```
unsigned long copy_to_user (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.

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

# **`_copy_from_user`**

## **LINUX**

Kernel Hackers Manual June 2013

## Name

`_copy_from_user` — Copy a block of data from user space.

## Synopsis

```
unsigned long _copy_from_user (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.

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.

## 4.3. More Memory Management Functions

### read\_cache\_pages

#### LINUX

Kernel Hackers Manual June 2013

## Name

`read_cache_pages` — populate an address space with some pages & start reads against them

## Synopsis

```
int read_cache_pages (struct address_space * mapping, struct  
list_head * pages, int (*filler) (void *, struct page *), void  
* data);
```

## Arguments

*mapping*

the address\_space

*pages*

The address of a list\_head which contains the target pages. These pages have their ->index populated and are otherwise uninitialised.

*filler*

callback routine for filling a single page.

*data*

private data for the callback routine.

## Description

Hides the details of the LRU cache etc from the filesystems.

# page\_cache\_sync\_readahead

## LINUX

Kernel Hackers Manual June 2013

## Name

page\_cache\_sync\_readahead — generic file readahead

## Synopsis

```
void page_cache_sync_readahead (struct address_space *  
mapping, struct file_ra_state * ra, struct file * filp,  
pgoff_t offset, unsigned long req_size);
```

## Arguments

*mapping*

address\_space which holds the pagecache and I/O vectors

*ra*

file\_ra\_state which holds the readahead state

*filp*

passed on to ->readpage and ->readpages

*offset*

start offset into *mapping*, in pagecache page-sized units

*req\_size*

hint: total size of the read which the caller is performing in pagecache pages

## Description

page\_cache\_sync\_readahead should be called when a cache miss happened: it will submit the read. The readahead logic may decide to piggyback more pages onto the read request if access patterns suggest it will improve performance.

# page\_cache\_async\_readahead

**LINUX**

Kernel Hackers Manual June 2013

## Name

page\_cache\_async\_readahead — file readahead for marked pages



## Synopsis

```
void page_cache_async_readahead (struct address_space *  
mapping, struct file_ra_state * ra, struct file * filp, struct  
page * page, pgoff_t offset, unsigned long req_size);
```

## Arguments

*mapping*

address\_space which holds the pagecache and I/O vectors

*ra*

file\_ra\_state which holds the readahead state

*filp*

passed on to ->readpage and ->readpages

*page*

the page at *offset* which has the PG\_readahead flag set

*offset*

start offset into *mapping*, in pagecache page-sized units

*req\_size*

hint: total size of the read which the caller is performing in pagecache pages

## Description

page\_cache\_async\_readahead should be called when a page is used which has the PG\_readahead flag; this is a marker to suggest that the application has used up enough of the readahead window that we should start pulling in more pages.

# delete\_from\_page\_cache

## LINUX

Kernel Hackers Manual June 2013

### Name

`delete_from_page_cache` — delete page from page cache

### Synopsis

```
void delete_from_page_cache (struct page * page);
```

### Arguments

*page*

the page which the kernel is trying to remove from page cache

### Description

This must be called only on pages that have been verified to be in the page cache and locked. It will never put the page into the free list, the caller has a reference on the page.

# filemap\_flush

## LINUX

## Name

`filemap_flush` — mostly a non-blocking flush

## Synopsis

```
int filemap_flush (struct address_space * mapping);
```

## Arguments

*mapping*

target address\_space

## Description

This is a mostly non-blocking flush. Not suitable for data-integrity purposes - I/O may not be started against all dirty pages.

# filemap\_fdatawait\_range

## LINUX

## Name

`filemap_fdatawait_range` — wait for writeback to complete

## Synopsis

```
int filemap_fdatawait_range (struct address_space * mapping,  
loff_t start_byte, loff_t end_byte);
```

## Arguments

*mapping*

address space structure to wait for

*start\_byte*

offset in bytes where the range starts

*end\_byte*

offset in bytes where the range ends (inclusive)

## Description

Walk the list of under-writeback pages of the given address space in the given range and wait for all of them.

# filemap\_fdatawait

**LINUX**

Kernel Hackers Manual June 2013

## Name

`filemap_fdatawait` — wait for all under-writeback pages to complete

## Synopsis

```
int filemap_fdatawait (struct address_space * mapping);
```

## Arguments

*mapping*

address space structure to wait for

## Description

Walk the list of under-writeback pages of the given address space and wait for all of them.

# filemap\_write\_and\_wait\_range

## LINUX

Kernel Hackers Manual June 2013

## Name

`filemap_write_and_wait_range` — write out & wait on a file range

## Synopsis

```
int filemap_write_and_wait_range (struct address_space *  
mapping, loff_t lstart, loff_t lend);
```

## Arguments

*mapping*

the address\_space for the pages

*lstart*

offset in bytes where the range starts

*lend*

offset in bytes where the range ends (inclusive)

## Description

Write out and wait upon file offsets lstart->lend, inclusive.

Note that ‘lend’ is inclusive (describes the last byte to be written) so that this function can be used to write to the very end-of-file (end = -1).

# replace\_page\_cache\_page

## LINUX

Kernel Hackers Manual June 2013

## Name

replace\_page\_cache\_page — replace a pagecache page with a new one

## Synopsis

```
int replace_page_cache_page (struct page * old, struct page *  
new, gfp_t gfp_mask);
```

## Arguments

*old*

page to be replaced

*new*

page to replace with

*gfp\_mask*

allocation mode

## Description

This function replaces a page in the pagecache with a new one. On success it acquires the pagecache reference for the new page and drops it for the old page. Both the old and new pages must be locked. This function does not add the new page to the LRU, the caller must do that.

The remove + add is atomic. The only way this function can fail is memory allocation failure.

# add\_to\_page\_cache\_locked

## LINUX

Kernel Hackers Manual June 2013

## Name

`add_to_page_cache_locked` — add a locked page to the pagecache

## Synopsis

```
int add_to_page_cache_locked (struct page * page, struct
address_space * mapping, pgoff_t offset, gfp_t gfp_mask);
```

## Arguments

*page*

page to add

*mapping*

the page's address\_space

*offset*

page index

*gfp\_mask*

page allocation mode

## Description

This function is used to add a page to the pagecache. It must be locked. This function does not add the page to the LRU. The caller must do that.

# add\_page\_wait\_queue

**LINUX**

Kernel Hackers Manual June 2013

## Name

add\_page\_wait\_queue — Add an arbitrary waiter to a page's wait queue



## Synopsis

```
void add_page_wait_queue (struct page * page, wait_queue_t *  
waiter);
```

## Arguments

*page*

Page defining the wait queue of interest

*waiter*

Waiter to add to the queue

## Description

Add an arbitrary *waiter* to the wait queue for the nominated *page*.

# unlock\_page

**LINUX**

Kernel Hackers Manual June 2013

## Name

unlock\_page — unlock a locked page

## Synopsis

```
void unlock_page (struct page * page);
```

## Arguments

*page*

the page

## Description

Unlocks the page and wakes up sleepers in `__wait_on_page_locked`. Also wakes sleepers in `wait_on_page_writeback` because the wakeup mechanism between PageLocked pages and PageWriteback pages is shared. But that's OK - sleepers in `wait_on_page_writeback` just go back to sleep.

The mb is necessary to enforce ordering between the `clear_bit` and the read of the waitqueue (to avoid SMP races with a parallel `wait_on_page_locked`).

## end\_page\_writeback

### LINUX

Kernel Hackers Manual June 2013

## Name

`end_page_writeback` — end writeback against a page

## Synopsis

```
void end_page_writeback (struct page * page);
```

## Arguments

*page*

the page

## \_\_lock\_page

**LINUX**

Kernel Hackers Manual June 2013

## Name

`__lock_page` — get a lock on the page, assuming we need to sleep to get it

## Synopsis

```
void __lock_page (struct page * page);
```

## Arguments

*page*

the page to lock

## find\_get\_page

**LINUX**

## Name

`find_get_page` — find and get a page reference

## Synopsis

```
struct page * find_get_page (struct address_space * mapping,  
pgoff_t offset);
```

## Arguments

*mapping*

the `address_space` to search

*offset*

the page index

## Description

Is there a pagecache struct page at the given (mapping, offset) tuple? If yes, increment its refcount and return it; if no, return NULL.

## `find_lock_page`

**LINUX**

## Name

`find_lock_page` — locate, pin and lock a pagecache page

## Synopsis

```
struct page * find_lock_page (struct address_space * mapping,  
pgoff_t offset);
```

## Arguments

*mapping*

the `address_space` to search

*offset*

the page index

## Description

Locates the desired pagecache page, locks it, increments its reference count and returns its address.

Returns zero if the page was not present. `find_lock_page` may sleep.

## `find_or_create_page`

**LINUX**

## Name

`find_or_create_page` — locate or add a pagecache page

## Synopsis

```
struct page * find_or_create_page (struct address_space *  
mapping, pgoff_t index, gfp_t gfp_mask);
```

## Arguments

*mapping*

the page's `address_space`

*index*

the page's index into the mapping

*gfp\_mask*

page allocation mode

## Description

Locates a page in the pagecache. If the page is not present, a new page is allocated using *gfp\_mask* and is added to the pagecache and to the VM's LRU list. The returned page is locked and has its reference count incremented.

`find_or_create_page` may sleep, even if *gfp\_flags* specifies an atomic allocation!

`find_or_create_page` returns the desired page's address, or zero on memory exhaustion.

# find\_get\_pages\_contig

## LINUX

Kernel Hackers Manual June 2013

### Name

`find_get_pages_contig` — gang contiguous pagecache lookup

### Synopsis

```
unsigned find_get_pages_contig (struct address_space *  
mapping, pgoff_t index, unsigned int nr_pages, struct page **  
pages);
```

### Arguments

*mapping*

The `address_space` to search

*index*

The starting page index

*nr\_pages*

The maximum number of pages

*pages*

Where the resulting pages are placed

### Description

`find_get_pages_contig` works exactly like `find_get_pages`, except that the returned number of pages are guaranteed to be contiguous.

`find_get_pages_contig` returns the number of pages which were found.

# find\_get\_pages\_tag

## LINUX

Kernel Hackers Manual June 2013

### Name

`find_get_pages_tag` — find and return pages that match *tag*

### Synopsis

```
unsigned find_get_pages_tag (struct address_space * mapping,  
pgoff_t * index, int tag, unsigned int nr_pages, struct page  
** pages);
```

### Arguments

*mapping*

the `address_space` to search

*index*

the starting page index

*tag*

the tag index

*nr\_pages*

the maximum number of pages

*pages*

where the resulting pages are placed



## Description

Like `find_get_pages`, except we only return pages which are tagged with `tag`. We update `index` to index the next page for the traversal.

# grab\_cache\_page\_nowait

## LINUX

Kernel Hackers Manual June 2013

## Name

`grab_cache_page_nowait` — returns locked page at given index in given cache

## Synopsis

```
struct page * grab_cache_page_nowait (struct address_space *
mapping, pgoff_t index);
```

## Arguments

*mapping*

target address\_space

*index*

the page index

## Description

Same as `grab_cache_page`, but do not wait if the page is unavailable. This is intended for speculative data generators, where the data can be regenerated if the

page couldn't be grabbed. This routine should be safe to call while holding the lock for another page.

Clear `__GFP_FS` when allocating the page to avoid recursion into the fs and deadlock against the caller's locked page.

## generic\_file\_aio\_read

### LINUX

Kernel Hackers Manual June 2013

### Name

`generic_file_aio_read` — generic filesystem read routine

### Synopsis

```
ssize_t generic_file_aio_read (struct kiocb * iocb, const
struct iovec * iov, unsigned long nr_segs, loff_t pos);
```

### Arguments

*iocb*

kernel I/O control block

*iov*

io vector request

*nr\_segs*

number of segments in the iovec

*pos*

current file position

## Description

This is the “`read`” routine for all filesystems that can use the page cache directly.

# filemap\_fault

## LINUX

Kernel Hackers Manual June 2013

## Name

`filemap_fault` — read in file data for page fault handling

## Synopsis

```
int filemap_fault (struct vm_area_struct * vma, struct
vm_fault * vmf);
```

## Arguments

*vma*

vma in which the fault was taken

*vmf*

struct `vm_fault` containing details of the fault

## Description

`filemap_fault` is invoked via the vma operations vector for a mapped memory region to read in file data during a page fault.

The goto's are kind of ugly, but this streamlines the normal case of having it in the page cache, and handles the special cases reasonably without having a lot of duplicated code.

## read\_cache\_page\_async

### LINUX

Kernel Hackers Manual June 2013

### Name

`read_cache_page_async` — read into page cache, fill it if needed

### Synopsis

```
struct page * read_cache_page_async (struct address_space *  
mapping, pgoff_t index, int (*filler) (void *, struct page  
*), void * data);
```

### Arguments

*mapping*

the page's address\_space

*index*

the page index

*filler*

function to perform the read

*data*

first arg to filler(data, page) function, often left as NULL

## Description

Same as `read_cache_page`, but don't wait for page to become unlocked after submitting it to the filler.

Read into the page cache. If a page already exists, and `PageUptodate` is not set, try to fill the page but don't wait for it to become unlocked.

If the page does not get brought uptodate, return `-EIO`.

# read\_cache\_page\_gfp

## LINUX

Kernel Hackers Manual June 2013

## Name

`read_cache_page_gfp` — read into page cache, using specified page allocation flags.

## Synopsis

```
struct page * read_cache_page_gfp (struct address_space *  
mapping, pgoff_t index, gfp_t gfp);
```

## Arguments

*mapping*

the page's `address_space`

*index*

the page index

*gfp*

the page allocator flags to use if allocating

## Description

This is the same as “`read_mapping_page(mapping, index, NULL)`”, but with any new page allocations done using the specified allocation flags.

If the page does not get brought uptodate, return `-EIO`.

# read\_cache\_page

## LINUX

Kernel Hackers Manual June 2013

## Name

`read_cache_page` — read into page cache, fill it if needed

## Synopsis

```
struct page * read_cache_page (struct address_space * mapping,
pgoff_t index, int (*filler) (void *, struct page *), void *
data);
```

## Arguments

*mapping*

the page’s `address_space`

*index*

the page index

*filler*

function to perform the read

*data*

first arg to filler(data, page) function, often left as NULL

## Description

Read into the page cache. If a page already exists, and `PageUptodate` is not set, try to fill the page then wait for it to become unlocked.

If the page does not get brought uptodate, return `-EIO`.

# \_\_generic\_file\_aio\_write

**LINUX**

Kernel Hackers Manual June 2013

## Name

`__generic_file_aio_write` — write data to a file

## Synopsis

```
ssize_t __generic_file_aio_write (struct kiocb * iocb, const
struct iovec * iov, unsigned long nr_segs, loff_t * ppos);
```

## Arguments

*iocb*

IO state structure (file, offset, etc.)

*iov*

vector with data to write

*nr\_segs*

number of segments in the vector

*ppos*

position where to write

## Description

This function does all the work needed for actually writing data to a file. It does all basic checks, removes SUID from the file, updates modification times and calls proper subroutines depending on whether we do direct IO or a standard buffered write.

It expects `i_mutex` to be grabbed unless we work on a block device or similar object which does not need locking at all.

This function does *\*not\** take care of syncing data in case of `O_SYNC` write. A caller has to handle it. This is mainly due to the fact that we want to avoid syncing under `i_mutex`.

# generic\_file\_aio\_write

## LINUX

Kernel Hackers Manual June 2013

## Name

`generic_file_aio_write` — write data to a file

## Synopsis

```
ssize_t generic_file_aio_write (struct kiocb * iocb, const
struct iovec * iov, unsigned long nr_segs, loff_t pos);
```



## Arguments

*iocb*

IO state structure

*iov*

vector with data to write

*nr\_segs*

number of segments in the vector

*pos*

position in file where to write

## Description

This is a wrapper around `__generic_file_aio_write` to be used by most filesystems. It takes care of syncing the file in case of `O_SYNC` file and acquires `i_mutex` as needed.

# try\_to\_release\_page

**LINUX**

Kernel Hackers Manual June 2013

## Name

`try_to_release_page` — release old fs-specific metadata on a page

## Synopsis

```
int try_to_release_page (struct page * page, gfp_t gfp_mask);
```

## Arguments

*page*

the page which the kernel is trying to free

*gfp\_mask*

memory allocation flags (and I/O mode)

## Description

The *address\_space* is to try to release any data against the page (presumably at *page->private*). If the release was successful, return '1'. Otherwise return zero.

This may also be called if *PG\_fscache* is set on a page, indicating that the page is known to the local caching routines.

The *gfp\_mask* argument specifies whether I/O may be performed to release this page (*\_\_GFP\_IO*), and whether the call may block (*\_\_GFP\_WAIT* & *\_\_GFP\_FS*).

## zap\_page\_range

### LINUX

Kernel Hackers Manual June 2013

## Name

*zap\_page\_range* — remove user pages in a given range

## Synopsis

```
unsigned long zap_page_range (struct vm_area_struct * vma,
unsigned long address, unsigned long size, struct zap_details
* details);
```

## Arguments

*vma*

vm\_area\_struct holding the applicable pages

*address*

starting address of pages to zap

*size*

number of bytes to zap

*details*

details of nonlinear truncation or shared cache invalidation

## zap\_vma\_ptes

**LINUX**

Kernel Hackers Manual June 2013

## Name

zap\_vma\_ptes — remove ptes mapping the vma

## Synopsis

```
int zap_vma_ptes (struct vm_area_struct * vma, unsigned long
address, unsigned long size);
```

## Arguments

*vma*

vm\_area\_struct holding ptes to be zapped

*address*

starting address of pages to zap

*size*

number of bytes to zap

## Description

This function only unmaps ptes assigned to VM\_PFNMAP vmas.

The entire address range must be fully contained within the vma.

Returns 0 if successful.

## \_\_get\_user\_pages

### LINUX

Kernel Hackers Manual June 2013

## Name

\_\_get\_user\_pages — pin user pages in memory

## Synopsis

```
int __get_user_pages (struct task_struct * tsk, struct
mm_struct * mm, unsigned long start, int nr_pages, unsigned
int gup_flags, struct page ** pages, struct vm_area_struct **
vmas, int * nonblocking);
```

## Arguments

*tsk*

task\_struct of target task

*mm*

mm\_struct of target mm

*start*

starting user address

*nr\_pages*

number of pages from start to pin

*gup\_flags*

flags modifying pin behaviour

*pages*

array that receives pointers to the pages pinned. Should be at least *nr\_pages* long. Or NULL, if caller only intends to ensure the pages are faulted in.

*vmas*

array of pointers to vmas corresponding to each page. Or NULL if the caller does not require them.

*nonblocking*

whether waiting for disk IO or mmap\_sem contention

## Description

Returns number of pages pinned. This may be fewer than the number requested. If `nr_pages` is 0 or negative, returns 0. If no pages were pinned, returns `-errno`. Each page returned must be released with a `put_page` call when it is finished with. `vmas` will only remain valid while `mmap_sem` is held.

Must be called with `mmap_sem` held for read or write.

`__get_user_pages` walks a process's page tables and takes a reference to each struct page that each user address corresponds to at a given instant. That is, it takes the page that would be accessed if a user thread accesses the given user virtual address at that instant.

This does not guarantee that the page exists in the user mappings when `__get_user_pages` returns, and there may even be a completely different page there in some cases (eg. if mmapped pagecache has been invalidated and subsequently re faulted). However it does guarantee that the page won't be freed completely. And mostly callers simply care that the page contains data that was valid \*at some point in time\*. Typically, an IO or similar operation cannot guarantee anything stronger anyway because locks can't be held over the syscall boundary.

If `gup_flags & FOLL_WRITE == 0`, the page must not be written to. If the page is written to, `set_page_dirty` (or `set_page_dirty_lock`, as appropriate) must be called after the page is finished with, and before `put_page` is called.

If `nonblocking != NULL`, `__get_user_pages` will not wait for disk IO or `mmap_sem` contention, and if waiting is needed to pin all pages, `*nonblocking` will be set to 0.

In most cases, `get_user_pages` or `get_user_pages_fast` should be used instead of `__get_user_pages`. `__get_user_pages` should be used only if you need some special `gup_flags`.

## vm\_insert\_page

### LINUX

Kernel Hackers Manual June 2013

### Name

`vm_insert_page` — insert single page into user vma

## Synopsis

```
int vm_insert_page (struct vm_area_struct * vma, unsigned long
addr, struct page * page);
```

## Arguments

*vma*

user vma to map to

*addr*

target user address of this page

*page*

source kernel page

## Description

This allows drivers to insert individual pages they've allocated into a user vma.

The page has to be a nice clean `_individual_` kernel allocation. If you allocate a compound page, you need to have marked it as such (`__GFP_COMP`), or manually just split the page up yourself (see `split_page`).

NOTE! Traditionally this was done with “`remap_pfn_range`” which took an arbitrary page protection parameter. This doesn't allow that. Your vma protection will have to be set up correctly, which means that if you want a shared writable mapping, you'd better ask for a shared writable mapping!

The page does not need to be reserved.

## vm\_insert\_pfn

**LINUX**

## Name

`vm_insert_pfn` — insert single pfn into user vma

## Synopsis

```
int vm_insert_pfn (struct vm_area_struct * vma, unsigned long  
addr, unsigned long pfn);
```

## Arguments

*vma*

user vma to map to

*addr*

target user address of this page

*pfn*

source kernel pfn

## Description

Similar to `vm_inert_page`, this allows drivers to insert individual pages they've allocated into a user vma. Same comments apply.

This function should only be called from a `vm_ops->fault` handler, and in that case the handler should return `NULL`.

*vma* cannot be a COW mapping.

As this is called only for pages that do not currently exist, we do not need to flush old virtual caches or the TLB.



# remap\_pfn\_range

## LINUX

Kernel Hackers Manual June 2013

### Name

`remap_pfn_range` — remap kernel memory to userspace

### Synopsis

```
int remap_pfn_range (struct vm_area_struct * vma, unsigned
long addr, unsigned long pfn, unsigned long size, pgprot_t
prot);
```

### Arguments

*vma*

user vma to map to

*addr*

target user address to start at

*pfn*

physical address of kernel memory

*size*

size of map area

*prot*

page protection flags for this mapping

### Note

this is only safe if the mm semaphore is held when called.

# vm\_iomap\_memory

## LINUX

Kernel Hackers Manual June 2013

### Name

`vm_iomap_memory` — remap memory to userspace

### Synopsis

```
int vm_iomap_memory (struct vm_area_struct * vma, phys_addr_t
start, unsigned long len);
```

### Arguments

*vma*

user vma to map to

*start*

start of area

*len*

size of area

### Description

This is a simplified `io_remap_pfn_range` for common driver use. The driver just needs to give us the physical memory range to be mapped, we'll figure out the rest from the vma information.

NOTE! Some drivers might want to tweak `vma->vm_page_prot` first to get whatever write-combining details or similar.

# unmap\_mapping\_range

## LINUX

Kernel Hackers Manual June 2013

### Name

`unmap_mapping_range` — unmap the portion of all mmaps in the specified `address_space` corresponding to the specified page range in the underlying file.

### Synopsis

```
void unmap_mapping_range (struct address_space * mapping,  
loff_t const holebegin, loff_t const holelen, int even_cows);
```

### Arguments

*mapping*

the address space containing mmaps to be unmapped.

*holebegin*

byte in first page to unmap, relative to the start of the underlying file. This will be rounded down to a `PAGE_SIZE` boundary. Note that this is different from `truncate_pagecache`, which must keep the partial page. In contrast, we must get rid of partial pages.

*holelen*

size of prospective hole in bytes. This will be rounded up to a `PAGE_SIZE` boundary. A `holelen` of zero truncates to the end of the file.

*even\_cows*

1 when truncating a file, unmap even private COWed pages; but 0 when invalidating pagecache, don't throw away private data.

## follow\_pfn

### LINUX

Kernel Hackers Manual June 2013

### Name

`follow_pfn` — look up PFN at a user virtual address

### Synopsis

```
int follow_pfn (struct vm_area_struct * vma, unsigned long
address, unsigned long * pfn);
```

### Arguments

*vma*

memory mapping

*address*

user virtual address

*pfn*

location to store found PFN

## Description

Only IO mappings and raw PFN mappings are allowed.

Returns zero and the pfn at *pfn* on success, -ve otherwise.

# vm\_unmap\_aliases

## LINUX

Kernel Hackers Manual June 2013

## Name

`vm_unmap_aliases` — unmap outstanding lazy aliases in the vmap layer

## Synopsis

```
void vm_unmap_aliases ( void );
```

## Arguments

*void*

no arguments

## Description

The vmap/vmalloc layer lazily flushes kernel virtual mappings primarily to amortize TLB flushing overheads. What this means is that any page you have now, may, in a former life, have been mapped into kernel virtual address by the vmap layer and so there might be some CPUs with TLB entries still referencing that page (additional to the regular 1:1 kernel mapping).

`vm_unmap_aliases` flushes all such lazy mappings. After it returns, we can be sure that none of the pages we have control over will have any aliases from the `vmap` layer.

## vm\_unmap\_ram

### LINUX

Kernel Hackers Manual June 2013

### Name

`vm_unmap_ram` — unmap linear kernel address space set up by `vm_map_ram`

### Synopsis

```
void vm_unmap_ram (const void * mem, unsigned int count);
```

### Arguments

*mem*

the pointer returned by `vm_map_ram`

*count*

the count passed to that `vm_map_ram` call (cannot unmap partial)

## vm\_map\_ram

### LINUX

## Name

`vm_map_ram` — map pages linearly into kernel virtual address (vmalloc space)

## Synopsis

```
void * vm_map_ram (struct page ** pages, unsigned int count,  
int node, pgprot_t prot);
```

## Arguments

*pages*

an array of pointers to the pages to be mapped

*count*

number of pages

*node*

prefer to allocate data structures on this node

*prot*

memory protection to use. `PAGE_KERNEL` for regular RAM

## Returns

a pointer to the address that has been mapped, or `NULL` on failure

# unmap\_kernel\_range\_noflush

**LINUX**

## Name

`unmap_kernel_range_noflush` — unmap kernel VM area

## Synopsis

```
void unmap_kernel_range_noflush (unsigned long addr, unsigned  
long size);
```

## Arguments

*addr*

start of the VM area to unmap

*size*

size of the VM area to unmap

## Description

Unmap `PFN_UP(size)` pages at *addr*. The VM area *addr* and *size* specify should have been allocated using `get_vm_area` and its friends.

## NOTE

This function does NOT do any cache flushing. The caller is responsible for calling `flush_cache_vunmap` on to-be-mapped areas before calling this function and `flush_tlb_kernel_range` after.



# vfree

## LINUX

Kernel Hackers Manual June 2013

### Name

`vfree` — release memory allocated by `vmalloc`

### Synopsis

```
void vfree (const void * addr);
```

### Arguments

*addr*

memory base address

### Description

Free the virtually continuous memory area starting at *addr*, as obtained from `vmalloc`, `vmalloc_32` or `__vmalloc`. If *addr* is `NULL`, no operation is performed.

Must not be called in interrupt context.

# vunmap

## LINUX

## Name

`vunmap` — release virtual mapping obtained by `vmap`

## Synopsis

```
void vunmap (const void * addr);
```

## Arguments

*addr*

memory base address

## Description

Free the virtually contiguous memory area starting at *addr*, which was created from the page array passed to `vmap`.

Must not be called in interrupt context.

# vmap

## LINUX

## Name

`vmap` — map an array of pages into virtually contiguous space

## Synopsis

```
void * vmap (struct page ** pages, unsigned int count,  
unsigned long flags, pgprot_t prot);
```

## Arguments

*pages*

array of page pointers

*count*

number of pages to map

*flags*

vm\_area->flags

*prot*

page protection for the mapping

## Description

Maps *count* pages from *pages* into contiguous kernel virtual space.

## vmalloc

**LINUX**

Kernel Hackers Manual June 2013

## Name

`vmalloc` — allocate virtually contiguous memory

## Synopsis

```
void * vmalloc (unsigned long size);
```

## Arguments

*size*

allocation size Allocate enough pages to cover *size* from the page level allocator and map them into contiguous kernel virtual space.

## Description

For tight control over page level allocator and protection flags use `__vmalloc` instead.

## vzalloc

### LINUX

Kernel Hackers Manual June 2013

## Name

`vzalloc` — allocate virtually contiguous memory with zero fill

## Synopsis

```
void * vzalloc (unsigned long size);
```

## Arguments

*size*

allocation size Allocate enough pages to cover *size* from the page level allocator and map them into contiguous kernel virtual space. The memory allocated is set to zero.

## Description

For tight control over page level allocator and protection flags use `__vmalloc` instead.

# vmalloc\_user

## LINUX

Kernel Hackers Manual June 2013

## Name

`vmalloc_user` — allocate zeroed virtually contiguous memory for userspace

## Synopsis

```
void * vmalloc_user (unsigned long size);
```

## Arguments

*size*

allocation size

## Description

The resulting memory area is zeroed so it can be mapped to userspace without leaking data.

# vmalloc\_node

## LINUX

Kernel Hackers Manual June 2013

## Name

`vmalloc_node` — allocate memory on a specific node

## Synopsis

```
void * vmalloc_node (unsigned long size, int node);
```

## Arguments

*size*

allocation size

*node*

numa node

## Description

Allocate enough pages to cover *size* from the page level allocator and map them into contiguous kernel virtual space.

For tight control over page level allocator and protection flags use `__vmalloc` instead.

## vzalloc\_node

### LINUX

Kernel Hackers Manual June 2013

### Name

`vzalloc_node` — allocate memory on a specific node with zero fill

### Synopsis

```
void * vzalloc_node (unsigned long size, int node);
```

### Arguments

*size*

allocation size

*node*

numa node

### Description

Allocate enough pages to cover *size* from the page level allocator and map them into contiguous kernel virtual space. The memory allocated is set to zero.

For tight control over page level allocator and protection flags use `__vmalloc_node` instead.

## vmalloc\_32

### LINUX

Kernel Hackers Manual June 2013

### Name

`vmalloc_32` — allocate virtually contiguous memory (32bit addressable)

### Synopsis

```
void * vmalloc_32 (unsigned long size);
```

### Arguments

*size*

allocation size

### Description

Allocate enough 32bit PA addressable pages to cover *size* from the page level allocator and map them into contiguous kernel virtual space.

## vmalloc\_32\_user

### LINUX



## Name

`vmalloc_32_user` — allocate zeroed virtually contiguous 32bit memory

## Synopsis

```
void * vmalloc_32_user (unsigned long size);
```

## Arguments

*size*

allocation size

## Description

The resulting memory area is 32bit addressable and zeroed so it can be mapped to userspace without leaking data.

# remap\_vmalloc\_range

## LINUX

## Name

`remap_vmalloc_range` — map vmalloc pages to userspace

## Synopsis

```
int remap_vmalloc_range (struct vm_area_struct * vma, void *  
addr, unsigned long pgoff);
```

## Arguments

*vma*

*vma* to cover (map full range of *vma*)

*addr*

vmalloc memory

*pgoff*

number of pages into *addr* before first page to map

## Returns

0 for success, -Exxx on failure

This function checks that *addr* is a valid vmalloc'ed area, and that it is big enough to cover the *vma*. Will return failure if that criteria isn't met.

Similar to `remap_pfn_range` (see `mm/memory.c`)

## alloc\_vm\_area

### LINUX

Kernel Hackers Manual June 2013

## Name

`alloc_vm_area` — allocate a range of kernel address space

## Synopsis

```
struct vm_struct * alloc_vm_area (size_t size);
```

## Arguments

*size*

size of the area

## Returns

NULL on failure, vm\_struct on success

This function reserves a range of kernel address space, and allocates pagetables to map that range. No actual mappings are created. If the kernel address space is not shared between processes, it syncs the pagetable across all processes.

# find\_next\_best\_node

## LINUX

Kernel Hackers Manual June 2013

## Name

`find_next_best_node` — find the next node that should appear in a given node's fallback list

## Synopsis

```
int find_next_best_node (int node, nodemask_t *  
used_node_mask);
```

## Arguments

*node*

node whose fallback list we're appending

*used\_node\_mask*

nodemask\_t of already used nodes

## Description

We use a number of factors to determine which is the next node that should appear on a given node's fallback list. The node should not have appeared already in *node*'s fallback list, and it should be the next closest node according to the distance array (which contains arbitrary distance values from each node to each node in the system), and should also prefer nodes with no CPUs, since presumably they'll have very little allocation pressure on them otherwise. It returns -1 if no node is found.

# free\_bootmem\_with\_active\_regions

## LINUX

Kernel Hackers Manual June 2013

## Name

`free_bootmem_with_active_regions` — Call `free_bootmem_node` for each active range

## Synopsis

```
void free_bootmem_with_active_regions (int nid, unsigned long  
max_low_pfn);
```

## Arguments

*nid*

The node to free memory on. If MAX\_NUMNODES, all nodes are freed.

*max\_low\_pfn*

The highest PFN that will be passed to `free_bootmem_node`

## Description

If an architecture guarantees that all ranges registered with `add_active_ranges` contain no holes and may be freed, this this function may be used instead of calling `free_bootmem` manually.

# sparse\_memory\_present\_with\_active\_regions

## LINUX

Kernel Hackers Manual June 2013

## Name

`sparse_memory_present_with_active_regions` — Call `memory_present` for each active range

## Synopsis

```
void sparse_memory_present_with_active_regions (int nid);
```

## Arguments

*nid*

The node to call `memory_present` for. If `MAX_NUMNODES`, all nodes will be used.

## Description

If an architecture guarantees that all ranges registered with `add_active_ranges` contain no holes and may be freed, this function may be used instead of calling `memory_present` manually.

# get\_pfn\_range\_for\_nid

## LINUX

Kernel Hackers Manual June 2013

## Name

`get_pfn_range_for_nid` — Return the start and end page frames for a node

## Synopsis

```
void __meminit get_pfn_range_for_nid (unsigned int nid,  
unsigned long * start_pfn, unsigned long * end_pfn);
```

## Arguments

*nid*

The `nid` to return the range for. If `MAX_NUMNODES`, the min and max PFN are returned.

*start\_pfn*

Passed by reference. On return, it will have the node *start\_pfn*.

*end\_pfn*

Passed by reference. On return, it will have the node *end\_pfn*.

## Description

It returns the start and end page frame of a node based on information provided by an arch calling `add_active_range`. If called for a node with no available memory, a warning is printed and the start and end PFNs will be 0.

# absent\_pages\_in\_range

## LINUX

Kernel Hackers Manual June 2013

## Name

`absent_pages_in_range` — Return number of page frames in holes within a range

## Synopsis

```
unsigned long absent_pages_in_range (unsigned long start_pfn,  
unsigned long end_pfn);
```

## Arguments

*start\_pfn*

The start PFN to start searching for holes

*end\_pfn*

The end PFN to stop searching for holes

## Description

It returns the number of pages frames in memory holes within a range.

# add\_active\_range

## LINUX

Kernel Hackers Manual June 2013

## Name

`add_active_range` — Register a range of PFNs backed by physical memory

## Synopsis

```
void add_active_range (unsigned int nid, unsigned long  
start_pfn, unsigned long end_pfn);
```

## Arguments

*nid*

The node ID the range resides on

*start\_pfn*

The start PFN of the available physical memory

*end\_pfn*

The end PFN of the available physical memory



## Description

These ranges are stored in an `early_node_map[]` and later used by `free_area_init_nodes` to calculate zone sizes and holes. If the range spans a memory hole, it is up to the architecture to ensure the memory is not freed by the bootmem allocator. If possible the range being registered will be merged with existing ranges.

## remove\_active\_range

### LINUX

Kernel Hackers Manual June 2013

## Name

`remove_active_range` — Shrink an existing registered range of PFNs

## Synopsis

```
void remove_active_range (unsigned int nid, unsigned long
    start_pfn, unsigned long end_pfn);
```

## Arguments

*nid*

The node id the range is on that should be shrunk

*start\_pfn*

The new PFN of the range

*end\_pfn*

The new PFN of the range

## Description

i386 with NUMA use `alloc_remap` to store a `node_mem_map` on a local node. The map is kept near the end physical page range that has already been registered. This function allows an arch to shrink an existing registered range.

# remove\_all\_active\_ranges

## LINUX

Kernel Hackers Manual June 2013

## Name

`remove_all_active_ranges` — Remove all currently registered regions

## Synopsis

```
void remove_all_active_ranges ( void );
```

## Arguments

*void*

no arguments

## Description

During discovery, it may be found that a table like SRAT is invalid and an alternative discovery method must be used. This function removes all currently registered regions.

# find\_min\_pfn\_with\_active\_regions

## LINUX

Kernel Hackers Manual June 2013

### Name

`find_min_pfn_with_active_regions` — Find the minimum PFN registered

### Synopsis

```
unsigned long find_min_pfn_with_active_regions ( void );
```

### Arguments

*void*

no arguments

### Description

It returns the minimum PFN based on information provided via `add_active_range`.

# free\_area\_init\_nodes

## LINUX

## Name

`free_area_init_nodes` — Initialise all `pg_data_t` and zone data

## Synopsis

```
void free_area_init_nodes (unsigned long * max_zone_pfn);
```

## Arguments

*max\_zone\_pfn*

an array of max PFNs for each zone

## Description

This will call `free_area_init_node` for each active node in the system. Using the page ranges provided by `add_active_range`, the size of each zone in each node and their holes is calculated. If the maximum PFN between two adjacent zones match, it is assumed that the zone is empty. For example, if `arch_max_dma_pfn == arch_max_dma32_pfn`, it is assumed that `arch_max_dma32_pfn` has no pages. It is also assumed that a zone starts where the previous one ended. For example, `ZONE_DMA32` starts at `arch_max_dma_pfn`.

## set\_dma\_reserve

**LINUX**

## Name

`set_dma_reserve` — set the specified number of pages reserved in the first zone

## Synopsis

```
void set_dma_reserve (unsigned long new_dma_reserve);
```

## Arguments

*new\_dma\_reserve*

The number of pages to mark reserved

## Description

The per-cpu batchsize and zone watermarks are determined by `present_pages`. In the DMA zone, a significant percentage may be consumed by kernel image and other unfreeable allocations which can skew the watermarks badly. This function may optionally be used to account for unfreeable pages in the first zone (e.g., `ZONE_DMA`). The effect will be lower watermarks and smaller per-cpu batchsize.

## setup\_per\_zone\_wmarks

**LINUX**

## Name

`setup_per_zone_wmarks` — called when `min_free_kbytes` changes or when memory is hot-`{added|removed}`

## Synopsis

```
void setup_per_zone_wmarks ( void );
```

## Arguments

*void*

no arguments

## Description

Ensures that the watermark[`min,low,high`] values for each zone are set correctly with respect to `min_free_kbytes`.

# get\_pageblock\_flags\_group

## LINUX

## Name

`get_pageblock_flags_group` — Return the requested group of flags for the `pageblock_nr_pages` block of pages

## Synopsis

```
unsigned long get_pageblock_flags_group (struct page * page,  
int start_bitidx, int end_bitidx);
```

## Arguments

*page*

The page within the block of interest

*start\_bitidx*

The first bit of interest to retrieve

*end\_bitidx*

The last bit of interest returns pageblock\_bits flags

## set\_pageblock\_flags\_group

### LINUX

Kernel Hackers Manual June 2013

## Name

`set_pageblock_flags_group` — Set the requested group of flags for a `pageblock_nr_pages` block of pages

## Synopsis

```
void set_pageblock_flags_group (struct page * page, unsigned  
long flags, int start_bitidx, int end_bitidx);
```

## Arguments

*page*

The page within the block of interest

*flags*

The flags to set

*start\_bitidx*

The first bit of interest

*end\_bitidx*

The last bit of interest

## mempool\_create

**LINUX**

Kernel Hackers Manual June 2013

### Name

`mempool_create` — create a memory pool

### Synopsis

```
mempool_t * mempool_create (int min_nr, mempool_alloc_t *  
alloc_fn, mempool_free_t * free_fn, void * pool_data);
```



## Arguments

*min\_nr*

the minimum number of elements guaranteed to be allocated for this pool.

*alloc\_fn*

user-defined element-allocation function.

*free\_fn*

user-defined element-freeing function.

*pool\_data*

optional private data available to the user-defined functions.

## Description

this function creates and allocates a guaranteed size, preallocated memory pool. The pool can be used from the `mempool_alloc` and `mempool_free` functions. This function might sleep. Both the `alloc_fn` and the `free_fn` functions might sleep - as long as the `mempool_alloc` function is not called from IRQ contexts.

# mempool\_resize

**LINUX**

Kernel Hackers Manual June 2013

## Name

`mempool_resize` — resize an existing memory pool

## Synopsis

```
int mempool_resize (mempool_t * pool, int new_min_nr, gfp_t
gfp_mask);
```

## Arguments

*pool*

pointer to the memory pool which was allocated via `mempool_create`.

*new\_min\_nr*

the new minimum number of elements guaranteed to be allocated for this pool.

*gfp\_mask*

the usual allocation bitmask.

## Description

This function shrinks/grows the pool. In the case of growing, it cannot be guaranteed that the pool will be grown to the new size immediately, but new `mempool_free` calls will refill it.

Note, the caller must guarantee that no `mempool_destroy` is called while this function is running. `mempool_alloc` & `mempool_free` might be called (eg. from IRQ contexts) while this function executes.

# mempool\_destroy

**LINUX**

Kernel Hackers Manual June 2013

## Name

`mempool_destroy` — deallocate a memory pool

## Synopsis

```
void mempool_destroy (mempool_t * pool);
```

## Arguments

*pool*

pointer to the memory pool which was allocated via `mempool_create`.

## Description

this function only sleeps if the `free_fn` function sleeps. The caller has to guarantee that all elements have been returned to the pool (ie: freed) prior to calling `mempool_destroy`.

# mempool\_alloc

## LINUX

Kernel Hackers Manual June 2013

## Name

`mempool_alloc` — allocate an element from a specific memory pool

## Synopsis

```
void * mempool_alloc (mempool_t * pool, gfp_t gfp_mask);
```

## Arguments

*pool*

pointer to the memory pool which was allocated via `mempool_create`.

*gfp\_mask*

the usual allocation bitmask.

## Description

this function only sleeps if the `alloc_fn` function sleeps or returns NULL. Note that due to preallocation, this function *\*never\** fails when called from process contexts. (it might fail if called from an IRQ context.)

# mempool\_free

## LINUX

Kernel Hackers Manual June 2013

## Name

`mempool_free` — return an element to the pool.

## Synopsis

```
void mempool_free (void * element, mempool_t * pool);
```

## Arguments

*element*

pool element pointer.

*pool*

pointer to the memory pool which was allocated via `mempool_create`.

## Description

this function only sleeps if the `free_fn` function sleeps.

# dma\_pool\_create

## LINUX

Kernel Hackers Manual June 2013

## Name

`dma_pool_create` — Creates a pool of consistent memory blocks, for dma.

## Synopsis

```
struct dma_pool * dma_pool_create (const char * name, struct  
device * dev, size_t size, size_t align, size_t boundary);
```

## Arguments

*name*

name of pool, for diagnostics

*dev*

device that will be doing the DMA

*size*

size of the blocks in this pool.

*align*

alignment requirement for blocks; must be a power of two

*boundary*

returned blocks won't cross this power of two boundary

## Context

`!in_interrupt`

## Description

Returns a dma allocation pool with the requested characteristics, or null if one can't be created. Given one of these pools, `dma_pool_alloc` may be used to allocate memory. Such memory will all have “consistent” DMA mappings, accessible by the device and its driver without using cache flushing primitives. The actual size of blocks allocated may be larger than requested because of alignment.

If *boundary* is nonzero, objects returned from `dma_pool_alloc` won't cross that size boundary. This is useful for devices which have addressing restrictions on individual DMA transfers, such as not crossing boundaries of 4KBytes.

# `dma_pool_destroy`

## LINUX

Kernel Hackers Manual June 2013

## Name

`dma_pool_destroy` — destroys a pool of dma memory blocks.

## Synopsis

```
void dma_pool_destroy (struct dma_pool * pool);
```

## Arguments

*pool*

dma pool that will be destroyed

## Context

`!in_interrupt`

## Description

Caller guarantees that no more memory from the pool is in use, and that nothing will try to use the pool after this call.

# dma\_pool\_alloc

## LINUX

Kernel Hackers Manual June 2013

## Name

`dma_pool_alloc` — get a block of consistent memory

## Synopsis

```
void * dma_pool_alloc (struct dma_pool * pool, gfp_t  
mem_flags, dma_addr_t * handle);
```

## Arguments

*pool*

dma pool that will produce the block

*mem\_flags*

GFP\_\* bitmask

*handle*

pointer to dma address of block

## Description

This returns the kernel virtual address of a currently unused block, and reports its dma address through the handle. If such a memory block can't be allocated, `NULL` is returned.

## `dma_pool_free`

### LINUX

Kernel Hackers Manual June 2013

## Name

`dma_pool_free` — put block back into dma pool

## Synopsis

```
void dma_pool_free (struct dma_pool * pool, void * vaddr,  
dma_addr_t dma);
```



## Arguments

*pool*

the dma pool holding the block

*vaddr*

virtual address of block

*dma*

dma address of block

## Description

Caller promises neither device nor driver will again touch this block unless it is first re-allocated.

# dmam\_pool\_create

## LINUX

Kernel Hackers Manual June 2013

## Name

dmam\_pool\_create — Managed dma\_pool\_create

## Synopsis

```
struct dma_pool * dmam_pool_create (const char * name, struct  
device * dev, size_t size, size_t align, size_t allocation);
```

## Arguments

*name*

name of pool, for diagnostics

*dev*

device that will be doing the DMA

*size*

size of the blocks in this pool.

*align*

alignment requirement for blocks; must be a power of two

*allocation*

returned blocks won't cross this boundary (or zero)

## Description

Managed `dma_pool_create`. DMA pool created with this function is automatically destroyed on driver detach.

# dmam\_pool\_destroy

## LINUX

Kernel Hackers Manual June 2013

## Name

`dmam_pool_destroy` — Managed `dma_pool_destroy`

## Synopsis

```
void dmam_pool_destroy (struct dma_pool * pool);
```

## Arguments

*pool*

dma pool that will be destroyed

## Description

Managed `dma_pool_destroy`.

# balance\_dirty\_pages\_ratelimited\_nr

## LINUX

Kernel Hackers Manual June 2013

## Name

`balance_dirty_pages_ratelimited_nr` — balance dirty memory state

## Synopsis

```
void balance_dirty_pages_ratelimited_nr (struct address_space  
* mapping, unsigned long nr_pages_dirtied);
```

## Arguments

*mapping*

address\_space which was dirtied

*nr\_pages\_dirtied*

number of pages which the caller has just dirtied

## Description

Processes which are dirtying memory should call in here once for each page which was newly dirtied. The function will periodically check the system's dirty state and will initiate writeback if needed.

On really big machines, `get_writeback_state` is expensive, so try to avoid calling it too often (ratelimiting). But once we're over the dirty memory limit we decrease the ratelimiting by a lot, to prevent individual processes from overshooting the limit by (`ratelimit_pages`) each.

# tag\_pages\_for\_writeback

**LINUX**

Kernel Hackers Manual June 2013

## Name

`tag_pages_for_writeback` — tag pages to be written by `write_cache_pages`

## Synopsis

```
void tag_pages_for_writeback (struct address_space * mapping,  
pgoff_t start, pgoff_t end);
```

## Arguments

*mapping*

address space structure to write

*start*

starting page index

*end*

ending page index (inclusive)

## Description

This function scans the page range from *start* to *end* (inclusive) and tags all pages that have DIRTY tag set with a special TOWRITE tag. The idea is that `write_cache_pages` (or whoever calls this function) will then use TOWRITE tag to identify pages eligible for writeback. This mechanism is used to avoid livelocking of writeback by a process steadily creating new dirty pages in the file (thus it is important for this function to be quick so that it can tag pages faster than a dirtying process can create them).

# write\_cache\_pages

## LINUX

Kernel Hackers Manual June 2013

## Name

`write_cache_pages` — walk the list of dirty pages of the given address space and write all of them.

## Synopsis

```
int write_cache_pages (struct address_space * mapping, struct
writeback_control * wbc, writepage_t writepage, void * data);
```

## Arguments

*mapping*

address space structure to write

*wbc*

subtract the number of written pages from `*wbc->nr_to_write`

*writepage*

function called for each page

*data*

data passed to writepage function

## Description

If a page is already under I/O, `write_cache_pages` skips it, even if it's dirty. This is desirable behaviour for memory-cleaning writeback, but it is INCORRECT for data-integrity system calls such as `fsync`. `fsync` and `msync` need to guarantee that all the data which was dirty at the time the call was made get new I/O started against them. If `wbc->sync_mode` is `WB_SYNC_ALL` then we were called for data integrity and we must wait for existing IO to complete.

To avoid livelocks (when other process dirties new pages), we first tag pages which should be written back with `TOWRITE` tag and only then start writing them. For data-integrity sync we have to be careful so that we do not miss some pages (e.g., because some other process has cleared `TOWRITE` tag we set). The rule we follow is that `TOWRITE` tag can be cleared only by the process clearing the `DIRTY` tag (and submitting the page for IO).

## generic\_writepages

**LINUX**

## Name

`generic_writepages` — walk the list of dirty pages of the given address space and `writepage` all of them.

## Synopsis

```
int generic_writepages (struct address_space * mapping, struct
writeback_control * wbc);
```

## Arguments

*mapping*

address space structure to write

*wbc*

subtract the number of written pages from `*wbc->nr_to_write`

## Description

This is a library function, which implements the `writepages` `address_space_operation`.

## write\_one\_page

**LINUX**

## Name

`write_one_page` — write out a single page and optionally wait on I/O

## Synopsis

```
int write_one_page (struct page * page, int wait);
```

## Arguments

*page*

the page to write

*wait*

if true, wait on writeout

## Description

The page must be locked by the caller and will be unlocked upon return.

`write_one_page` returns a negative error code if I/O failed.

# wait\_for\_stable\_page

## LINUX

## Name

`wait_for_stable_page` — wait for writeback to finish, if necessary.



## Synopsis

```
void wait_for_stable_page (struct page * page);
```

## Arguments

*page*

The page to wait on.

## Description

This function determines if the given page is related to a backing device that requires page contents to be held stable during writeback. If so, then it will wait for any pending writeback to complete.

# truncate\_inode\_pages\_range

## LINUX

Kernel Hackers Manual June 2013

## Name

`truncate_inode_pages_range` — truncate range of pages specified by start & end byte offsets

## Synopsis

```
void truncate_inode_pages_range (struct address_space *  
mapping, loff_t lstart, loff_t lend);
```

## Arguments

*mapping*

mapping to truncate

*lstart*

offset from which to truncate

*lend*

offset to which to truncate

## Description

Truncate the page cache, removing the pages that are between specified offsets (and zeroing out partial page (if *lstart* is not page aligned)).

Truncate takes two passes - the first pass is nonblocking. It will not block on page locks and it will not block on writeback. The second pass will wait. This is to prevent as much IO as possible in the affected region. The first pass will remove most pages, so the search cost of the second pass is low.

We pass down the cache-hot hint to the page freeing code. Even if the mapping is large, it is probably the case that the final pages are the most recently touched, and freeing happens in ascending file offset order.

## truncate\_inode\_pages

**LINUX**

Kernel Hackers Manual June 2013

### Name

`truncate_inode_pages` — truncate *\*all\** the pages from an offset

## Synopsis

```
void truncate_inode_pages (struct address_space * mapping,  
loff_t lstart);
```

## Arguments

*mapping*

mapping to truncate

*lstart*

offset from which to truncate

## Description

Called under (and serialised by) inode->i\_mutex.

## Note

When this function returns, there can be a page in the process of deletion (inside `__delete_from_page_cache`) in the specified range. Thus `mapping->nrpages` can be non-zero when this function returns even after truncation of the whole `mapping`.

## invalidate\_mapping\_pages

**LINUX**

## Name

`invalidate_mapping_pages` — Invalidate all the unlocked pages of one inode

## Synopsis

```
unsigned long invalidate_mapping_pages (struct address_space *  
mapping, pgoff_t start, pgoff_t end);
```

## Arguments

*mapping*

the `address_space` which holds the pages to invalidate

*start*

the offset 'from' which to invalidate

*end*

the offset 'to' which to invalidate (inclusive)

## Description

This function only removes the unlocked pages, if you want to remove all the pages of one inode, you must call `truncate_inode_pages`.

`invalidate_mapping_pages` will not block on IO activity. It will not invalidate pages which are dirty, locked, under writeback or mapped into pagetables.

# invalidate\_inode\_pages2\_range

## LINUX

Kernel Hackers Manual June 2013

### Name

`invalidate_inode_pages2_range` — remove range of pages from an `address_space`

### Synopsis

```
int invalidate_inode_pages2_range (struct address_space *  
mapping, pgoff_t start, pgoff_t end);
```

### Arguments

*mapping*

the `address_space`

*start*

the page offset 'from' which to invalidate

*end*

the page offset 'to' which to invalidate (inclusive)

### Description

Any pages which are found to be mapped into pagetables are unmapped prior to invalidation.

Returns `-EBUSY` if any pages could not be invalidated.

# invalidate\_inode\_pages2

## LINUX

Kernel Hackers Manual June 2013

### Name

`invalidate_inode_pages2` — remove all pages from an `address_space`

### Synopsis

```
int invalidate_inode_pages2 (struct address_space * mapping);
```

### Arguments

*mapping*

the `address_space`

### Description

Any pages which are found to be mapped into pagetables are unmapped prior to invalidation.

Returns `-EBUSY` if any pages could not be invalidated.

# truncate\_pagecache

## LINUX

## Name

`truncate_pagecache` — unmap and remove pagecache that has been truncated

## Synopsis

```
void truncate_pagecache (struct inode * inode, loff_t oldsize,  
loff_t newsize);
```

## Arguments

*inode*

inode

*oldsize*

old file size

*newsize*

new file size

## Description

inode's new `i_size` must already be written before `truncate_pagecache` is called.

This function should typically be called before the filesystem releases resources associated with the freed range (eg. deallocates blocks). This way, pagecache will always stay logically coherent with on-disk format, and the filesystem would not have to deal with situations such as `writepage` being called for a page that has already had its underlying blocks deallocated.

# truncate\_setsize

## LINUX

Kernel Hackers Manual June 2013

### Name

`truncate_setsize` — update inode and pagecache for a new file size

### Synopsis

```
void truncate_setsize (struct inode * inode, loff_t newsize);
```

### Arguments

*inode*

inode

*newsize*

new file size

### Description

`truncate_setsize` updates `i_size` and performs pagecache truncation (if necessary) to `newsize`. It will be typically be called from the filesystem's `setattr` function when `ATTR_SIZE` is passed in.

Must be called with `inode_mutex` held and before all filesystem specific block truncation has been performed.



# vmtruncate

## LINUX

Kernel Hackers Manual June 2013

### Name

`vmtruncate` — unmap mappings “freed” by `truncate` syscall

### Synopsis

```
int vmtruncate (struct inode * inode, loff_t newsize);
```

### Arguments

*inode*

inode of the file used

*newsize*

file offset to start truncating

### Description

This function is deprecated and `truncate_setsize` or `truncate_pagecache` should be used instead, together with filesystem specific block truncation.



# Chapter 5. Kernel IPC facilities

## 5.1. IPC utilities

### ipc\_init

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`ipc_init` — initialise IPC subsystem

#### Synopsis

```
int ipc_init ( void );
```

#### Arguments

*void*

no arguments

#### Description

The various system5 IPC resources (semaphores, messages and shared memory) are initialised. A callback routine is registered into the memory hotplug notifier.

## chain

since msgmni scales to lowmem this callback routine will be called upon successful memory add / remove to recompute msgmni.

# ipc\_init\_ids

## LINUX

Kernel Hackers Manual June 2013

## Name

`ipc_init_ids` — initialise IPC identifiers

## Synopsis

```
void ipc_init_ids (struct ipc_ids * ids);
```

## Arguments

*ids*

Identifier set

## Description

Set up the sequence range to use for the ipc identifier range (limited below IPCMNI) then initialise the ids idr.

# ipc\_init\_proc\_interface

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipc_init_proc_interface` — Create a proc interface for sysipc types using a `seq_file` interface.

### Synopsis

```
void ipc_init_proc_interface (const char * path, const char *  
header, int ids, int (*show) (struct seq_file *, void *));
```

### Arguments

*path*

Path in `procfs`

*header*

Banner to be printed at the beginning of the file.

*ids*

ipc id table to iterate.

*show*

show routine.

# ipc\_findkey

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipc_findkey` — find a key in an ipc identifier set

### Synopsis

```
struct kern_ipc_perm * ipc_findkey (struct ipc_ids * ids,  
key_t key);
```

### Arguments

*ids*

Identifier set

*key*

The key to find

### Description

Requires `ipc_ids.rw_mutex` locked. Returns the LOCKED pointer to the ipc structure if found or NULL if not. If key is found ipc points to the owning ipc structure

# ipc\_get\_maxid

## LINUX

## Name

`ipc_get_maxid` — get the last assigned id

## Synopsis

```
int ipc_get_maxid (struct ipc_ids * ids);
```

## Arguments

*ids*

IPC identifier set

## Description

Called with `ipc_ids.rw_mutex` held.

# ipc\_addid

## LINUX

## Name

`ipc_addid` — add an IPC identifier

## Synopsis

```
int ipc_addid (struct ipc_ids * ids, struct kern_ipc_perm *  
new, int size);
```

## Arguments

*ids*

IPC identifier set

*new*

new IPC permission set

*size*

limit for the number of used ids

## Description

Add an entry 'new' to the IPC ids idr. The permissions object is initialised and the first free entry is set up and the id assigned is returned. The 'new' entry is returned in a locked state on success. On failure the entry is not locked and a negative err-code is returned.

Called with ipc\_ids.rw\_mutex held as a writer.

## ipcget\_new

### LINUX

Kernel Hackers Manual June 2013

## Name

ipcget\_new — create a new ipc object



## Synopsis

```
int ipcget_new (struct ipc_namespace * ns, struct ipc_ids *
ids, struct ipc_ops * ops, struct ipc_params * params);
```

## Arguments

*ns*

namespace

*ids*

IPC identifier set

*ops*

the actual creation routine to call

*params*

its parameters

## Description

This routine is called by `sys_msgget`, `sys_semget` and `sys_shmget` when the key is `IPC_PRIVATE`.

## ipc\_check\_perms

**LINUX**

Kernel Hackers Manual June 2013

## Name

`ipc_check_perms` — check security and permissions for an IPC

## Synopsis

```
int ipc_check_perms (struct ipc_namespace * ns, struct
kern_ipc_perm * ipcp, struct ipc_ops * ops, struct ipc_params
* params);
```

## Arguments

*ns*

IPC namespace

*ipcp*

ipc permission set

*ops*

the actual security routine to call

*params*

its parameters

## Description

This routine is called by `sys_msgget`, `sys_semget` and `sys_shmget` when the key is not `IPC_PRIVATE` and that key already exists in the ids IDR.

On success, the IPC id is returned.

It is called with `ipc_ids.rw_mutex` and `ipcp->lock` held.

## ipcget\_public

**LINUX**

## Name

`ipcget_public` — get an ipc object or create a new one

## Synopsis

```
int ipcget_public (struct ipc_namespace * ns, struct ipc_ids *  
ids, struct ipc_ops * ops, struct ipc_params * params);
```

## Arguments

*ns*

namespace

*ids*

IPC identifier set

*ops*

the actual creation routine to call

*params*

its parameters

## Description

This routine is called by `sys_msgget`, `sys_semget` and `sys_shmget` when the key is not `IPC_PRIVATE`. It adds a new entry if the key is not found and does some permission / security checkings if the key is found.

On success, the ipc id is returned.

# ipc\_rmid

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipc_rmid` — remove an IPC identifier

### Synopsis

```
void ipc_rmid (struct ipc_ids * ids, struct kern_ipc_perm *  
ipcp);
```

### Arguments

*ids*

IPC identifier set

*ipcp*

ipc perm structure containing the identifier to remove

### Description

`ipc_ids.rw_mutex` (as a writer) and the spinlock for this ID are held before this function is called, and remain locked on the exit.

# ipc\_alloc

## LINUX

## Name

`ipc_alloc` — allocate ipc space

## Synopsis

```
void* ipc_alloc (int size);
```

## Arguments

*size*

size desired

## Description

Allocate memory from the appropriate pools and return a pointer to it. NULL is returned if the allocation fails

# ipc\_free

## LINUX

## Name

`ipc_free` — free ipc space

## Synopsis

```
void ipc_free (void * ptr, int size);
```

## Arguments

*ptr*

pointer returned by `ipc_alloc`

*size*

size of block

## Description

Free a block created with `ipc_alloc`. The caller must know the size used in the allocation call.

## ipc\_rcu\_alloc

### LINUX

Kernel Hackers Manual June 2013

## Name

`ipc_rcu_alloc` — allocate ipc and rcu space

## Synopsis

```
void* ipc_rcu_alloc (int size);
```

## Arguments

*size*

size desired

## Description

Allocate memory for the rcu header structure + the object. Returns the pointer to the object. NULL is returned if the allocation fails.

# ipc\_schedule\_free

## LINUX

Kernel Hackers Manual June 2013

## Name

`ipc_schedule_free` — free ipc + rcu space

## Synopsis

```
void ipc_schedule_free (struct rcu_head * head);
```

## Arguments

*head*

RCU callback structure for queued work

## Description

Since RCU callback function is called in bh, we need to defer the vfree to `schedule_work`.

# ipc\_immediate\_free

## LINUX

Kernel Hackers Manual June 2013

## Name

`ipc_immediate_free` — free ipc + rcu space

## Synopsis

```
void ipc_immediate_free (struct rcu_head * head);
```

## Arguments

*head*

RCU callback structure that contains pointer to be freed

## Description

Free from the RCU callback context.



# ipcperms

## LINUX

Kernel Hackers Manual June 2013

### Name

ipcperms — check IPC permissions

### Synopsis

```
int ipcperms (struct ipc_namespace * ns, struct kern_ipc_perm  
* ipcp, short flag);
```

### Arguments

*ns*

IPC namespace

*ipcp*

IPC permission set

*flag*

desired permission set.

### Description

Check user, group, other permissions for access to ipc resources. return 0 if allowed

*flag* will most probably be 0 or S\_...UGO from <linux/stat.h>

# kernel\_to\_ipc64\_perm

## LINUX

Kernel Hackers Manual June 2013

### Name

`kernel_to_ipc64_perm` — convert kernel ipc permissions to user

### Synopsis

```
void kernel_to_ipc64_perm (struct kern_ipc_perm * in, struct  
ipc64_perm * out);
```

### Arguments

*in*

kernel permissions

*out*

new style IPC permissions

### Description

Turn the kernel object *in* into a set of permissions descriptions for returning to userspace (*out*).

# ipc64\_perm\_to\_ipc\_perm

## LINUX

## Name

`ipc64_perm_to_ipc_perm` — convert new ipc permissions to old

## Synopsis

```
void ipc64_perm_to_ipc_perm (struct ipc64_perm * in, struct  
ipc_perm * out);
```

## Arguments

*in*

new style IPC permissions

*out*

old style IPC permissions

## Description

Turn the new style permissions object *in* into a compatibility object and store it into the *out* pointer.

## ipc\_lock

**LINUX**

## Name

`ipc_lock` — Lock an ipc structure without `rw_mutex` held

## Synopsis

```
struct kern_ipc_perm * ipc_lock (struct ipc_ids * ids, int
id);
```

## Arguments

*ids*

IPC identifier set

*id*

ipc id to look for

## Description

Look for an `id` in the `ipc_ids idr` and lock the associated ipc object.

The ipc object is locked on exit.

## ipcget

**LINUX**

## Name

`ipcget` — Common `sys_*get` code

## Synopsis

```
int ipcget (struct ipc_namespace * ns, struct ipc_ids * ids,  
            struct ipc_ops * ops, struct ipc_params * params);
```

## Arguments

*ns*

namespace

*ids*

IPC identifier set

*ops*

operations to be called on ipc object creation, permission checks and further checks

*params*

the parameters needed by the previous operations.

## Description

Common routine called by `sys_msgget`, `sys_semget` and `sys_shmget`.

# ipc\_update\_perm

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipc_update_perm` — update the permissions of an IPC.

### Synopsis

```
void ipc_update_perm (struct ipc64_perm * in, struct  
kern_ipc_perm * out);
```

### Arguments

*in*

the permission given as input.

*out*

the permission of the ipc to set.

# ipcctl\_pre\_down

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipcctl_pre_down` — retrieve an ipc and check permissions for some  
IPC\_XXX cmd

## Synopsis

```
struct kern_ipc_perm * ipcctl_pre_down (struct ipc_namespace *
ns, struct ipc_ids * ids, int id, int cmd, struct ipc64_perm *
perm, int extra_perm);
```

## Arguments

*ns*

the ipc namespace

*ids*

the table of ids where to look for the ipc

*id*

the id of the ipc to retrieve

*cmd*

the cmd to check

*perm*

the permission to set

*extra\_perm*

one extra permission parameter used by msq

## Description

This function does some common audit and permissions check for some IPC\_XXX cmd and is called from semctl\_down, shmctl\_down and msgctl\_down. It must be called without any lock held and - retrieves the ipc with the given id in the given table. - performs some audit and permission check, depending on the given cmd - returns the ipc with both ipc and rw\_mutex locks held in case of success or an err-code without any lock held otherwise.

# ipc\_parse\_version

## LINUX

Kernel Hackers Manual June 2013

### Name

`ipc_parse_version` — IPC call version

### Synopsis

```
int ipc_parse_version (int * cmd);
```

### Arguments

*cmd*

pointer to command

### Description

Return `IPC_64` for new style IPC and `IPC_OLD` for old style IPC. The *cmd* value is turned from an encoding command and version into just the command code.



# Chapter 6. FIFO Buffer

## 6.1. kfifo interface

### DECLARE\_KFIFO\_PTR

**LINUX**

Kernel Hackers Manual June 2013

#### Name

DECLARE\_KFIFO\_PTR — macro to declare a fifo pointer object

#### Synopsis

```
DECLARE_KFIFO_PTR ( fifo, type );
```

#### Arguments

*fifo*

name of the declared fifo

*type*

type of the fifo elements

### DECLARE\_KFIFO

**LINUX**

## Name

`DECLARE_KFIFO` — macro to declare a fifo object

## Synopsis

```
DECLARE_KFIFO ( fifo, type, size );
```

## Arguments

*fifo*

name of the declared fifo

*type*

type of the fifo elements

*size*

the number of elements in the fifo, this must be a power of 2

# INIT\_KFIFO

## LINUX

## Name

`INIT_KFIFO` — Initialize a fifo declared by `DECLARE_KFIFO`

## Synopsis

```
INIT_KFIFO ( fifo);
```

## Arguments

*fifo*

name of the declared fifo datatype

# DEFINE\_KFIFO

## LINUX

Kernel Hackers Manual June 2013

## Name

DEFINE\_KFIFO — macro to define and initialize a fifo

## Synopsis

```
DEFINE_KFIFO ( fifo,  type,  size);
```

## Arguments

*fifo*

name of the declared fifo datatype

*type*

type of the fifo elements

*size*

the number of elements in the fifo, this must be a power of 2

## Note

the macro can be used for global and local fifo data type variables.

# kfifo\_initialized

## LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_initialized` — Check if the fifo is initialized

## Synopsis

```
kfifo_initialized ( fifo );
```

## Arguments

*fifo*

address of the fifo to check

## Description

Return `true` if `fifo` is initialized, otherwise `false`. Assumes the `fifo` was 0 before.

## kfifo\_esize

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_esize` — returns the size of the element managed by the `fifo`

## Synopsis

```
kfifo_esize ( fifo );
```

## Arguments

*fifo*

address of the `fifo` to be used

## kfifo\_resize

### LINUX

## Name

`kfifo_recsz` — returns the size of the record length field

## Synopsis

```
kfifo_recsz ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

# kfifo\_size

## LINUX

## Name

`kfifo_size` — returns the size of the fifo in elements

## Synopsis

```
kfifo_size ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## kfifo\_reset

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_reset` — removes the entire fifo content

## Synopsis

```
kfifo_reset ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## Note

usage of `kfifo_reset` is dangerous. It should be only called when the fifo is excluded locked or when it is secured that no other thread is accessing the fifo.

# kfifo\_reset\_out

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_reset_out` — skip fifo content

### Synopsis

```
kfifo_reset_out ( fifo );
```

### Arguments

*fifo*

address of the fifo to be used

### Note

The usage of `kfifo_reset_out` is safe until it will be only called from the reader thread and there is only one concurrent reader. Otherwise it is dangerous and must be handled in the same way as `kfifo_reset`.

# kfifo\_len

## LINUX



## Name

`kfifo_len` — returns the number of used elements in the fifo

## Synopsis

```
kfifo_len ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

# kfifo\_is\_empty

## LINUX

## Name

`kfifo_is_empty` — returns true if the fifo is empty

## Synopsis

```
kfifo_is_empty ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## kfifo\_is\_full

### LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_is_full` — returns true if the fifo is full

### Synopsis

```
kfifo_is_full ( fifo );
```

### Arguments

*fifo*

address of the fifo to be used

## kfifo\_avail

### LINUX

## Name

`kfifo_avail` — returns the number of unused elements in the fifo

## Synopsis

```
kfifo_avail ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## kfifo\_skip

### LINUX

## Name

`kfifo_skip` — skip output data

## Synopsis

```
kfifo_skip ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## kfifo\_peek\_len

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_peek_len` — gets the size of the next fifo record

## Synopsis

```
kfifo_peek_len ( fifo );
```

## Arguments

*fifo*

address of the fifo to be used

## Description

This function returns the size of the next fifo record in number of bytes.

# kfifo\_alloc

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_alloc` — dynamically allocates a new fifo buffer

### Synopsis

```
kfifo_alloc ( fifo, size, gfp_mask );
```

### Arguments

*fifo*

pointer to the fifo

*size*

the number of elements in the fifo, this must be a power of 2

*gfp\_mask*

get\_free\_pages mask, passed to `kmalloc`

### Description

This macro dynamically allocates a new fifo buffer.

The number of elements will be rounded-up to a power of 2. The fifo will be released with `kfifo_free`. Return 0 if no error, otherwise an error code.

# kfifo\_free

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_free` — frees the fifo

### Synopsis

```
kfifo_free ( fifo );
```

### Arguments

*fifo*

the fifo to be freed

# kfifo\_init

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_init` — initialize a fifo using a preallocated buffer

## Synopsis

```
kfifo_init ( fifo, buffer, size );
```

## Arguments

*fifo*

the fifo to assign the buffer

*buffer*

the preallocated buffer to be used

*size*

the size of the internal buffer, this have to be a power of 2

## Description

This macro initialize a fifo using a preallocated buffer.

The numer of elements will be rounded-up to a power of 2. Return 0 if no error, otherwise an error code.

## kfifo\_put

**LINUX**

Kernel Hackers Manual June 2013

## Name

`kfifo_put` — put data into the fifo

## Synopsis

```
kfifo_put ( fifo, val );
```

## Arguments

*fifo*

address of the fifo to be used

*val*

the data to be added

## Description

This macro copies the given value into the fifo. It returns 0 if the fifo was full. Otherwise it returns the number processed elements.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_get

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_get` — get data from the fifo



## Synopsis

```
kfifo_get ( fifo, val );
```

## Arguments

*fifo*

address of the fifo to be used

*val*

the var where to store the data to be added

## Description

This macro reads the data from the fifo. It returns 0 if the fifo was empty. Otherwise it returns the number processed elements.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_peek

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_peek` — get data from the fifo without removing

## Synopsis

```
kfifo_peek ( fifo, val );
```

## Arguments

*fifo*

address of the fifo to be used

*val*

the var where to store the data to be added

## Description

This reads the data from the fifo without removing it from the fifo. It returns 0 if the fifo was empty. Otherwise it returns the number processed elements.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_in

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_in` — put data into the fifo

## Synopsis

```
kfifo_in ( fifo, buf, n );
```

## Arguments

*fifo*

address of the fifo to be used

*buf*

the data to be added

*n*

number of elements to be added

## Description

This macro copies the given buffer into the fifo and returns the number of copied elements.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_in\_spinlocked

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_in_spinlocked` — put data into the fifo using a spinlock for locking

## Synopsis

```
kfifo_in_spinlocked ( fifo, buf, n, lock );
```

## Arguments

*fifo*

address of the fifo to be used

*buf*

the data to be added

*n*

number of elements to be added

*lock*

pointer to the spinlock to use for locking

## Description

This macro copies the given values buffer into the fifo and returns the number of copied elements.

## kfifo\_out

**LINUX**

Kernel Hackers Manual June 2013

## Name

`kfifo_out` — get data from the fifo

## Synopsis

```
kfifo_out ( fifo, buf, n );
```

## Arguments

*fifo*

address of the fifo to be used

*buf*

pointer to the storage buffer

*n*

max. number of elements to get

## Description

This macro get some data from the fifo and return the numbers of elements copied.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_out\_spinlocked

### LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_out_spinlocked` — get data from the fifo using a spinlock for locking

## Synopsis

```
kfifo_out_spinlocked ( fifo, buf, n, lock );
```

## Arguments

*fifo*

address of the fifo to be used

*buf*

pointer to the storage buffer

*n*

max. number of elements to get

*lock*

pointer to the spinlock to use for locking

## Description

This macro get the data from the fifo and return the numbers of elements copied.

## kfifo\_from\_user

**LINUX**

Kernel Hackers Manual June 2013

## Name

`kfifo_from_user` — puts some data from user space into the fifo

## Synopsis

```
kfifo_from_user ( fifo, from, len, copied );
```

## Arguments

*fifo*

address of the fifo to be used

*from*

pointer to the data to be added

*len*

the length of the data to be added

*copied*

pointer to output variable to store the number of copied bytes

## Description

This macro copies at most *len* bytes from the *from* into the fifo, depending of the available space and returns -EFAULT/0.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.

## kfifo\_to\_user

**LINUX**

## Name

`kfifo_to_user` — copies data from the fifo into user space

## Synopsis

```
kfifo_to_user ( fifo, to, len, copied );
```

## Arguments

*fifo*

address of the fifo to be used

*to*

where the data must be copied

*len*

the size of the destination buffer

*copied*

pointer to output variable to store the number of copied bytes

## Description

This macro copies at most *len* bytes from the fifo into the *to* buffer and returns `-EFAULT/0`.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.



# kfifo\_dma\_in\_prepare

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_dma_in_prepare` — setup a scatterlist for DMA input

### Synopsis

```
kfifo_dma_in_prepare ( fifo, sgl, nents, len );
```

### Arguments

*fifo*

address of the fifo to be used

*sgl*

pointer to the scatterlist array

*nents*

number of entries in the scatterlist array

*len*

number of elements to transfer

### Description

This macro fills a scatterlist for DMA input. It returns the number entries in the scatterlist array.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macros.

# kfifo\_dma\_in\_finish

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_dma_in_finish` — finish a DMA IN operation

### Synopsis

```
kfifo_dma_in_finish ( fifo, len );
```

### Arguments

*fifo*

address of the fifo to be used

*len*

number of bytes to received

### Description

This macro finish a DMA IN operation. The in counter will be updated by the `len` parameter. No error checking will be done.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macros.

# kfifo\_dma\_out\_prepare

## LINUX

Kernel Hackers Manual June 2013

## Name

`kfifo_dma_out_prepare` — setup a scatterlist for DMA output

## Synopsis

```
kfifo_dma_out_prepare ( fifo, sgl, nents, len );
```

## Arguments

*fifo*

address of the fifo to be used

*sgl*

pointer to the scatterlist array

*nents*

number of entries in the scatterlist array

*len*

number of elements to transfer

## Description

This macro fills a scatterlist for DMA output which at most *len* bytes to transfer. It returns the number entries in the scatterlist array. A zero means there is no space available and the scatterlist is not filled.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macros.

# kfifo\_dma\_out\_finish

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_dma_out_finish` — finish a DMA OUT operation

### Synopsis

```
kfifo_dma_out_finish ( fifo, len );
```

### Arguments

*fifo*

address of the fifo to be used

*len*

number of bytes transferd

### Description

This macro finish a DMA OUT operation. The out counter will be updated by the `len` parameter. No error checking will be done.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macros.

# kfifo\_out\_peek

## LINUX

Kernel Hackers Manual June 2013

### Name

`kfifo_out_peek` — gets some data from the fifo

### Synopsis

```
kfifo_out_peek ( fifo, buf, n );
```

### Arguments

*fifo*

address of the fifo to be used

*buf*

pointer to the storage buffer

*n*

max. number of elements to get

### Description

This macro get the data from the fifo and return the numbers of elements copied. The data is not removed from the fifo.

Note that with only one concurrent reader and one concurrent writer, you don't need extra locking to use these macro.



# Chapter 7. relay interface support

Relay interface support is designed to provide an efficient mechanism for tools and facilities to relay large amounts of data from kernel space to user space.

## 7.1. relay interface

### relay\_buf\_full

#### LINUX

Kernel Hackers Manual June 2013

#### Name

`relay_buf_full` — boolean, is the channel buffer full?

#### Synopsis

```
int relay_buf_full (struct rchan_buf * buf);
```

#### Arguments

*buf*

channel buffer

#### Description

Returns 1 if the buffer is full, 0 otherwise.

# relay\_reset

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_reset` — reset the channel

### Synopsis

```
void relay_reset (struct rchan * chan);
```

### Arguments

*chan*

the channel

### Description

This has the effect of erasing all data from all channel buffers and restarting the channel in its initial state. The buffers are not freed, so any mappings are still in effect.

NOTE. Care should be taken that the channel isn't actually being used by anything when this call is made.

# relay\_open

## LINUX



## Name

`relay_open` — create a new relay channel

## Synopsis

```
struct rchan * relay_open (const char * base_filename, struct  
dentry * parent, size_t subbuf_size, size_t n_subbufs, struct  
rchan_callbacks * cb, void * private_data);
```

## Arguments

*base\_filename*

base name of files to create, `NULL` for buffering only

*parent*

dentry of parent directory, `NULL` for root directory or buffer

*subbuf\_size*

size of sub-buffers

*n\_subbufs*

number of sub-buffers

*cb*

client callback functions

*private\_data*

user-defined data

## Description

Returns channel pointer if successful, `NULL` otherwise.

Creates a channel buffer for each cpu using the sizes and attributes specified. The created channel buffer files will be named `base_filename0...base_filenameN-1`. File permissions will be `S_IRUSR`.

## relay\_switch\_subbuf

### LINUX

Kernel Hackers Manual June 2013

### Name

`relay_switch_subbuf` — switch to a new sub-buffer

### Synopsis

```
size_t relay_switch_subbuf (struct rchan_buf * buf, size_t  
length);
```

### Arguments

*buf*

channel buffer

*length*

size of current event

### Description

Returns either the length passed in or 0 if full.

Performs sub-buffer-switch tasks such as invoking callbacks, updating padding counts, waking up readers, etc.

# relay\_subbufs\_consumed

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_subbufs_consumed` — update the buffer's sub-buffers-consumed count

### Synopsis

```
void relay_subbufs_consumed (struct rchan * chan, unsigned int  
cpu, size_t subbufs_consumed);
```

### Arguments

*chan*

the channel

*cpu*

the cpu associated with the channel buffer to update

*subbufs\_consumed*

number of sub-buffers to add to current buf's count

### Description

Adds to the channel buffer's consumed sub-buffer count. `subbufs_consumed` should be the number of sub-buffers newly consumed, not the total consumed.

NOTE. Kernel clients don't need to call this function if the channel mode is 'overwrite'.

## relay\_close

### LINUX

Kernel Hackers Manual June 2013

### Name

`relay_close` — close the channel

### Synopsis

```
void relay_close (struct rchan * chan);
```

### Arguments

*chan*

the channel

### Description

Closes all channel buffers and frees the channel.

# relay\_flush

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_flush` — close the channel

### Synopsis

```
void relay_flush (struct rchan * chan);
```

### Arguments

*chan*

the channel

### Description

Flushes all channel buffers, i.e. forces buffer switch.

# relay\_mmap\_buf

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_mmap_buf` — mmap channel buffer to process address space

## Synopsis

```
int relay_mmap_buf (struct rchan_buf * buf, struct
vm_area_struct * vma);
```

## Arguments

*buf*

relay channel buffer

*vma*

vm\_area\_struct describing memory to be mapped

## Description

Returns 0 if ok, negative on error

Caller should already have grabbed mmap\_sem.

## relay\_alloc\_buf

### LINUX

Kernel Hackers Manual June 2013

## Name

relay\_alloc\_buf — allocate a channel buffer

## Synopsis

```
void * relay_alloc_buf (struct rchan_buf * buf, size_t *
size);
```

## Arguments

*buf*

the buffer struct

*size*

total size of the buffer

## Description

Returns a pointer to the resulting buffer, `NULL` if unsuccessful. The passed in size will get page aligned, if it isn't already.

# relay\_create\_buf

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_create_buf` — allocate and initialize a channel buffer

## Synopsis

```
struct rchan_buf * relay_create_buf (struct rchan * chan);
```

## Arguments

*chan*

the relay channel

## Description

Returns channel buffer if successful, `NULL` otherwise.

# relay\_destroy\_channel

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_destroy_channel` — free the channel struct

## Synopsis

```
void relay_destroy_channel (struct kref * kref);
```

## Arguments

*kref*

target kernel reference that contains the relay channel



## Description

Should only be called from `kref_put`.

# relay\_destroy\_buf

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_destroy_buf` — destroy an `rchan_buf` struct and associated buffer

## Synopsis

```
void relay_destroy_buf (struct rchan_buf * buf);
```

## Arguments

*buf*

the buffer struct

# relay\_remove\_buf

## LINUX

## Name

`relay_remove_buf` — remove a channel buffer

## Synopsis

```
void relay_remove_buf (struct kref * kref);
```

## Arguments

*kref*

target kernel reference that contains the relay buffer

## Description

Removes the file from the filesystem, which also frees the `rchan_buf_struct` and the channel buffer. Should only be called from `kref_put`.

# relay\_buf\_empty

## LINUX

## Name

`relay_buf_empty` — boolean, is the channel buffer empty?

## Synopsis

```
int relay_buf_empty (struct rchan_buf * buf);
```

## Arguments

*buf*

channel buffer

## Description

Returns 1 if the buffer is empty, 0 otherwise.

# wakeup\_readers

## LINUX

Kernel Hackers Manual June 2013

## Name

`wakeup_readers` — wake up readers waiting on a channel

## Synopsis

```
void wakeup_readers (unsigned long data);
```

## Arguments

*data*

contains the channel buffer

## Description

This is the timer function used to defer reader waking.

## \_\_relay\_reset

### LINUX

Kernel Hackers Manual June 2013

## Name

`__relay_reset` — reset a channel buffer

## Synopsis

```
void __relay_reset (struct rchan_buf * buf, unsigned int  
init);
```

## Arguments

*buf*

the channel buffer

*init*

1 if this is a first-time initialization

## Description

See `relay_reset` for description of effect.

# relay\_close\_buf

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_close_buf` — close a channel buffer

## Synopsis

```
void relay_close_buf (struct rchan_buf * buf);
```

## Arguments

*buf*

channel buffer

## Description

Marks the buffer finalized and restores the default callbacks. The channel buffer and channel buffer data structure are then freed automatically when the last reference is given up.

# relay\_hotcpu\_callback

## LINUX

Kernel Hackers Manual June 2013

### Name

relay\_hotcpu\_callback — CPU hotplug callback

### Synopsis

```
int __cpuinit relay_hotcpu_callback (struct notifier_block *  
nb, unsigned long action, void * hcpu);
```

### Arguments

*nb*

notifier block

*action*

hotplug action to take

*hcpu*

CPU number

### Description

Returns the success/failure of the operation. (NOTIFY\_OK, NOTIFY\_BAD)

# relay\_late\_setup\_files

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_late_setup_files` — triggers file creation

### Synopsis

```
int relay_late_setup_files (struct rchan * chan, const char *  
base_filename, struct dentry * parent);
```

### Arguments

*chan*

channel to operate on

*base\_filename*

base name of files to create

*parent*

dentry of parent directory, `NULL` for root directory

### Description

Returns 0 if successful, non-zero otherwise.

Use to setup files for a previously buffer-only channel. Useful to do early tracing in kernel, before VFS is up, for example.

# relay\_file\_open

## LINUX

Kernel Hackers Manual June 2013

### Name

`relay_file_open` — open file op for relay files

### Synopsis

```
int relay_file_open (struct inode * inode, struct file *  
filp);
```

### Arguments

*inode*

the inode

*filp*

the file

### Description

Increments the channel buffer refcount.

# relay\_file\_mmap

## LINUX



## Name

`relay_file_mmap` — mmap file op for relay files

## Synopsis

```
int relay_file_mmap (struct file * filp, struct vm_area_struct
* vma);
```

## Arguments

*filp*

the file

*vma*

the vma describing what to map

## Description

Calls upon `relay_mmap_buf` to map the file into user space.

# relay\_file\_poll

## LINUX

## Name

`relay_file_poll` — poll file op for relay files

## Synopsis

```
unsigned int relay_file_poll (struct file * filp, poll_table *  
wait);
```

## Arguments

*filp*

the file

*wait*

poll table

## Description

Poll implementation.

# relay\_file\_release

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_file_release` — release file op for relay files

## Synopsis

```
int relay_file_release (struct inode * inode, struct file *  
filp);
```

## Arguments

*inode*

the inode

*filp*

the file

## Description

Decrements the channel refcount, as the filesystem is no longer using it.

# relay\_file\_read\_subbuf\_avail

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_file_read_subbuf_avail` — return bytes available in sub-buffer

## Synopsis

```
size_t relay_file_read_subbuf_avail (size_t read_pos, struct  
rchan_buf * buf);
```

## Arguments

*read\_pos*

file read position

*buf*

relay channel buffer

# relay\_file\_read\_start\_pos

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_file_read_start_pos` — find the first available byte to read

## Synopsis

```
size_t relay_file_read_start_pos (size_t read_pos, struct  
rchan_buf * buf);
```

## Arguments

*read\_pos*

file read position

*buf*

relay channel buffer

## Description

If the *read\_pos* is in the middle of padding, return the position of the first actually available byte, otherwise return the original value.

# relay\_file\_read\_end\_pos

## LINUX

Kernel Hackers Manual June 2013

## Name

`relay_file_read_end_pos` — return the new read position

## Synopsis

```
size_t relay_file_read_end_pos (struct rchan_buf * buf, size_t
read_pos, size_t count);
```

## Arguments

*buf*

relay channel buffer

*read\_pos*

file read position

*count*

number of bytes to be read



# Chapter 8. Module Support

## 8.1. Module Loading

### `__request_module`

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`__request_module` — try to load a kernel module

#### Synopsis

```
int __request_module (bool wait, const char * fmt, ...);
```

#### Arguments

*wait*

wait (or not) for the operation to complete

*fmt*

printf style format string for the name of the module @...: arguments as specified in the format string

...

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\_setup

## LINUX

Kernel Hackers Manual June 2013

## Name

`call_usermodehelper_setup` — prepare to call a usermode helper

## Synopsis

```
struct subprocess_info * call_usermodehelper_setup (char *  
path, char ** argv, char ** envp, gfp_t gfp_mask);
```

## Arguments

*path*

path to usermode executable

*argv*

arg vector for process



*envp*

environment for process

*gfp\_mask*

gfp mask for memory allocation

## Description

Returns either `NULL` on allocation failure, or a `subprocess_info` structure. This should be passed to `call_usermodehelper_exec` to exec the process and free the structure.

# call\_usermodehelper\_setfns

## LINUX

Kernel Hackers Manual June 2013

## Name

`call_usermodehelper_setfns` — set a cleanup/init function

## Synopsis

```
void call_usermodehelper_setfns (struct subprocess_info *
info, int (*init) (struct subprocess_info *info, struct cred
*new), void (*cleanup) (struct subprocess_info *info), void *
data);
```

## Arguments

*info*

a subprocess\_info returned by call\_usermodehelper\_setup

*init*

an init function

*cleanup*

a cleanup function

*data*

arbitrary context sensitive data

## Description

The init function is used to customize the helper process prior to exec. A non-zero return code causes the process to error out, exit, and return the failure to the calling process

The cleanup function is just before the subprocess\_info is about to be freed. This can be used for freeing the argv and envp. The Function must be runnable in either a process context or the context in which call\_usermodehelper\_exec is called.

# call\_usermodehelper\_exec

**LINUX**

Kernel Hackers Manual June 2013

## Name

call\_usermodehelper\_exec — start a usermode application

## Synopsis

```
int call_usermodehelper_exec (struct subprocess_info *
sub_info, enum umh_wait wait);
```

## Arguments

*sub\_info*

information about the subprocessa

*wait*

wait for the application to finish and return status. when -1 don't wait at all, but you get no useful error back when the program couldn't be exec'ed. This makes it safe to call from interrupt context.

## 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).

## 8.2. Inter Module support

Refer to the file kernel/module.c for more information.



# Chapter 9. Hardware Interfaces

## 9.1. Interrupt Handling

### synchronize\_irq

#### LINUX

Kernel Hackers Manual June 2013

#### Name

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

#### Synopsis

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

#### Arguments

*irq*

interrupt number to wait for

#### Description

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

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

# irq\_set\_affinity\_notifier

## LINUX

Kernel Hackers Manual June 2013

### Name

`irq_set_affinity_notifier` — control notification of IRQ affinity changes

### Synopsis

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

### Arguments

*irq*

Interrupt for which to enable/disable notification

*notify*

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

### Description

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

# disable\_irq\_nosync

## LINUX

Kernel Hackers Manual June 2013

### Name

`disable_irq_nosync` — disable an irq without waiting

### Synopsis

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

### Arguments

*irq*

Interrupt to disable

### Description

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

This function may be called from IRQ context.

# disable\_irq

## LINUX

## Name

`disable_irq` — disable an irq and wait for completion

## Synopsis

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

## Arguments

*irq*

Interrupt to disable

## Description

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

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

## `enable_irq`

### LINUX

## Name

`enable_irq` — enable handling of an irq



## Synopsis

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

## Arguments

*irq*

Interrupt to enable

## Description

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

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

## irq\_set\_irq\_wake

### LINUX

Kernel Hackers Manual June 2013

## Name

`irq_set_irq_wake` — control irq power management wakeup

## Synopsis

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

## Arguments

*irq*

interrupt to control

*on*

enable/disable power management wakeup

## Description

Enable/disable power management wakeup mode, which is disabled by default.

Enables and disables must match, just as they match for non-wakeup mode support.

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

## setup\_irq

### LINUX

Kernel Hackers Manual June 2013

### Name

setup\_irq — setup an interrupt

### Synopsis

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

## Arguments

*irq*

Interrupt line to setup

*act*

irqaction for the interrupt

## Description

Used to statically setup interrupts in the early boot process.

## remove\_irq

### LINUX

Kernel Hackers Manual June 2013

## Name

`remove_irq` — free an interrupt

## Synopsis

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

## Arguments

*irq*

Interrupt line to free

*act*

irqaction for the interrupt

## Description

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

## free\_irq

### LINUX

Kernel Hackers Manual June 2013

## Name

`free_irq` — free an interrupt allocated with `request_irq`

## Synopsis

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

## Arguments

*irq*

Interrupt line to free

*dev\_id*

Device identity to free

## Description

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

This function must not be called from interrupt context.

# request\_threaded\_irq

## LINUX

Kernel Hackers Manual June 2013

## Name

`request_threaded_irq` — allocate an interrupt line

## Synopsis

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

## Arguments

*irq*

Interrupt line to allocate

*handler*

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

*thread\_fn*

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

*irqflags*

Interrupt type flags

*devname*

An ascii name for the claiming device

*dev\_id*

A cookie passed back to the handler function

## Description

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

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

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

If your interrupt is shared you must pass a non NULL *dev\_id* as this is required when freeing the interrupt.

## Flags

`IRQF_SHARED` Interrupt is shared `IRQF_SAMPLE_RANDOM` The interrupt can be used for entropy `IRQF_TRIGGER_*` Specify active edge(s) or level

# request\_any\_context\_irq

## LINUX

Kernel Hackers Manual June 2013

### Name

`request_any_context_irq` — allocate an interrupt line

### Synopsis

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

### Arguments

*irq*

Interrupt line to allocate

*handler*

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

*flags*

Interrupt type flags

*name*

An ascii name for the claiming device

*dev\_id*

A cookie passed back to the handler function

## Description

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

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

## 9.2. DMA Channels

### request\_dma

#### LINUX

Kernel Hackers Manual June 2013

#### Name

`request_dma` — request and reserve a system DMA channel

#### Synopsis

```
int request_dma (unsigned int dmanr, const char * device_id);
```

#### Arguments

*dmanr*

DMA channel number

*device\_id*

reserving device ID string, used in `/proc/dma`



## free\_dma

### LINUX

Kernel Hackers Manual June 2013

#### Name

`free_dma` — free a reserved system DMA channel

#### Synopsis

```
void free_dma (unsigned int dmanr);
```

#### Arguments

*dmanr*

DMA channel number

## 9.3. Resources Management

### request\_resource\_conflict

### LINUX

## Name

`request_resource_conflict` — request and reserve an I/O or memory resource

## Synopsis

```
struct resource * request_resource_conflict (struct resource *  
root, struct resource * new);
```

## Arguments

*root*

root resource descriptor

*new*

resource descriptor desired by caller

## Description

Returns 0 for success, conflict resource on error.

# reallocate\_resource

**LINUX**

## Name

`reallocate_resource` — allocate a slot in the resource tree given range & alignment. The resource will be relocated if the new size cannot be reallocated in the current location.

## Synopsis

```
int reallocate_resource (struct resource * root, struct  
resource * old, resource_size_t newsize, struct  
resource_constraint * constraint);
```

## Arguments

*root*

root resource descriptor

*old*

resource descriptor desired by caller

*newsize*

new size of the resource descriptor

*constraint*

the size and alignment constraints to be met.

## insert\_resource\_conflict

**LINUX**

## Name

`insert_resource_conflict` — Inserts resource in the resource tree

## Synopsis

```
struct resource * insert_resource_conflict (struct resource *  
parent, struct resource * new);
```

## Arguments

*parent*

parent of the new resource

*new*

new resource to insert

## Description

Returns 0 on success, conflict resource if the resource can't be inserted.

This function is equivalent to `request_resource_conflict` when no conflict happens. If a conflict happens, and the conflicting resources entirely fit within the range of the new resource, then the new resource is inserted and the conflicting resources become children of the new resource.

## `insert_resource`

**LINUX**

## Name

`insert_resource` — Inserts a resource in the resource tree

## Synopsis

```
int insert_resource (struct resource * parent, struct resource
* new);
```

## Arguments

*parent*

parent of the new resource

*new*

new resource to insert

## Description

Returns 0 on success, -EBUSY if the resource can't be inserted.

# insert\_resource\_expand\_to\_fit

## LINUX

## Name

`insert_resource_expand_to_fit` — Insert a resource into the resource

tree

## Synopsis

```
void insert_resource_expand_to_fit (struct resource * root,  
struct resource * new);
```

## Arguments

*root*

root resource descriptor

*new*

new resource to insert

## Description

Insert a resource into the resource tree, possibly expanding it in order to make it encompass any conflicting resources.

# resource\_alignment

**LINUX**

Kernel Hackers Manual June 2013

## Name

`resource_alignment` — calculate resource's alignment

## Synopsis

```
resource_size_t resource_alignment (struct resource * res);
```

## Arguments

*res*

resource pointer

## Description

Returns alignment on success, 0 (invalid alignment) on failure.

# request\_resource

## LINUX

Kernel Hackers Manual June 2013

## Name

`request_resource` — request and reserve an I/O or memory resource

## Synopsis

```
int request_resource (struct resource * root, struct resource  
* new);
```

## Arguments

*root*

root resource descriptor

*new*

resource descriptor desired by caller

## Description

Returns 0 for success, negative error code on error.

# release\_resource

## LINUX

Kernel Hackers Manual June 2013

## Name

`release_resource` — release a previously reserved resource

## Synopsis

```
int release_resource (struct resource * old);
```

## Arguments

*old*

resource pointer



# allocate\_resource

## LINUX

Kernel Hackers Manual June 2013

### Name

`allocate_resource` — allocate empty slot in the resource tree given range & alignment. The resource will be reallocated with a new size if it was already allocated

### Synopsis

```
int allocate_resource (struct resource * root, struct resource
* new, resource_size_t size, resource_size_t min,
resource_size_t max, resource_size_t align, resource_size_t
(*alignf) (void *, const struct resource *, resource_size_t,
resource_size_t), void * alignf_data);
```

### Arguments

*root*

root resource descriptor

*new*

resource descriptor desired by caller

*size*

requested resource region size

*min*

minimum size to allocate

*max*

maximum size to allocate

*align*

alignment requested, in bytes

*alignf*

alignment function, optional, called if not NULL

*alignf\_data*

arbitrary data to pass to the *alignf* function

## adjust\_resource

### LINUX

Kernel Hackers Manual June 2013

### Name

`adjust_resource` — modify a resource's start and size

### Synopsis

```
int adjust_resource (struct resource * res, resource_size_t  
start, resource_size_t size);
```

### Arguments

*res*

resource to modify

*start*

new start value

*size*

new size

## Description

Given an existing resource, change its start and size to match the arguments.

Returns 0 on success, -EBUSY if it can't fit. Existing children of the resource are assumed to be immutable.

## \_\_request\_region

### LINUX

Kernel Hackers Manual June 2013

## Name

`__request_region` — create a new busy resource region

## Synopsis

```
struct resource * __request_region (struct resource * parent,
resource_size_t start, resource_size_t n, const char * name,
int flags);
```

## Arguments

*parent*

parent resource descriptor

*start*

resource start address

*n*

resource region size

*name*

reserving caller's ID string

*flags*

IO resource flags

## **\_\_check\_region**

**LINUX**

Kernel Hackers Manual June 2013

### **Name**

`__check_region` — check if a resource region is busy or free

### **Synopsis**

```
int __check_region (struct resource * parent, resource_size_t  
start, resource_size_t n);
```

### **Arguments**

*parent*

parent resource descriptor

*start*

resource start address

*n*

resource region size

## Description

Returns 0 if the region is free at the moment it is checked, returns `-EBUSY` if the region is busy.

## NOTE

This function is deprecated because its use is racy. Even if it returns 0, a subsequent call to `request_region` may fail because another driver etc. just allocated the region. Do NOT use it. It will be removed from the kernel.

# \_\_release\_region

## LINUX

Kernel Hackers Manual June 2013

## Name

`__release_region` — release a previously reserved resource region

## Synopsis

```
void __release_region (struct resource * parent,
resource_size_t start, resource_size_t n);
```

## Arguments

*parent*

parent resource descriptor

*start*

resource start address

*n*

resource region size

## Description

The described resource region must match a currently busy region.

## 9.4. MTRR Handling

### mtrr\_add

**LINUX**

Kernel Hackers Manual June 2013

### Name

`mtrr_add` — Add a memory type region

### Synopsis

```
int mtrr_add (unsigned long base, unsigned long size, unsigned  
int type, bool 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

## LINUX

Kernel Hackers Manual June 2013

### 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.

## 9.5. PCI Support Library

### pci\_bus\_max\_busnr

#### LINUX

Kernel Hackers Manual June 2013

#### Name

`pci_bus_max_busnr` — returns maximum PCI bus number of given bus' children

#### Synopsis

```
unsigned char 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\_find\_capability

## LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

Kernel Hackers Manual June 2013

### 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\_ext\_capability

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_find_ext_capability` — Find an extended capability

### Synopsis

```
int pci_find_ext_capability (struct pci_dev * dev, int cap);
```

### Arguments

*dev*

PCI device to query

*cap*

capability code

### Description

Returns the address of the requested extended capability structure within the device's PCI configuration space or 0 if the device does not support it. Possible values for *cap*:

PCI\_EXT\_CAP\_ID\_ERR Advanced Error Reporting PCI\_EXT\_CAP\_ID\_VC Virtual Channel  
PCI\_EXT\_CAP\_ID\_DSN Device Serial Number PCI\_EXT\_CAP\_ID\_PWR Power Budgeting

# pci\_find\_next\_ht\_capability

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_find_next_ht_capability` — query a device's Hypertransport capabilities

### Synopsis

```
int pci_find_next_ht_capability (struct pci_dev * dev, int  
pos, int ht_cap);
```

### Arguments

*dev*

PCI device to query

*pos*

Position from which to continue searching

*ht\_cap*

Hypertransport capability code

### Description

To be used in conjunction with `pci_find_ht_capability` to search for all capabilities matching *ht\_cap*. *pos* should always be a value returned from `pci_find_ht_capability`.

NB. To be 100% safe against broken PCI devices, the caller should take steps to avoid an infinite loop.

# pci\_find\_ht\_capability

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_find_ht_capability` — query a device’s Hypertransport capabilities

### Synopsis

```
int pci_find_ht_capability (struct pci_dev * dev, int ht_cap);
```

### Arguments

*dev*

PCI device to query

*ht\_cap*

Hypertransport capability code

### Description

Tell if a device supports a given Hypertransport capability. Returns an address within the device’s PCI configuration space or 0 in case the device does not support the request capability. The address points to the PCI capability, of type `PCI_CAP_ID_HT`, which has a Hypertransport capability matching *ht\_cap*.

# pci\_find\_parent\_resource

## LINUX

Kernel Hackers Manual June 2013

### 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\_complete\_power\_transition

## LINUX

## Name

`__pci_complete_power_transition` — Complete power transition of a PCI device

## Synopsis

```
int __pci_complete_power_transition (struct pci_dev * dev,  
pci_power_t state);
```

## Arguments

*dev*

PCI device to handle.

*state*

State to put the device into.

## Description

This function should not be called directly by device drivers.

# pci\_set\_power\_state

**LINUX**



## 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 handle.

*state*

PCI power state (D0, D1, D2, D3hot) to put the device into.

## Description

Transition a device to a new power state, using the platform firmware and/or the device's PCI PM registers.

## RETURN VALUE

-EINVAL if the requested state is invalid. -EIO if device does not support PCI PM or its PM capabilities register has a wrong version, or device doesn't support the requested state. 0 if device already is in the requested state. 0 if device's power state has been successfully changed.

# pci\_choose\_state

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_choose_state` — Choose the power state of a PCI device

### Synopsis

```
pci_power_t pci_choose_state (struct pci_dev * dev,  
pm_message_t state);
```

### Arguments

*dev*

PCI device to be suspended

*state*

target sleep state for the whole system. This is the value that is passed to suspend function.

### Description

Returns PCI power state suitable for given device and given system message.

# pci\_save\_state

## LINUX

## 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

# pci\_restore\_state

## LINUX

## Name

`pci_restore_state` — Restore the saved state of a PCI device

## Synopsis

```
void pci_restore_state (struct pci_dev * dev);
```

## Arguments

*dev*

- PCI device that we're dealing with

## pci\_store\_saved\_state

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_store_saved_state` — Allocate and return an opaque struct containing the device saved state.

## Synopsis

```
struct pci_saved_state * pci_store_saved_state (struct pci_dev  
* dev);
```

## Arguments

*dev*

PCI device that we're dealing with

## Description

Return NULL if no state or error.

# pci\_load\_saved\_state

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_load_saved_state` — Reload the provided save state into struct `pci_dev`.

### Synopsis

```
int pci_load_saved_state (struct pci_dev * dev, struct
pci_saved_state * state);
```

### Arguments

*dev*

PCI device that we're dealing with

*state*

Saved state returned from `pci_store_saved_state`

# pci\_load\_and\_free\_saved\_state

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_load_and_free_saved_state` — Reload the save state pointed to by `state`, and free the memory allocated for it.

## Synopsis

```
int pci_load_and_free_saved_state (struct pci_dev * dev,  
struct pci_saved_state ** state);
```

## Arguments

*dev*

PCI device that we're dealing with

*state*

Pointer to saved state returned from `pci_store_saved_state`

## pci\_reenable\_device

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_reenable_device` — Resume abandoned device

## Synopsis

```
int pci_reenable_device (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device to be resumed

## Description

Note this function is a backend of `pci_default_resume` and is not supposed to be called by normal code, write proper resume handler and use it instead.

# pci\_enable\_device\_io

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_device_io` — Initialize a device for use with IO space

## Synopsis

```
int pci_enable_device_io (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 resources. Wake up the device if it was suspended. Beware, this function can fail.

# pci\_enable\_device\_mem

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_device_mem` — Initialize a device for use with Memory space

## Synopsis

```
int pci_enable_device_mem (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 Memory resources. Wake up the device if it was suspended. Beware, this function can fail.



# pci\_enable\_device

## LINUX

Kernel Hackers Manual June 2013

### 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.

Note we don't actually enable the device many times if we call this function repeatedly (we just increment the count).

# pcim\_enable\_device

## LINUX

## Name

`pcim_enable_device` — Managed `pci_enable_device`

## Synopsis

```
int pcim_enable_device (struct pci_dev * pdev);
```

## Arguments

*pdev*

PCI device to be initialized

## Description

Managed `pci_enable_device`.

# pcim\_pin\_device

## LINUX

## Name

`pcim_pin_device` — Pin managed PCI device

## Synopsis

```
void pcim_pin_device (struct pci_dev * pdev);
```

## Arguments

*pdev*

PCI device to pin

## Description

Pin managed PCI device *pdev*. Pinned device won't be disabled on driver detach. *pdev* must have been enabled with `pcim_enable_device`.

# pci\_disable\_device

## LINUX

Kernel Hackers Manual June 2013

## 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.

Note we don't actually disable the device until all callers of `pci_enable_device` have called `pci_disable_device`.

# pci\_set\_pcie\_reset\_state

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_set_pcie_reset_state` — set reset state for device `dev`

## Synopsis

```
int pci_set_pcie_reset_state (struct pci_dev * dev, enum  
pcie_reset_state state);
```

## Arguments

*dev*

the PCIe device reset

*state*

Reset state to enter into

## Description

Sets the PCI reset state for the device.

# pci\_pme\_capable

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_pme_capable` — check the capability of PCI device to generate PME#

## Synopsis

```
bool pci_pme_capable (struct pci_dev * dev, pci_power_t
state);
```

## Arguments

*dev*

PCI device to handle.

*state*

PCI state from which device will issue PME#.

## pci\_pme\_active

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_pme_active` — enable or disable PCI device's PME# function

### Synopsis

```
void pci_pme_active (struct pci_dev * dev, bool enable);
```

### Arguments

*dev*

PCI device to handle.

*enable*

'true' to enable PME# generation; 'false' to disable it.

### Description

The caller must verify that the device is capable of generating PME# before calling this function with *enable* equal to 'true'.

## \_\_pci\_enable\_wake

### LINUX

## Name

`__pci_enable_wake` — enable PCI device as wakeup event source

## Synopsis

```
int __pci_enable_wake (struct pci_dev * dev, pci_power_t
state, bool runtime, bool enable);
```

## Arguments

*dev*

PCI device affected

*state*

PCI state from which device will issue wakeup events

*runtime*

True if the events are to be generated at run time

*enable*

True to enable event generation; false to disable

## Description

This enables the device as a wakeup event source, or disables it. When such events involves platform-specific hooks, those hooks are called automatically by this routine.

Devices with legacy power management (no standard PCI PM capabilities) always require such platform hooks.

## RETURN VALUE

0 is returned on success -EINVAL is returned if device is not supposed to wake up the system Error code depending on the platform is returned if both the platform and the native mechanism fail to enable the generation of wake-up events

## pci\_wake\_from\_d3

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_wake_from_d3` — enable/disable device to wake up from D3\_hot or D3\_cold

### Synopsis

```
int pci_wake_from_d3 (struct pci_dev * dev, bool enable);
```

### Arguments

*dev*

PCI device to prepare

*enable*

True to enable wake-up event generation; false to disable

### Description

Many drivers want the device to wake up the system from D3\_hot or D3\_cold and this function allows them to set that up cleanly - `pci_enable_wake` should not be



called twice in a row to enable wake-up due to PCI PM vs ACPI ordering constraints.

This function only returns error code if the device is not capable of generating PME# from both D3\_hot and D3\_cold, and the platform is unable to enable wake-up power for it.

## pci\_target\_state

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_target_state` — find an appropriate low power state for a given PCI dev

### Synopsis

```
pci_power_t pci_target_state (struct pci_dev * dev);
```

### Arguments

*dev*

PCI device

### Description

Use underlying platform code to find a supported low power state for *dev*. If the platform can't manage *dev*, return the deepest state from which it can generate wake events, based on any available PME info.

# pci\_prepare\_to\_sleep

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_prepare_to_sleep` — prepare PCI device for system-wide transition into a sleep state

### Synopsis

```
int pci_prepare_to_sleep (struct pci_dev * dev);
```

### Arguments

*dev*

Device to handle.

### Description

Choose the power state appropriate for the device depending on whether it can wake up the system and/or is power manageable by the platform (`PCI_D3hot` is the default) and put the device into that state.

# pci\_back\_from\_sleep

## LINUX

## Name

`pci_back_from_sleep` — turn PCI device on during system-wide transition into working state

## Synopsis

```
int pci_back_from_sleep (struct pci_dev * dev);
```

## Arguments

*dev*

Device to handle.

## Description

Disable device's system wake-up capability and put it into D0.

# pci\_dev\_run\_wake

## LINUX

## Name

`pci_dev_run_wake` — Check if device can generate run-time wake-up events.

## Synopsis

```
bool pci_dev_run_wake (struct pci_dev * dev);
```

## Arguments

*dev*

Device to check.

## Description

Return true if the device itself is capable of generating wake-up events (through the platform or using the native PCIe PME) or if the device supports PME and one of its upstream bridges can generate wake-up events.

## pci\_enable\_ido

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_ido` — enable ID-based ordering on a device

## Synopsis

```
void pci_enable_ido (struct pci_dev * dev, unsigned long  
type);
```

## Arguments

*dev*

the PCI device

*type*

which types of IDO to enable

## Description

Enable ID-based ordering on *dev*. *type* can contain the bits `PCI_EXP_IDO_REQUEST` and/or `PCI_EXP_IDO_COMPLETION` to indicate which types of transactions are allowed to be re-ordered.

# pci\_disable\_ido

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_disable_ido` — disable ID-based ordering on a device

## Synopsis

```
void pci_disable_ido (struct pci_dev * dev, unsigned long
type);
```

## Arguments

*dev*

the PCI device

*type*

which types of IDO to disable

## pci\_enable\_obff

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_obff` — enable optimized buffer flush/fill

## Synopsis

```
int pci_enable_obff (struct pci_dev * dev, enum  
pci_obff_signal_type type);
```

## Arguments

*dev*

PCI device

*type*

type of signaling to use

## Description

Try to enable *type* OBFF signaling on *dev*. It will try using WAKE# signaling if possible, falling back to message signaling only if WAKE# isn't supported. *type* should indicate whether the PCIe link be brought out of L0s or L1 to send the message. It should be either `PCI_EXP_OBFF_SIGNAL_ALWAYS` or `PCI_OBFF_SIGNAL_L0`.

If your device can benefit from receiving all messages, even at the power cost of bringing the link back up from a low power state, use `PCI_EXP_OBFF_SIGNAL_ALWAYS`. Otherwise, use `PCI_OBFF_SIGNAL_L0` (the preferred type).

## RETURNS

Zero on success, appropriate error number on failure.

# pci\_disable\_obff

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_disable_obff` — disable optimized buffer flush/fill

## Synopsis

```
void pci_disable_obff (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device

## Description

Disable OBFF on *dev*.

# pci\_ltr\_supported

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_ltr_supported` — check whether a device supports LTR

## Synopsis

```
bool pci_ltr_supported (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device

## RETURNS

True if *dev* supports latency tolerance reporting, false otherwise.



# pci\_enable\_ltr

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_enable_ltr` — enable latency tolerance reporting

### Synopsis

```
int pci_enable_ltr (struct pci_dev * dev);
```

### Arguments

*dev*

PCI device

### Description

Enable LTR on *dev* if possible, which means enabling it first on upstream ports.

### RETURNS

Zero on success, `errno` on failure.

## pci\_disable\_ltr

### LINUX

Kernel Hackers Manual June 2013

#### Name

`pci_disable_ltr` — disable latency tolerance reporting

#### Synopsis

```
void pci_disable_ltr (struct pci_dev * dev);
```

#### Arguments

*dev*

PCI device

## pci\_set\_ltr

### LINUX

Kernel Hackers Manual June 2013

#### Name

`pci_set_ltr` — set LTR latency values

## Synopsis

```
int pci_set_ltr (struct pci_dev * dev, int snoop_lat_ns, int
nosnoop_lat_ns);
```

## Arguments

*dev*

PCI device

*snoop\_lat\_ns*

snoop latency in nanoseconds

*nosnoop\_lat\_ns*

nosnoop latency in nanoseconds

## Description

Figure out the scale and set the LTR values accordingly.

# pci\_release\_region

## LINUX

Kernel Hackers Manual June 2013

## 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_request_region` — Reserve PCI I/O and memory resource

## Synopsis

```
int pci_request_region (struct pci_dev * pdev, int bar, const  
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* BAR *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\_request\_region\_exclusive

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_request_region_exclusive` — Reserved PCI I/O and memory resource

## Synopsis

```
int pci_request_region_exclusive (struct pci_dev * pdev, int  
bar, const 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.

The key difference that `_exclusive` makes it that userspace is explicitly not allowed to map the resource via `/dev/mem` or `sysfs`.

## pci\_release\_selected\_regions

**LINUX**

## Name

`pci_release_selected_regions` — Release selected PCI I/O and memory resources

## Synopsis

```
void pci_release_selected_regions (struct pci_dev * pdev, int  
bars);
```

## Arguments

*pdev*

PCI device whose resources were previously reserved

*bars*

Bitmask of BARs to be released

## Description

Release selected PCI I/O and memory resources previously reserved. Call this function only after all use of the PCI regions has ceased.

## `pci_request_selected_regions`

**LINUX**

## Name

`pci_request_selected_regions` — Reserve selected PCI I/O and memory resources

## Synopsis

```
int pci_request_selected_regions (struct pci_dev * pdev, int
bars, const char * res_name);
```

## Arguments

*pdev*

PCI device whose resources are to be reserved

*bars*

Bitmask of BARs to be requested

*res\_name*

Name to be associated with resource

# pci\_release\_regions

## LINUX

## 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_request_regions` — Reserved PCI I/O and memory resources

## Synopsis

```
int pci_request_regions (struct pci_dev * pdev, const 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\_request\_regions\_exclusive

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_request_regions_exclusive` — Reserved PCI I/O and memory resources

## Synopsis

```
int pci_request_regions_exclusive (struct pci_dev * pdev,  
const 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.

`pci_request_regions_exclusive` will mark the region so that `/dev/mem` and the sysfs MMIO access will not be allowed.

Returns 0 on success, or `EBUSY` on error. A warning message is also printed on failure.

## pci\_set\_master

### LINUX

Kernel Hackers Manual June 2013

### 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\_clear\_master

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_clear_master` — disables bus-mastering for device *dev*

## Synopsis

```
void pci_clear_master (struct pci_dev * dev);
```

## Arguments

*dev*

the PCI device to disable

# pci\_set\_cacheline\_size

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_set_cacheline_size` — ensure the `CACHE_LINE_SIZE` register is programmed

### Synopsis

```
int pci_set_cacheline_size (struct pci_dev * dev);
```

### Arguments

*dev*

the PCI device for which MWI is to be enabled

### Description

Helper function for `pci_set_mwi`. Originally copied from `drivers/net/acenic.c`.  
Copyright 1998-2001 by Jes Sorensen, <[jesttrained-monkey.org](mailto:jesttrained-monkey.org)>.

### RETURNS

An appropriate `-ERRNO` error value on error, or zero for success.

## pci\_set\_mwi

### LINUX

Kernel Hackers Manual June 2013

### 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`.

### RETURNS

An appropriate `-ERRNO` error value on error, or zero for success.

## pci\_try\_set\_mwi

### LINUX

## Name

`pci_try_set_mwi` — enables memory-write-invalidate PCI transaction

## Synopsis

```
int pci_try_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`. Callers are not required to check the return value.

## RETURNS

An appropriate `-ERRNO` error value on error, or zero for success.

## `pci_clear_mwi`

**LINUX**

## 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

# pci\_intx

## LINUX

## Name

`pci_intx` — enables/disables PCI INTx for device `dev`



## Synopsis

```
void pci_intx (struct pci_dev * pdev, int enable);
```

## Arguments

*pdev*

the PCI device to operate on

*enable*

boolean: whether to enable or disable PCI INTx

## Description

Enables/disables PCI INTx for device *dev*

## pci\_msi\_off

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_msi_off` — disables any msi or msix capabilities

## Synopsis

```
void pci_msi_off (struct pci_dev * dev);
```

## Arguments

*dev*

the PCI device to operate on

## Description

If you want to use msi see `pci_enable_msi` and friends. This is a lower level primitive that allows us to disable msi operation at the device level.

# \_\_pci\_reset\_function

## LINUX

Kernel Hackers Manual June 2013

## Name

`__pci_reset_function` — reset a PCI device function

## Synopsis

```
int __pci_reset_function (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device to reset

## Description

Some devices allow an individual function to be reset without affecting other functions in the same device. The PCI device must be responsive to PCI config space in order to use this function.

The device function is presumed to be unused when this function is called. Resetting the device will make the contents of PCI configuration space random, so any caller of this must be prepared to reinitialise the device including MSI, bus mastering, BARs, decoding IO and memory spaces, etc.

Returns 0 if the device function was successfully reset or negative if the device doesn't support resetting a single function.

## pci\_reset\_function

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_reset_function` — quiesce and reset a PCI device function

### Synopsis

```
int pci_reset_function (struct pci_dev * dev);
```

### Arguments

*dev*

PCI device to reset

## Description

Some devices allow an individual function to be reset without affecting other functions in the same device. The PCI device must be responsive to PCI config space in order to use this function.

This function does not just reset the PCI portion of a device, but clears all the state associated with the device. This function differs from `__pci_reset_function` in that it saves and restores device state over the reset.

Returns 0 if the device function was successfully reset or negative if the device doesn't support resetting a single function.

## pcix\_get\_max\_mmrbc

### LINUX

Kernel Hackers Manual June 2013

### Name

`pcix_get_max_mmrbc` — get PCI-X maximum designed memory read byte count

### Synopsis

```
int pcix_get_max_mmrbc (struct pci_dev * dev);
```

### Arguments

*dev*

PCI device to query

## Returns mmrbc

maximum designed memory read count in bytes or appropriate error value.

# pcix\_get\_mmrbc

## LINUX

Kernel Hackers Manual June 2013

## Name

`pcix_get_mmrbc` — get PCI-X maximum memory read byte count

## Synopsis

```
int pcix_get_mmrbc (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device to query

## Returns mmrbc

maximum memory read count in bytes or appropriate error value.

# pcix\_set\_mmrbc

## LINUX

Kernel Hackers Manual June 2013

### Name

`pcix_set_mmrbc` — set PCI-X maximum memory read byte count

### Synopsis

```
int pcix_set_mmrbc (struct pci_dev * dev, int mmrbc);
```

### Arguments

*dev*

PCI device to query

*mmrbc*

maximum memory read count in bytes valid values are 512, 1024, 2048, 4096

### Description

If possible sets maximum memory read byte count, some bridges have erratas that prevent this.

# pcie\_get\_readrq

## LINUX

## Name

`pcie_get_readrq` — get PCI Express read request size

## Synopsis

```
int pcie_get_readrq (struct pci_dev * dev);
```

## Arguments

*dev*

PCI device to query

## Description

Returns maximum memory read request in bytes or appropriate error value.

# pcie\_set\_readrq

## LINUX

## Name

`pcie_set_readrq` — set PCI Express maximum memory read request

## Synopsis

```
int pcie_set_readrq (struct pci_dev * dev, int rq);
```

## Arguments

*dev*

PCI device to query

*rq*

maximum memory read count in bytes valid values are 128, 256, 512, 1024, 2048, 4096

## Description

If possible sets maximum read byte count

## pci\_selectBars

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_selectBars` — Make BAR mask from the type of resource

## Synopsis

```
int pci_selectBars (struct pci_dev * dev, unsigned long  
flags);
```



## Arguments

*dev*

the PCI device for which BAR mask is made

*flags*

resource type mask to be selected

## Description

This helper routine makes bar mask from the type of resource.

## pci\_add\_dynid

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_add_dynid` — add a new PCI device ID to this driver and re-probe devices

## Synopsis

```
int pci_add_dynid (struct pci_driver * drv, unsigned int  
vendor, unsigned int device, unsigned int subvendor, unsigned  
int subdevice, unsigned int class, unsigned int class_mask,  
unsigned long driver_data);
```

## Arguments

*drv*

target pci driver

*vendor*

PCI vendor ID

*device*

PCI device ID

*subvendor*

PCI subvendor ID

*subdevice*

PCI subdevice ID

*class*

PCI class

*class\_mask*

PCI class mask

*driver\_data*

private driver data

## Description

Adds a new dynamic pci device ID to this driver and causes the driver to probe for all devices again. *drv* must have been registered prior to calling this function.

## CONTEXT

Does GFP\_KERNEL allocation.

## RETURNS

0 on success, -errno on failure.

# pci\_match\_id

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_match_id` — See if a pci device matches a given `pci_id` table

## Synopsis

```
const struct pci_device_id * pci_match_id (const struct
pci_device_id * ids, struct pci_dev * dev);
```

## Arguments

*ids*

array of PCI device id structures to search in

*dev*

the PCI device structure to match against.

## Description

Used by a driver to check whether a PCI device present in the system is in its list of supported devices. Returns the matching `pci_device_id` structure or `NULL` if there is no match.

Deprecated, don't use this as it will not catch any dynamic ids that a driver might want to check for.

# \_\_pci\_register\_driver

## LINUX

Kernel Hackers Manual June 2013

### Name

`__pci_register_driver` — register a new pci driver

### Synopsis

```
int __pci_register_driver (struct pci_driver * drv, struct
module * owner, const char * mod_name);
```

### Arguments

*drv*

the driver structure to register

*owner*

owner module of drv

*mod\_name*

module name string

### Description

Adds the driver structure to the list of registered drivers. Returns a negative value on error, otherwise 0. If no error occurred, the driver remains registered even if no device was claimed during registration.

# pci\_unregister\_driver

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_unregister_driver` — unregister a pci driver

### Synopsis

```
void pci_unregister_driver (struct pci_driver * drv);
```

### Arguments

*drv*

the driver structure to unregister

### Description

Deletes the driver structure from the list of registered PCI drivers, gives it a chance to clean up by calling its `remove` function for each device it was responsible for, and marks those devices as driverless.

# pci\_dev\_driver

## LINUX

## Name

`pci_dev_driver` — get the `pci_driver` of a device

## Synopsis

```
struct pci_driver * pci_dev_driver (const struct pci_dev *  
dev);
```

## Arguments

*dev*

the device to query

## Description

Returns the appropriate `pci_driver` structure or `NULL` if there is no registered driver for the device.

# pci\_dev\_get

## LINUX

## Name

`pci_dev_get` — increments the reference count of the pci device structure

## Synopsis

```
struct pci_dev * pci_dev_get (struct pci_dev * dev);
```

## Arguments

*dev*

the device being referenced

## Description

Each live reference to a device should be refcounted.

Drivers for PCI devices should normally record such references in their `probe` methods, when they bind to a device, and release them by calling `pci_dev_put`, in their `disconnect` methods.

A pointer to the device with the incremented reference counter is returned.

## pci\_dev\_put

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_dev_put` — release a use of the pci device structure

## Synopsis

```
void pci_dev_put (struct pci_dev * dev);
```

## Arguments

*dev*

device that's been disconnected

## Description

Must be called when a user of a device is finished with it. When the last user of the device calls this function, the memory of the device is freed.

# pci\_remove\_bus\_device

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_remove_bus_device` — remove a PCI device and any children

## Synopsis

```
void pci_remove_bus_device (struct pci_dev * dev);
```

## Arguments

*dev*

the device to remove



## Description

Remove a PCI device from the device lists, informing the drivers that the device has been removed. We also remove any subordinate buses and children in a depth-first manner.

For each device we remove, delete the device structure from the device lists, remove the /proc entry, and notify userspace (/sbin/hotplug).

# pci\_remove\_behind\_bridge

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_remove_behind_bridge` — remove all devices behind a PCI bridge

## Synopsis

```
void pci_remove_behind_bridge (struct pci_dev * dev);
```

## Arguments

*dev*

PCI bridge device

## Description

Remove all devices on the bus, except for the parent bridge. This also removes any child buses, and any devices they may contain in a depth-first manner.

## pci\_stop\_bus\_device

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_stop_bus_device` — stop a PCI device and any children

### Synopsis

```
void pci_stop_bus_device (struct pci_dev * dev);
```

### Arguments

*dev*

the device to stop

### Description

Stop a PCI device (detach the driver, remove from the global list and so on). This also stop any subordinate buses and children in a depth-first manner.

## pci\_find\_bus

### LINUX

## Name

`pci_find_bus` — locate PCI bus from a given domain and bus number

## Synopsis

```
struct pci_bus * pci_find_bus (int domain, int busnr);
```

## Arguments

*domain*

number of PCI domain to search

*busnr*

number of desired PCI bus

## Description

Given a PCI bus number and domain number, the desired PCI bus is located in the global list of PCI buses. If the bus is found, a pointer to its data structure is returned. If no bus is found, `NULL` is returned.

## `pci_find_next_bus`

**LINUX**

## Name

`pci_find_next_bus` — begin or continue searching for a PCI bus

## Synopsis

```
struct pci_bus * pci_find_next_bus (const struct pci_bus *  
from);
```

## Arguments

*from*

Previous PCI bus found, or `NULL` for new search.

## Description

Iterates through the list of known PCI busses. A new search is initiated by passing `NULL` as the *from* argument. Otherwise if *from* is not `NULL`, searches continue from next device on the global list.

# pci\_get\_slot

## LINUX

## Name

`pci_get_slot` — locate PCI device for a given PCI slot

## Synopsis

```
struct pci_dev * pci_get_slot (struct pci_bus * bus, unsigned
int devfn);
```

## Arguments

*bus*

PCI bus on which desired PCI device resides

*devfn*

encodes number of PCI slot in which the desired PCI device resides and the logical device number within that slot in case of multi-function devices.

## Description

Given a PCI bus and slot/function number, the desired PCI device is located in the list of PCI devices. If the device is found, its reference count is increased and this function returns a pointer to its data structure. The caller must decrement the reference count by calling `pci_dev_put`. If no device is found, `NULL` is returned.

## pci\_get\_domain\_bus\_and\_slot

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_get_domain_bus_and_slot` — locate PCI device for a given PCI domain (segment), bus, and slot

## Synopsis

```
struct pci_dev * pci_get_domain_bus_and_slot (int domain,  
unsigned int bus, unsigned int devfn);
```

## Arguments

*domain*

PCI domain/segment on which the PCI device resides.

*bus*

PCI bus on which desired PCI device resides

*devfn*

encodes number of PCI slot in which the desired PCI device resides and the logical device number within that slot in case of multi-function devices.

## Description

Given a PCI domain, bus, and slot/function number, the desired PCI device is located in the list of PCI devices. If the device is found, its reference count is increased and this function returns a pointer to its data structure. The caller must decrement the reference count by calling `pci_dev_put`. If no device is found, `NULL` is returned.

## pci\_get\_subsys

**LINUX**

## Name

`pci_get_subsys` — begin or continue searching for a PCI device by  
vendor/subvendor/device/subdevice id

## Synopsis

```
struct pci_dev * pci_get_subsys (unsigned int vendor, unsigned  
int device, unsigned int ss_vendor, unsigned int ss_device,  
struct pci_dev * from);
```

## Arguments

*vendor*

PCI vendor id to match, or `PCI_ANY_ID` to match all vendor ids

*device*

PCI device id to match, or `PCI_ANY_ID` to match all device ids

*ss\_vendor*

PCI subsystem vendor id to match, or `PCI_ANY_ID` to match all vendor ids

*ss\_device*

PCI subsystem device id to match, or `PCI_ANY_ID` to match all device ids

*from*

Previous PCI device found in search, or `NULL` for new search.

## Description

Iterates through the list of known PCI devices. If a PCI device is found with a matching *vendor*, *device*, *ss\_vendor* and *ss\_device*, a pointer to its device structure is returned, and the reference count to the device is incremented.

Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` as the *from*

argument. Otherwise if *from* is not `NULL`, searches continue from next device on the global list. The reference count for *from* is always decremented if it is not `NULL`.

## pci\_get\_device

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_get_device` — begin or continue searching for a PCI device by vendor/device id

### Synopsis

```
struct pci_dev * pci_get_device (unsigned int vendor, unsigned  
int device, struct pci_dev * from);
```

### Arguments

*vendor*

PCI vendor id to match, or `PCI_ANY_ID` to match all vendor ids

*device*

PCI device id to match, or `PCI_ANY_ID` to match all device ids

*from*

Previous PCI device found in search, or `NULL` for new search.



## Description

Iterates through the list of known PCI devices. If a PCI device is found with a matching *vendor* and *device*, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` as the *from* argument. Otherwise if *from* is not `NULL`, searches continue from next device on the global list. The reference count for *from* is always decremented if it is not `NULL`.

## pci\_get\_class

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_get_class` — begin or continue searching for a PCI device by class

### Synopsis

```
struct pci_dev * pci_get_class (unsigned int class, struct
pci_dev * from);
```

### Arguments

*class*

search for a PCI device with this class designation

*from*

Previous PCI device found in search, or `NULL` for new search.

## Description

Iterates through the list of known PCI devices. If a PCI device is found with a matching *class*, the reference count to the device is incremented and a pointer to its device structure is returned. Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` as the *from* argument. Otherwise if *from* is not `NULL`, searches continue from next device on the global list. The reference count for *from* is always decremented if it is not `NULL`.

## pci\_dev\_present

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_dev_present` — Returns 1 if device matching the device list is present, 0 if not.

### Synopsis

```
int pci_dev_present (const struct pci_device_id * ids);
```

### Arguments

*ids*

A pointer to a null terminated list of struct `pci_device_id` structures that describe the type of PCI device the caller is trying to find.

## Obvious fact

You do not have a reference to any device that might be found by this function, so if that device is removed from the system right after this function is finished, the value will be stale. Use this function to find devices that are usually built into a system, or for a general hint as to if another device happens to be present at this specific moment in time.

# pci\_enable\_msi\_block

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_msi_block` — configure device's MSI capability structure

## Synopsis

```
int pci_enable_msi_block (struct pci_dev * dev, unsigned int
nvec);
```

## Arguments

*dev*

device to configure

*nvec*

number of interrupts to configure

## Description

Allocate IRQs for a device with the MSI capability. This function returns a negative `errno` if an error occurs. If it is unable to allocate the number of interrupts requested, it returns the number of interrupts it might be able to allocate. If it successfully allocates at least the number of interrupts requested, it returns 0 and updates the `dev`'s `irq` member to the lowest new interrupt number; the other interrupt numbers allocated to this device are consecutive.

## pci\_enable\_msix

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_enable_msix` — configure device's MSI-X capability structure

### Synopsis

```
int pci_enable_msix (struct pci_dev * dev, struct msix_entry *
entries, int nvec);
```

### Arguments

*dev*

pointer to the `pci_dev` data structure of MSI-X device function

*entries*

pointer to an array of MSI-X entries

*nvec*

number of MSI-X irqs requested for allocation by device driver

## Description

Setup the MSI-X capability structure of device function with the number of requested irqs upon its software driver call to request for MSI-X mode enabled on its hardware device function. A return of zero indicates the successful configuration of MSI-X capability structure with new allocated MSI-X irqs. A return of < 0 indicates a failure. Or a return of > 0 indicates that driver request is exceeding the number of irqs or MSI-X vectors available. Driver should use the returned value to re-send its request.

## pci\_msi\_enabled

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_msi_enabled` — is MSI enabled?

## Synopsis

```
int pci_msi_enabled ( void );
```

## Arguments

*void*

no arguments

## Description

Returns true if MSI has not been disabled by the command-line option `pci=noms`.

# pci\_bus\_alloc\_resource

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_bus_alloc_resource` — allocate a resource from a parent bus

### Synopsis

```
int pci_bus_alloc_resource (struct pci_bus * bus, struct
resource * res, resource_size_t size, resource_size_t align,
resource_size_t min, unsigned int type_mask, resource_size_t
(*alignf) (void *, const struct resource *, resource_size_t,
resource_size_t), void * alignf_data);
```

### Arguments

*bus*

PCI bus

*res*

resource to allocate

*size*

size of resource to allocate

*align*

alignment of resource to allocate

*min*

minimum /proc/iomem address to allocate

*type\_mask*

IORESOURCE\_\* type flags

*alignf*

resource alignment function

*alignf\_data*

data argument for resource alignment function

## Description

Given the PCI bus a device resides on, the size, minimum address, alignment and type, try to find an acceptable resource allocation for a specific device resource.

# pci\_bus\_add\_device

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_bus_add_device` — add a single device

## Synopsis

```
int pci_bus_add_device (struct pci_dev * dev);
```

## Arguments

*dev*

device to add

## Description

This adds a single pci device to the global device list and adds sysfs and procfs entries

# pci\_bus\_add\_devices

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_bus_add_devices` — insert newly discovered PCI devices

## Synopsis

```
void pci_bus_add_devices (const struct pci_bus * bus);
```

## Arguments

*bus*

bus to check for new devices

## Description

Add newly discovered PCI devices (which are on the `bus->devices` list) to the global PCI device list, add the sysfs and procfs entries. Where a bridge is found, add the discovered bus to the parents list of child buses, and recurse (breadth-first to be compatible with 2.4)

Call hotplug for each new devices.



# pci\_bus\_set\_ops

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_bus_set_ops` — Set raw operations of pci bus

### Synopsis

```
struct pci_ops * pci_bus_set_ops (struct pci_bus * bus, struct  
pci_ops * ops);
```

### Arguments

*bus*

pci bus struct

*ops*

new raw operations

### Description

Return previous raw operations

# pci\_read\_vpd

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_read_vpd` — Read one entry from Vital Product Data

### Synopsis

```
ssize_t pci_read_vpd (struct pci_dev * dev, loff_t pos, size_t
count, void * buf);
```

### Arguments

*dev*

pci device struct

*pos*

offset in vpd space

*count*

number of bytes to read

*buf*

pointer to where to store result

# pci\_write\_vpd

## LINUX

## Name

`pci_write_vpd` — Write entry to Vital Product Data

## Synopsis

```
ssize_t pci_write_vpd (struct pci_dev * dev, loff_t pos,  
size_t count, const void * buf);
```

## Arguments

*dev*

pci device struct

*pos*

offset in vpd space

*count*

number of bytes to write

*buf*

buffer containing write data

## `pci_vpd_truncate`

**LINUX**

## Name

`pci_vpd_truncate` — Set available Vital Product Data size

## Synopsis

```
int pci_vpd_truncate (struct pci_dev * dev, size_t size);
```

## Arguments

*dev*

pci device struct

*size*

available memory in bytes

## Description

Adjust size of available VPD area.

# pci\_block\_user\_cfg\_access

## LINUX

## Name

`pci_block_user_cfg_access` — Block userspace PCI config reads/writes

## Synopsis

```
void pci_block_user_cfg_access (struct pci_dev * dev);
```

## Arguments

*dev*

pci device struct

## Description

When user access is blocked, any reads or writes to config space will sleep until access is unblocked again. We don't allow nesting of block/unblock calls.

# pci\_unblock\_user\_cfg\_access

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_unblock_user_cfg_access` — Unblock userspace PCI config reads/writes

## Synopsis

```
void pci_unblock_user_cfg_access (struct pci_dev * dev);
```

## Arguments

*dev*

pci device struct

## Description

This function allows userspace PCI config accesses to resume.

# pci\_lost\_interrupt

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_lost_interrupt` — reports a lost PCI interrupt

## Synopsis

```
enum pci_lost_interrupt_reason pci_lost_interrupt (struct  
pci_dev * pdev);
```

## Arguments

*pdev*

device whose interrupt is lost

## Description

The primary function of this routine is to report a lost interrupt in a standard way which users can recognise (instead of blaming the driver).

## Returns

a suggestion for fixing it (although the driver is not required to act on this).

# \_\_ht\_create\_irq

## LINUX

Kernel Hackers Manual June 2013

## Name

`__ht_create_irq` — create an irq and attach it to a device.

## Synopsis

```
int __ht_create_irq (struct pci_dev * dev, int idx,  
ht_irq_update_t * update);
```

## Arguments

*dev*

The hypertransport device to find the irq capability on.

*idx*

Which of the possible irqs to attach to.

*update*

Function to be called when changing the htirq message

## Description

The irq number of the new irq or a negative error value is returned.

# ht\_create\_irq

## LINUX

Kernel Hackers Manual June 2013

## Name

`ht_create_irq` — create an irq and attach it to a device.

## Synopsis

```
int ht_create_irq (struct pci_dev * dev, int idx);
```

## Arguments

*dev*

The hypertransport device to find the irq capability on.

*idx*

Which of the possible irqs to attach to.



## Description

`ht_create_irq` needs to be called for all hypertransport devices that generate irqs.

The irq number of the new irq or a negative error value is returned.

## ht\_destroy\_irq

### LINUX

Kernel Hackers Manual June 2013

## Name

`ht_destroy_irq` — destroy an irq created with `ht_create_irq`

## Synopsis

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

## Arguments

*irq*

irq to be destroyed

## Description

This reverses `ht_create_irq` removing the specified irq from existence. The irq should be free before this happens.

# pci\_scan\_slot

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_scan_slot` — scan a PCI slot on a bus for devices.

### Synopsis

```
int pci_scan_slot (struct pci_bus * bus, int devfn);
```

### Arguments

*bus*

PCI bus to scan

*devfn*

slot number to scan (must have zero function.)

### Description

Scan a PCI slot on the specified PCI bus for devices, adding discovered devices to the *bus*->devices list. New devices will not have `is_added` set.

Returns the number of new devices found.

# pci\_rescan\_bus

## LINUX

## Name

`pci_rescan_bus` — scan a PCI bus for devices.

## Synopsis

```
unsigned int __ref pci_rescan_bus (struct pci_bus * bus);
```

## Arguments

*bus*

PCI bus to scan

## Description

Scan a PCI bus and child buses for new devices, adds them, and enables them.

Returns the max number of subordinate bus discovered.

# pci\_create\_slot

## LINUX

## Name

`pci_create_slot` — create or increment refcount for physical PCI slot

## Synopsis

```
struct pci_slot * pci_create_slot (struct pci_bus * parent,  
int slot_nr, const char * name, struct hotplug_slot *  
hotplug);
```

## Arguments

*parent*

struct pci\_bus of parent bridge

*slot\_nr*

PCI\_SLOT(pci\_dev->devfn) or -1 for placeholder

*name*

user visible string presented in /sys/bus/pci/slots/<name>

*hotplug*

set if caller is hotplug driver, NULL otherwise

## Description

PCI slots have first class attributes such as address, speed, width, and a struct pci\_slot is used to manage them. This interface will either return a new struct pci\_slot to the caller, or if the pci\_slot already exists, its refcount will be incremented.

Slots are uniquely identified by a *pci\_bus*, *slot\_nr* tuple.

There are known platforms with broken firmware that assign the same name to multiple slots. Workaround these broken platforms by renaming the slots on behalf of the caller. If firmware assigns name N to

## multiple slots

The first slot is assigned N The second slot is assigned N-1 The third slot is assigned N-2 etc.

## Placeholder slots

In most cases, `pci_bus`, `slot_nr` will be sufficient to uniquely identify a slot. There is one notable exception - pSeries (rpaphp), where the `slot_nr` cannot be determined until a device is actually inserted into the slot. In this scenario, the caller may pass -1 for `slot_nr`.

The following semantics are imposed when the caller passes `slot_nr == -1`. First, we no longer check for an existing `struct pci_slot`, as there may be many slots with `slot_nr` of -1. The other change in semantics is user-visible, which is the 'address' parameter presented in sysfs will

## consist solely of a dddd

bb tuple, where dddd is the PCI domain of the `struct pci_bus` and bb is the bus number. In other words, the devfn of the 'placeholder' slot will not be displayed.

## pci\_renumber\_slot

### LINUX

Kernel Hackers Manual June 2013

### Name

`pci_renumber_slot` — update `struct pci_slot` -> number

### Synopsis

```
void pci_renumber_slot (struct pci_slot * slot, int slot_nr);
```

## Arguments

*slot*

struct `pci_slot` to update

*slot\_nr*

new number for slot

## Description

The primary purpose of this interface is to allow callers who earlier created a placeholder slot in `pci_create_slot` by passing a -1 as `slot_nr`, to update their struct `pci_slot` with the correct *slot\_nr*.

# pci\_destroy\_slot

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_destroy_slot` — decrement refcount for physical PCI slot

## Synopsis

```
void pci_destroy_slot (struct pci_slot * slot);
```

## Arguments

*slot*

struct `pci_slot` to decrement

## Description

`struct pci_slot` is refcounted, so destroying them is really easy; we just call `kobject_put` on its `kobj` and let our release methods do the rest.

# pci\_hp\_create\_module\_link

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_hp_create_module_link` — create symbolic link to the hotplug driver module.

## Synopsis

```
void pci_hp_create_module_link (struct pci_slot * pci_slot);
```

## Arguments

*pci\_slot*

`struct pci_slot`

## Description

Helper function for `pci_hotplug_core.c` to create symbolic link to the hotplug driver module.

# pci\_hp\_remove\_module\_link

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_hp_remove_module_link` — remove symbolic link to the hotplug driver module.

### Synopsis

```
void pci_hp_remove_module_link (struct pci_slot * pci_slot);
```

### Arguments

*pci\_slot*  
struct pci\_slot

### Description

Helper function for `pci_hotplug_core.c` to remove symbolic link to the hotplug driver module.

# pci\_enable\_rom

## LINUX



## Name

`pci_enable_rom` — enable ROM decoding for a PCI device

## Synopsis

```
int pci_enable_rom (struct pci_dev * pdev);
```

## Arguments

*pdev*

PCI device to enable

## Description

Enable ROM decoding on *dev*. This involves simply turning on the last bit of the PCI ROM BAR. Note that some cards may share address decoders between the ROM and other resources, so enabling it may disable access to MMIO registers or other card memory.

# pci\_disable\_rom

## LINUX

## Name

`pci_disable_rom` — disable ROM decoding for a PCI device

## Synopsis

```
void pci_disable_rom (struct pci_dev * pdev);
```

## Arguments

*pdev*

PCI device to disable

## Description

Disable ROM decoding on a PCI device by turning off the last bit in the ROM BAR.

## pci\_map\_rom

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_map_rom` — map a PCI ROM to kernel space

## Synopsis

```
void __iomem * pci_map_rom (struct pci_dev * pdev, size_t *  
size);
```

## Arguments

*pdev*

pointer to pci device struct

*size*

pointer to receive size of pci window over ROM

## Return

kernel virtual pointer to image of ROM

Map a PCI ROM into kernel space. If ROM is boot video ROM, the shadow BIOS copy will be returned instead of the actual ROM.

# pci\_unmap\_rom

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_unmap_rom` — unmap the ROM from kernel space

## Synopsis

```
void pci_unmap_rom (struct pci_dev * pdev, void __iomem *  
rom);
```

## Arguments

*pdev*

pointer to pci device struct

*rom*

virtual address of the previous mapping

## Description

Remove a mapping of a previously mapped ROM

# pci\_enable\_sriov

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_enable_sriov` — enable the SR-IOV capability

## Synopsis

```
int pci_enable_sriov (struct pci_dev * dev, int nr_virtfn);
```

## Arguments

*dev*

the PCI device

`nr_virtfn`

number of virtual functions to enable

## Description

Returns 0 on success, or negative on failure.

# pci\_disable\_sriov

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_disable_sriov` — disable the SR-IOV capability

## Synopsis

```
void pci_disable_sriov (struct pci_dev * dev);
```

## Arguments

`dev`

the PCI device

# pci\_sriov\_migration

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_sriov_migration` — notify SR-IOV core of Virtual Function Migration

### Synopsis

```
irqreturn_t pci_sriov_migration (struct pci_dev * dev);
```

### Arguments

*dev*

the PCI device

### Description

Returns `IRQ_HANDLED` if the IRQ is handled, or `IRQ_NONE` if not.

Physical Function driver is responsible to register IRQ handler using VF Migration Interrupt Message Number, and call this function when the interrupt is generated by the hardware.

# pci\_num\_vf

## LINUX

## Name

`pci_num_vf` — return number of VFs associated with a PF device\_release\_driver

## Synopsis

```
int pci_num_vf (struct pci_dev * dev);
```

## Arguments

*dev*

the PCI device

## Description

Returns number of VFs, or 0 if SR-IOV is not enabled.

# pci\_read\_legacy\_io

## LINUX

## Name

`pci_read_legacy_io` — read byte(s) from legacy I/O port space

## Synopsis

```
ssize_t pci_read_legacy_io (struct file * filp, struct kobject  
* kobj, struct bin_attribute * bin_attr, char * buf, loff_t  
off, size_t count);
```

## Arguments

*filp*

open sysfs file

*kobj*

kobject corresponding to file to read from

*bin\_attr*

struct bin\_attribute for this file

*buf*

buffer to store results

*off*

offset into legacy I/O port space

*count*

number of bytes to read

## Description

Reads 1, 2, or 4 bytes from legacy I/O port space using an arch specific callback routine (`pci_legacy_read`).



# pci\_write\_legacy\_io

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_write_legacy_io` — write byte(s) to legacy I/O port space

### Synopsis

```
ssize_t pci_write_legacy_io (struct file * filp, struct  
kobject * kobj, struct bin_attribute * bin_attr, char * buf,  
loff_t off, size_t count);
```

### Arguments

*filp*

open sysfs file

*kobj*

kobject corresponding to file to read from

*bin\_attr*

struct bin\_attribute for this file

*buf*

buffer containing value to be written

*off*

offset into legacy I/O port space

*count*

number of bytes to write

## Description

Writes 1, 2, or 4 bytes from legacy I/O port space using an arch specific callback routine (`pci_legacy_write`).

# pci\_mmap\_legacy\_mem

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_mmap_legacy_mem` — map legacy PCI memory into user memory space

## Synopsis

```
int pci_mmap_legacy_mem (struct file * filp, struct kobject *  
kobj, struct bin_attribute * attr, struct vm_area_struct *  
vma);
```

## Arguments

*filp*

open sysfs file

*kobj*

kobject corresponding to device to be mapped

*attr*

struct bin\_attribute for this file

*vma*

struct vm\_area\_struct passed to mmap

## Description

Uses an arch specific callback, `pci_mmap_legacy_mem_page_range`, to mmap legacy memory space (first meg of bus space) into application virtual memory space.

## pci\_mmap\_legacy\_io

### LINUX

Kernel Hackers Manual June 2013

## Name

`pci_mmap_legacy_io` — map legacy PCI IO into user memory space

## Synopsis

```
int pci_mmap_legacy_io (struct file * filp, struct kobject *
kobj, struct bin_attribute * attr, struct vm_area_struct *
vma);
```

## Arguments

*filp*

open sysfs file

*kobj*

kobject corresponding to device to be mapped

*attr*

struct bin\_attribute for this file

*vma*

struct vm\_area\_struct passed to mmap

## Description

Uses an arch specific callback, `pci_mmap_legacy_io_page_range`, to mmap legacy IO space (first meg of bus space) into application virtual memory space. Returns `-ENOSYS` if the operation isn't supported

# pci\_adjust\_legacy\_attr

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_adjust_legacy_attr` — adjustment of legacy file attributes

## Synopsis

```
void __weak pci_adjust_legacy_attr (struct pci_bus * b, enum  
pci_mmap_state mmap_type);
```

## Arguments

*b*

bus to create files under

*mmap\_type*

I/O port or memory

## Description

Stub implementation. Can be overridden by arch if necessary.

# pci\_create\_legacy\_files

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_create_legacy_files` — create legacy I/O port and memory files

## Synopsis

```
void pci_create_legacy_files (struct pci_bus * b);
```

## Arguments

*b*

bus to create files under

## Description

Some platforms allow access to legacy I/O port and ISA memory space on a per-bus basis. This routine creates the files and ties them into their associated read, write and mmap files from `pci-sysfs.c`

On error unwind, but don't propagate the error to the caller as it is ok to set up the PCI bus without these files.

# pci\_mmap\_resource

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_mmap_resource` — map a PCI resource into user memory space

### Synopsis

```
int pci_mmap_resource (struct kobject * kobj, struct
bin_attribute * attr, struct vm_area_struct * vma, int
write_combine);
```

### Arguments

*kobj*

kobject for mapping

*attr*

struct bin\_attribute for the file being mapped

*vma*

struct vm\_area\_struct passed into the mmap

*write\_combine*

1 for write\_combine mapping

### Description

Use the regular PCI mapping routines to map a PCI resource into userspace.

# pci\_remove\_resource\_files

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_remove_resource_files` — cleanup resource files

### Synopsis

```
void pci_remove_resource_files (struct pci_dev * pdev);
```

### Arguments

*pdev*

dev to cleanup

### Description

If we created resource files for *pdev*, remove them from sysfs and free their resources.

# pci\_create\_resource\_files

## LINUX

## Name

`pci_create_resource_files` — create resource files in sysfs for *dev*

## Synopsis

```
int pci_create_resource_files (struct pci_dev * pdev);
```

## Arguments

*pdev*

dev in question

## Description

Walk the resources in *pdev* creating files for each resource available.

# pci\_write\_rom

## LINUX

## Name

`pci_write_rom` — used to enable access to the PCI ROM display



## Synopsis

```
ssize_t pci_write_rom (struct file * filp, struct kobject *
kobj, struct bin_attribute * bin_attr, char * buf, loff_t
off, size_t count);
```

## Arguments

*filp*

sysfs file

*kobj*

kernel object handle

*bin\_attr*

struct bin\_attribute for this file

*buf*

user input

*off*

file offset

*count*

number of byte in input

## Description

writing anything except 0 enables it

## pci\_read\_rom

**LINUX**

## Name

`pci_read_rom` — read a PCI ROM

## Synopsis

```
ssize_t pci_read_rom (struct file * filp, struct kobject *  
kobj, struct bin_attribute * bin_attr, char * buf, loff_t  
off, size_t count);
```

## Arguments

*filp*

sysfs file

*kobj*

kernel object handle

*bin\_attr*

struct bin\_attribute for this file

*buf*

where to put the data we read from the ROM

*off*

file offset

*count*

number of bytes to read

## Description

Put *count* bytes starting at *off* into *buf* from the ROM in the PCI device corresponding to *kobj*.

# pci\_remove\_sysfs\_dev\_files

## LINUX

Kernel Hackers Manual June 2013

### Name

`pci_remove_sysfs_dev_files` — cleanup PCI specific sysfs files

### Synopsis

```
void pci_remove_sysfs_dev_files (struct pci_dev * pdev);
```

### Arguments

*pdev*

device whose entries we should free

### Description

Cleanup when *pdev* is removed from sysfs.

## 9.6. PCI Hotplug Support Library

### `__pci_hp_register`

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`__pci_hp_register` — register a `hotplug_slot` with the PCI hotplug subsystem

#### Synopsis

```
int __pci_hp_register (struct hotplug_slot * slot, struct
pci_bus * bus, int devnr, const char * name, struct module *
owner, const char * mod_name);
```

#### Arguments

*slot*

pointer to the struct `hotplug_slot` to register

*bus*

bus this slot is on

*devnr*

device number

*name*

name registered with kobject core

*owner*

caller module owner

*mod\_name*

caller module name

## 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_hp_deregister` — deregister a `hotplug_slot` with the PCI hotplug subsystem

## Synopsis

```
int pci_hp_deregister (struct hotplug_slot * hotplug);
```

## Arguments

*hotplug*

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

## LINUX

Kernel Hackers Manual June 2013

## Name

`pci_hp_change_slot_info` — changes the slot's information structure in the core

## Synopsis

```
int __must_check pci_hp_change_slot_info (struct hotplug_slot
* hotplug, struct hotplug_slot_info * info);
```

## Arguments

*hotplug*

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.

## 9.7. MCA Architecture

### 9.7.1. MCA Device Functions

Refer to the file `arch/x86/kernel/mca_32.c` for more information.

### 9.7.2. MCA Bus DMA

## `mca_enable_dma`

### LINUX

Kernel Hackers Manual June 2013

### Name

`mca_enable_dma` — channel to enable DMA on

### Synopsis

```
void mca_enable_dma (unsigned int dmanr);
```

## Arguments

*dmnr*

DMA channel

## Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.

# mca\_disable\_dma

## LINUX

Kernel Hackers Manual June 2013

## Name

`mca_disable_dma` — channel to disable DMA on

## Synopsis

```
void mca_disable_dma (unsigned int dmnr);
```

## Arguments

*dmnr*

DMA channel

## Description

Enable the MCA bus DMA on a channel. This can be called from IRQ context.



# mca\_set\_dma\_addr

## LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

## 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`

### LINUX

## 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

## LINUX

## 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

### LINUX

Kernel Hackers Manual June 2013

## Name

`mca_set_dma_mode` — set the DMA mode

## Synopsis

```
void mca_set_dma_mode (unsigned int dmanr, unsigned int mode);
```

## Arguments

*dmamr*

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 10. Firmware Interfaces

## 10.1. DMI Interfaces

### dmi\_check\_system

**LINUX**

Kernel Hackers Manual June 2013

#### Name

`dmi_check_system` — check system DMI data

#### Synopsis

```
int dmi_check_system (const struct dmi_system_id * list);
```

#### Arguments

*list*

array of `dmi_system_id` structures to match against All non-null elements of the list must match their slot's (field index's) data (i.e., each list string must be a substring of the specified DMI slot's string data) to be considered a successful match.

#### Description

Walk the blacklist table running matching functions until someone returns non zero or we hit the end. Callback function is called for each successful match. Returns the number of matches.

# dmi\_first\_match

## LINUX

Kernel Hackers Manual June 2013

### Name

`dmi_first_match` — find `dmi_system_id` structure matching system DMI data

### Synopsis

```
const struct dmi_system_id * dmi_first_match (const struct
dmi_system_id * list);
```

### Arguments

*list*

array of `dmi_system_id` structures to match against All non-null elements of the list must match their slot's (field index's) data (i.e., each list string must be a substring of the specified DMI slot's string data) to be considered a successful match.

### Description

Walk the blacklist table until the first match is found. Return the pointer to the matching entry or NULL if there's no match.



# dmi\_get\_system\_info

## LINUX

Kernel Hackers Manual June 2013

### Name

`dmi_get_system_info` — return DMI data value

### Synopsis

```
const char * dmi_get_system_info (int field);
```

### Arguments

*field*

data index (see enum `dmi_field`)

### Description

Returns one DMI data value, can be used to perform complex DMI data checks.

# dmi\_name\_in\_vendors

## LINUX

## Name

`dmi_name_in_vendors` — Check if string is anywhere in the DMI vendor information.

## Synopsis

```
int dmi_name_in_vendors (const char * str);
```

## Arguments

*str*

Case sensitive Name

# dmi\_find\_device

## LINUX

## Name

`dmi_find_device` — find onboard device by type/name

## Synopsis

```
const struct dmi_device * dmi_find_device (int type, const  
char * name, const struct dmi_device * from);
```

## Arguments

*type*

device type or `DMI_DEV_TYPE_ANY` to match all device types

*name*

device name string or `NULL` to match all

*from*

previous device found in search, or `NULL` for new search.

## Description

Iterates through the list of known onboard devices. If a device is found with a matching *vendor* and *device*, a pointer to its device structure is returned. Otherwise, `NULL` is returned. A new search is initiated by passing `NULL` as the *from* argument. If *from* is not `NULL`, searches continue from next device.

## dmi\_get\_date

### LINUX

Kernel Hackers Manual June 2013

### Name

`dmi_get_date` — parse a DMI date

## Synopsis

```
bool dmi_get_date (int field, int * yearp, int * monthp, int *
dayp);
```

## Arguments

*field*

data index (see enum dmi\_field)

*yearp*

optional out parameter for the year

*monthp*

optional out parameter for the month

*dayp*

optional out parameter for the day

## Description

The date field is assumed to be in the form resembling [mm[/dd]]/yy[yy] and the result is stored in the out parameters any or all of which can be omitted.

If the field doesn't exist, all out parameters are set to zero and false is returned. Otherwise, true is returned with any invalid part of date set to zero.

On return, year, month and day are guaranteed to be in the range of [0,9999], [0,12] and [0,31] respectively.

## dmi\_walk

### LINUX

Kernel Hackers Manual June 2013

## Name

dmi\_walk — Walk the DMI table and get called back for every record

## Synopsis

```
int dmi_walk (void (*decode) (const struct dmi_header *, void
*), void * private_data);
```

## Arguments

*decode*

Callback function

*private\_data*

Private data to be passed to the callback function

## Description

Returns -1 when the DMI table can't be reached, 0 on success.

## dmi\_match

### LINUX

Kernel Hackers Manual June 2013

### Name

`dmi_match` — compare a string to the dmi field (if exists)

## Synopsis

```
bool dmi_match (enum dmi_field f, const char * str);
```

## Arguments

*f*

DMI field identifier

*str*

string to compare the DMI field to

## Description

Returns true if the requested field equals to the str (including NULL).

## 10.2. EDD Interfaces

### edd\_show\_raw\_data

#### LINUX

Kernel Hackers Manual June 2013

#### Name

edd\_show\_raw\_data — copies raw data to buffer for userspace to parse

#### Synopsis

```
ssize_t edd_show_raw_data (struct edd_device * edev, char *  
buf);
```

## Arguments

*edev*

target edd\_device

*buf*

output buffer

## Returns

number of bytes written, or -EINVAL on failure

# edd\_release

## LINUX

Kernel Hackers Manual June 2013

## Name

edd\_release — free edd structure

## Synopsis

```
void edd_release (struct kobject * kobj);
```

## Arguments

*kobj*

kobject of edd structure

## Description

This is called when the refcount of the edd structure reaches 0. This should happen right after we unregister, but just in case, we use the release callback anyway.

# edd\_dev\_is\_type

## LINUX

Kernel Hackers Manual June 2013

## Name

`edd_dev_is_type` — is this EDD device a 'type' device?

## Synopsis

```
int edd_dev_is_type (struct edd_device * edev, const char *  
type);
```

## Arguments

*edev*

target edd\_device

*type*

a host bus or interface identifier string per the EDD spec

## Description

Returns 1 (TRUE) if it is a 'type' device, 0 otherwise.



# edd\_get\_pci\_dev

## LINUX

Kernel Hackers Manual June 2013

### Name

`edd_get_pci_dev` — finds `pci_dev` that matches `edev`

### Synopsis

```
struct pci_dev * edd_get_pci_dev (struct edd_device * edev);
```

### Arguments

`edev`

`edd_device`

### Description

Returns `pci_dev` if found, or NULL

# edd\_init

## LINUX

Kernel Hackers Manual June 2013

### Name

`edd_init` — creates sysfs tree of EDD data

## **Synopsis**

```
int edd_init ( void );
```

## **Arguments**

*void*

no arguments

# Chapter 11. Security Framework

## security\_init

**LINUX**

Kernel Hackers Manual June 2013

### Name

`security_init` — initializes the security framework

### Synopsis

```
int security_init ( void );
```

### Arguments

*void*

no arguments

### Description

This should be called early in the kernel initialization sequence.

## security\_module\_enable

**LINUX**

## Name

`security_module_enable` — Load given security module on boot ?

## Synopsis

```
int security_module_enable (struct security_operations * ops);
```

## Arguments

*ops*

a pointer to the struct `security_operations` that is to be checked.

## Description

Each LSM must pass this method before registering its own operations to avoid security registration races. This method may also be used to check if your LSM is currently loaded during kernel initialization.

## Return true if

-The passed LSM is the one chosen by user at boot time, -or the passed LSM is configured as the default and the user did not choose an alternate LSM at boot time. Otherwise, return false.

## register\_security

**LINUX**

## 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 allows a security module to register itself with the kernel security subsystem. Some rudimentary checking is done on the *ops* value passed to this function. You'll need to check first if your LSM is allowed to register its *ops* by calling `security_module_enable(ops)`.

If there is already a security module registered with the kernel, an error will be returned. Otherwise 0 is returned on success.

## securityfs\_create\_file

**LINUX**

## Name

`securityfs_create_file` — create a file in the securityfs filesystem

## Synopsis

```
struct dentry * securityfs_create_file (const char * name,
mode_t mode, struct dentry * parent, void * data, const 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 securityfs filesystem.

*data*

a pointer to something that the caller will want to get to later on. The `inode.i_private` 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 securityfs. It allows for a wide range of flexibility in creating a file, or a directory (if you want to create a directory, the

`securityfs_create_dir` function is recommended to be used instead).

This function returns a pointer to a dentry if it succeeds. This pointer must be passed to the `securityfs_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, the function will return the error value (via `ERR_PTR`).

If `securityfs` is not enabled in the kernel, the value `-ENODEV` is returned.

## securityfs\_create\_dir

### LINUX

Kernel Hackers Manual June 2013

### Name

`securityfs_create_dir` — create a directory in the `securityfs` filesystem

### Synopsis

```
struct dentry * securityfs_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 `securityfs` filesystem.

## Description

This function creates a directory in `securityfs` with the given *name*.

This function returns a pointer to a `dentry` if it succeeds. This pointer must be passed to the `securityfs_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 `securityfs` is not enabled in the kernel, the value `-ENODEV` is 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.

## securityfs\_remove

### LINUX

Kernel Hackers Manual June 2013

### Name

`securityfs_remove` — removes a file or directory from the `securityfs` filesystem

### Synopsis

```
void securityfs_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 securityfs that was previously created with a call to another securityfs function (like `securityfs_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.



# Chapter 12. Audit Interfaces

## audit\_log\_start

### LINUX

Kernel Hackers Manual June 2013

### Name

`audit_log_start` — obtain an audit buffer

### Synopsis

```
struct audit_buffer * audit_log_start (struct audit_context *  
ctx, gfp_t gfp_mask, int type);
```

### Arguments

*ctx*

audit\_context (may be NULL)

*gfp\_mask*

type of allocation

*type*

audit message type

### Description

Returns `audit_buffer` pointer on success or NULL on error.

Obtain an audit buffer. This routine does locking to obtain the audit buffer, but then no locking is required for calls to `audit_log_*format`. If the task (`ctx`) is a task that is

currently in a syscall, then the syscall is marked as auditable and an audit record will be written at syscall exit. If there is no associated task, then task context (ctx) should be NULL.

## audit\_log\_format

### LINUX

Kernel Hackers Manual June 2013

### Name

`audit_log_format` — format a message into the audit buffer.

### Synopsis

```
void audit_log_format (struct audit_buffer * ab, const char *  
fmt, ...);
```

### Arguments

*ab*

audit\_buffer

*fmt*

format string @...: optional parameters matching *fmt* string

...

variable arguments

## Description

All the work is done in `audit_log_vformat`.

# audit\_log\_untrustedstring

## LINUX

Kernel Hackers Manual June 2013

## Name

`audit_log_untrustedstring` — log a string that may contain random characters

## Synopsis

```
void audit_log_untrustedstring (struct audit_buffer * ab,  
const char * string);
```

## Arguments

*ab*

`audit_buffer`

*string*

string to be logged

## Description

Same as `audit_log_n_untrustedstring`, except that `strlen` is used to determine string length.

# audit\_log\_end

## LINUX

Kernel Hackers Manual June 2013

### Name

`audit_log_end` — end one audit record

### Synopsis

```
void audit_log_end (struct audit_buffer * ab);
```

### Arguments

*ab*

the `audit_buffer`

### Description

The `netlink_*` functions cannot be called inside an irq context, so the audit buffer is placed on a queue and a tasklet is scheduled to remove them from the queue outside the irq context. May be called in any context.

# audit\_log

## LINUX

## Name

`audit_log` — Log an audit record

## Synopsis

```
void audit_log (struct audit_context * ctx, gfp_t gfp_mask,  
int type, const char * fmt, ...);
```

## Arguments

*ctx*

audit context

*gfp\_mask*

type of allocation

*type*

audit message type

*fmt*

format string to use @...: variable parameters matching the format string

...

variable arguments

## Description

This is a convenience function that calls `audit_log_start`, `audit_log_vformat`, and `audit_log_end`. It may be called in any context.

# audit\_alloc

## LINUX

Kernel Hackers Manual June 2013

### Name

`audit_alloc` — allocate an audit context block for a task

### Synopsis

```
int audit_alloc (struct task_struct * tsk);
```

### Arguments

*tsk*

task

### Description

Filter on the task information and allocate a per-task audit context if necessary. Doing so turns on system call auditing for the specified task. This is called from `copy_process`, so no lock is needed.

# audit\_free

## LINUX



## Name

`audit_free` — free a per-task audit context

## Synopsis

```
void audit_free (struct task_struct * tsk);
```

## Arguments

*tsk*

task whose audit context block to free

## Description

Called from `copy_process` and `do_exit`

# audit\_syscall\_entry

## LINUX

## Name

`audit_syscall_entry` — fill in an audit record at syscall entry

## Synopsis

```
void audit_syscall_entry (int arch, int major, unsigned long  
a1, unsigned long a2, unsigned long a3, unsigned long a4);
```

## Arguments

*arch*

architecture type

*major*

major syscall type (function)

*a1*

additional syscall register 1

*a2*

additional syscall register 2

*a3*

additional syscall register 3

*a4*

additional syscall register 4

## Description

Fill in audit context at syscall entry. This only happens if the audit context was created when the task was created and the state or filters demand the audit context be built. If the state from the per-task filter or from the per-syscall filter is `AUDIT_RECORD_CONTEXT`, then the record will be written at syscall exit time (otherwise, it will only be written if another part of the kernel requests that it be written).

# audit\_syscall\_exit

## LINUX

Kernel Hackers Manual June 2013

### Name

`audit_syscall_exit` — deallocate audit context after a system call

### Synopsis

```
void audit_syscall_exit (int valid, long return_code);
```

### Arguments

*valid*

success/failure flag

*return\_code*

syscall return value

### Description

Tear down after system call. If the audit context has been marked as auditable (either because of the `AUDIT_RECORD_CONTEXT` state from filtering, or because some other part of the kernel write an audit message), then write out the syscall information. In call cases, free the names stored from `getname`.

# \_\_audit\_getname

## LINUX

## Name

`__audit_getname` — add a name to the list

## Synopsis

```
void __audit_getname (const char * name);
```

## Arguments

*name*

name to add

## Description

Add a name to the list of audit names for this context. Called from `fs/namei.c:getname`.

# `__audit_inode`

## LINUX

## Name

`__audit_inode` — store the inode and device from a lookup

## Synopsis

```
void __audit_inode (const char * name, const struct dentry *
dentry);
```

## Arguments

*name*

name being audited

*dentry*

dentry being audited

## Description

Called from fs/namei.c:path\_lookup.

# auditsc\_get\_stamp

## LINUX

Kernel Hackers Manual June 2013

## Name

auditsc\_get\_stamp — get local copies of audit\_context values

## Synopsis

```
int auditsc_get_stamp (struct audit_context * ctx, struct
timespec * t, unsigned int * serial);
```

## Arguments

*ctx*

audit\_context for the task

*t*

timespec to store time recorded in the audit\_context

*serial*

serial value that is recorded in the audit\_context

## Description

Also sets the context as auditable.

# audit\_set\_loginuid

**LINUX**

Kernel Hackers Manual June 2013

## Name

audit\_set\_loginuid — set a task's audit\_context loginuid

## Synopsis

```
int audit_set_loginuid (struct task_struct * task, uid_t  
loginuid);
```

## Arguments

*task*

task whose audit context is being modified

*loginuid*

loginuid value

## Description

Returns 0.

Called (set) from fs/proc/base.c::proc\_loginuid\_write.

# \_\_audit\_mq\_open

## LINUX

Kernel Hackers Manual June 2013

## Name

\_\_audit\_mq\_open — record audit data for a POSIX MQ open

## Synopsis

```
void __audit_mq_open (int oflag, mode_t mode, struct mq_attr *
attr);
```

## Arguments

*oflag*

open flag

*mode*

mode bits

*attr*

queue attributes

## \_\_audit\_mq\_sendrecv

### LINUX

Kernel Hackers Manual June 2013

## Name

`__audit_mq_sendrecv` — record audit data for a POSIX MQ timed send/receive

## Synopsis

```
void __audit_mq_sendrecv (mqd_t mqdes, size_t msg_len,  
unsigned int msg_prio, const struct timespec * abs_timeout);
```

## Arguments

*mqdes*

MQ descriptor



*msg\_len*

Message length

*msg\_prio*

Message priority

*abs\_timeout*

Message timeout in absolute time

## **\_\_audit\_mq\_notify**

### **LINUX**

Kernel Hackers Manual June 2013

### **Name**

`__audit_mq_notify` — record audit data for a POSIX MQ notify

### **Synopsis**

```
void __audit_mq_notify (mqd_t mqdes, const struct sigevent *
notification);
```

### **Arguments**

*mqdes*

MQ descriptor

*notification*

Notification event

# \_\_audit\_mq\_getsetattr

## LINUX

Kernel Hackers Manual June 2013

### Name

`__audit_mq_getsetattr` — record audit data for a POSIX MQ get/set attribute

### Synopsis

```
void __audit_mq_getsetattr (mqd_t mqdes, struct mq_attr *  
mqstat);
```

### Arguments

*mqdes*

MQ descriptor

*mqstat*

MQ flags

# \_\_audit\_ipc\_obj

## LINUX

Kernel Hackers Manual June 2013

### Name

`__audit_ipc_obj` — record audit data for ipc object

## Synopsis

```
void __audit_ipc_obj (struct kern_ipc_perm * icp);
```

## Arguments

*icp*

ipc permissions

## \_\_audit\_ipc\_set\_perm

### LINUX

Kernel Hackers Manual June 2013

## Name

\_\_audit\_ipc\_set\_perm — record audit data for new ipc permissions

## Synopsis

```
void __audit_ipc_set_perm (unsigned long qbytes, uid_t uid,  
gid_t gid, mode_t mode);
```

## Arguments

*qbytes*

msgq bytes

*uid*

msgq user id

*gid*

msgq group id

*mode*

msgq mode (permissions)

## Description

Called only after `audit_ipc_obj`.

# audit\_socketcall

**LINUX**

Kernel Hackers Manual June 2013

## Name

`audit_socketcall` — record audit data for `sys_socketcall`

## Synopsis

```
void audit_socketcall (int nargs, unsigned long * args);
```

## Arguments

*nargs*

number of args

*args*

args array

## **\_\_audit\_fd\_pair**

### **LINUX**

Kernel Hackers Manual June 2013

### **Name**

`__audit_fd_pair` — record audit data for pipe and socketpair

### **Synopsis**

```
void __audit_fd_pair (int fd1, int fd2);
```

### **Arguments**

*fd1*

the first file descriptor

*fd2*

the second file descriptor

## **audit\_sockaddr**

### **LINUX**

## Name

`audit_sockaddr` — record audit data for `sys_bind`, `sys_connect`, `sys_sendto`

## Synopsis

```
int audit_sockaddr (int len, void * a);
```

## Arguments

*len*

data length in user space

*a*

data address in kernel space

## Description

Returns 0 for success or NULL context or < 0 on error.

# \_\_audit\_signal\_info

## LINUX

## Name

`__audit_signal_info` — record signal info for shutting down audit subsystem

## Synopsis

```
int __audit_signal_info (int sig, struct task_struct * t);
```

## Arguments

*sig*

signal value

*t*

task being signaled

## Description

If the audit subsystem is being terminated, record the task (pid) and uid that is doing that.

# \_\_audit\_log\_bprm\_fcaps

## LINUX

Kernel Hackers Manual June 2013

## Name

`__audit_log_bprm_fcaps` — store information about a loading bprm and relevant fcaps

## Synopsis

```
int __audit_log_bprm_fcaps (struct linux_binprm * bprm, const
struct cred * new, const struct cred * old);
```

## Arguments

*bprm*

pointer to the bprm being processed

*new*

the proposed new credentials

*old*

the old credentials

## Description

Simply check if the proc already has the caps given by the file and if not store the priv escalation info for later auditing at the end of the syscall

-Eric

# \_\_audit\_log\_capset

## LINUX

Kernel Hackers Manual June 2013

## Name

`__audit_log_capset` — store information about the arguments to the capset syscall

## Synopsis

```
void __audit_log_capset (pid_t pid, const struct cred * new,  
const struct cred * old);
```



## Arguments

*pid*

target pid of the capset call

*new*

the new credentials

*old*

the old (current) credentials

## Description

Record the arguments userspace sent to `sys_capset` for later printing by the audit system if applicable

# audit\_core\_dumps

**LINUX**

Kernel Hackers Manual June 2013

## Name

`audit_core_dumps` — record information about processes that end abnormally

## Synopsis

```
void audit_core_dumps (long signr);
```

## Arguments

*signr*

signal value

## Description

If a process ends with a core dump, something fishy is going on and we should record the event for investigation.

# audit\_receive\_filter

## LINUX

Kernel Hackers Manual June 2013

## Name

`audit_receive_filter` — apply all rules to the specified message type

## Synopsis

```
int audit_receive_filter (int type, int pid, int uid, int  
seq, void * data, size_t datasz, uid_t loginuid, u32  
sessionid, u32 sid);
```

## Arguments

*type*

audit message type

*pid*

target pid for netlink audit messages

*uid*

target uid for netlink audit messages

*seq*

netlink audit message sequence (serial) number

*data*

payload data

*datasz*

size of payload data

*loginuid*

loginuid of sender

*sessionid*

sessionid for netlink audit message

*sid*

SE Linux Security ID of sender



# Chapter 13. Accounting Framework

## sys\_acct

### LINUX

Kernel Hackers Manual June 2013

### Name

`sys_acct` — enable/disable process accounting

### Synopsis

```
long sys_acct (const char __user * name);
```

### Arguments

*name*

file name for accounting records or NULL to shutdown accounting

### Description

Returns 0 for success or negative errno values for failure.

`sys_acct` is the only system call needed to implement process accounting. It takes the name of the file where accounting records should be written. If the filename is NULL, accounting will be shutdown.

# acct\_auto\_close\_mnt

## LINUX

Kernel Hackers Manual June 2013

### Name

`acct_auto_close_mnt` — turn off a filesystem's accounting if it is on

### Synopsis

```
void acct_auto_close_mnt (struct vfsmount * m);
```

### Arguments

*m*

vfsmount being shut down

### Description

If the accounting is turned on for a file in the subtree pointed to to by *m*, turn accounting off. Done when *m* is about to die.

# acct\_auto\_close

## LINUX

## Name

`acct_auto_close` — turn off a filesystem's accounting if it is on

## Synopsis

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

## Arguments

*sb*

super block for the filesystem

## Description

If the accounting is turned on for a file in the filesystem pointed to by *sb*, turn accounting off.

# acct\_collect

## LINUX

## Name

`acct_collect` — collect accounting information into `pacct_struct`

## Synopsis

```
void acct_collect (long exitcode, int group_dead);
```

## Arguments

*exitcode*

task exit code

*group\_dead*

not 0, if this thread is the last one in the process.

## acct\_process

### LINUX

Kernel Hackers Manual June 2013

## Name

`acct_process` — now just a wrapper around `acct_process_in_ns`, which in turn is a wrapper around `do_acct_process`.

## Synopsis

```
void acct_process ( void );
```



## **Arguments**

*void*

no arguments

## **Description**

handles process accounting for an exiting task



# Chapter 14. Block Devices

## blk\_get\_backing\_dev\_info

### LINUX

Kernel Hackers Manual June 2013

### 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\_delay\_queue

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_delay_queue` — restart queueing after defined interval

### Synopsis

```
void blk_delay_queue (struct request_queue * q, unsigned long  
msecs);
```

### Arguments

*q*

The struct `request_queue` in question

*msecs*

Delay in msecs

### Description

Sometimes queueing needs to be postponed for a little while, to allow resources to come back. This function will make sure that queueing is restarted around the specified time. Queue lock must be held.

# blk\_start\_queue

## LINUX

## Name

`blk_start_queue` — restart a previously stopped queue

## Synopsis

```
void blk_start_queue (struct request_queue * q);
```

## Arguments

*q*

The struct `request_queue` 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

## LINUX

## Name

`blk_stop_queue` — stop a queue

## Synopsis

```
void blk_stop_queue (struct request_queue * q);
```

## Arguments

*q*

The struct request\_queue 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

### LINUX

Kernel Hackers Manual June 2013

## 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 that 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.

This function does not cancel any asynchronous activity arising out of elevator or throttling code. That would require `elevator_exit` and `blk_throtl_exit` to be called with queue lock initialized.

## `__blk_run_queue`

### LINUX

Kernel Hackers Manual June 2013

## Name

`__blk_run_queue` — run a single device queue

## Synopsis

```
void __blk_run_queue (struct request_queue * q);
```

## Arguments

*q*

The queue to run

## Description

See *blk\_run\_queue*. This variant must be called with the queue lock held and interrupts disabled.

# blk\_run\_queue\_async

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_run_queue_async` — run a single device queue in workqueue context

## Synopsis

```
void blk_run_queue_async (struct request_queue * q);
```

## Arguments

*q*

The queue to run



## Description

Tells kblockd to perform the equivalent of *blk\_run\_queue* on behalf of us.

# blk\_run\_queue

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_run_queue` — run a single device queue

## Synopsis

```
void blk_run_queue (struct request_queue * q);
```

## Arguments

*q*

The queue to run

## Description

Invoke request handling on this queue, if it has pending work to do. May be used to restart queueing when a request has completed.

# blk\_cleanup\_queue

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_cleanup_queue` — shutdown a request queue

### Synopsis

```
void blk_cleanup_queue (struct request_queue * q);
```

### Arguments

*q*  
request queue to shutdown

### Description

Mark *q* DYING, drain all pending requests, mark *q* DEAD, destroy and put it. All future requests will be failed immediately with -ENODEV.

# blk\_init\_queue

## LINUX

## Name

`blk_init_queue` — prepare a request queue for use with a block device

## Synopsis

```
struct request_queue * 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; this lock will be taken also from interrupt context, so irq disabling is needed for it.

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\_make\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_make_request` — given a bio, allocate a corresponding struct request.

## Synopsis

```
struct request * blk_make_request (struct request_queue * q,  
struct bio * bio, gfp_t gfp_mask);
```

## Arguments

*q*

target request queue

*bio*

The bio describing the memory mappings that will be submitted for IO. It may be a chained-bio properly constructed by block/bio layer.

*gfp\_mask*

gfp flags to be used for memory allocation

## Description

`blk_make_request` is the parallel of `generic_make_request` for `BLOCK_PC` type commands. Where the struct request needs to be farther initialized by the caller. It is passed a struct bio, which describes the memory info of the I/O transfer.

The caller of `blk_make_request` must make sure that `bi_io_vec` are set to describe the memory buffers. That `bio_data_dir` will return the needed direction of the request. (And all bio's in the passed bio-chain are properly set accordingly)

If called under none-sleepable conditions, mapped bio buffers must not need bouncing, by calling the appropriate masked or flagged allocator, suitable for the target device. Otherwise the call to `blk_queue_bounce` will BUG.

## WARNING

When allocating/cloning a bio-chain, careful consideration should be given to how you allocate bios. In particular, you cannot use `__GFP_WAIT` for anything but the first bio in the chain. Otherwise you risk waiting for IO completion of a bio that hasn't been submitted yet, thus resulting in a deadlock. Alternatively bios should be allocated using `bio_kmalloc` instead of `bio_alloc`, as that avoids the mempool deadlock. If possible a big IO should be split into smaller parts when allocation fails. Partial allocation should not be an error, or you risk a live-lock.

# blk\_requeue\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_requeue_request` — put a request back on queue

## Synopsis

```
void blk_requeue_request (struct request_queue * 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

**LINUX**

Kernel Hackers Manual June 2013

## Name

blk\_insert\_request — insert a special request into a request queue

## Synopsis

```
void blk_insert_request (struct request_queue * q, struct  
request * rq, int at_head, void * data);
```

## 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

## 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_TYPE_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.

# part\_round\_stats

## LINUX

Kernel Hackers Manual June 2013

### Name

`part_round_stats` — Round off the performance stats on a struct `disk_stats`.

### Synopsis

```
void part_round_stats (int cpu, struct hd_struct * part);
```

### Arguments

*cpu*

cpu number for stats access

*part*

target partition

### Description

The average IO queue length and utilisation statistics are maintained by observing the current state of the queue length and the amount of time it has been in this state for.

Normally, that accounting is done on IO completion, but that can result in more than a second's worth of IO being accounted for within any one second, leading to >100% utilisation. To deal with that, we call this function to do a round-off before returning the results when reading `/proc/diskstats`. This accounts immediately for all queue usage up to the current jiffies and restarts the counters again.



# blk\_add\_request\_payload

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_add_request_payload` — add a payload to a request

### Synopsis

```
void blk_add_request_payload (struct request * rq, struct page  
* page, unsigned int len);
```

### Arguments

*rq*

request to update

*page*

page backing the payload

*len*

length of the payload.

### Description

This allows to later add a payload to an already submitted request by a block driver. The driver needs to take care of freeing the payload itself.

Note that this is a quite horrible hack and nothing but handling of discard requests should ever use it.

# submit\_bio

## LINUX

Kernel Hackers Manual June 2013

### Name

`submit_bio` — submit a bio to the block device layer for I/O

### 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.

# blk\_rq\_check\_limits

## LINUX

## Name

`blk_rq_check_limits` — Helper function to check a request for the queue limit

## Synopsis

```
int blk_rq_check_limits (struct request_queue * q, struct  
request * rq);
```

## Arguments

*q*

the queue

*rq*

the request being checked

## Description

*rq* may have been made based on weaker limitations of upper-level queues in request stacking drivers, and it may violate the limitation of *q*. Since the block layer and the underlying device driver trust *rq* after it is inserted to *q*, it should be checked against *q* before the insertion using this generic function.

This function should also be useful for request stacking drivers in some cases below, so export this function. Request stacking drivers like request-based dm may change the queue limits while requests are in the queue (e.g. dm's table swapping). Such request stacking drivers should check those requests against the new queue limits again when they dispatch those requests, although such checkings are also done against the old queue limits when submitting requests.

# blk\_insert\_cloned\_request

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_insert_cloned_request` — Helper for stacking drivers to submit a request

### Synopsis

```
int blk_insert_cloned_request (struct request_queue * q,  
struct request * rq);
```

### Arguments

*q*  
the queue to submit the request

*rq*  
the request being queued

# blk\_rq\_err\_bytes

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_rq_err_bytes` — determine number of bytes till the next failure boundary

## Synopsis

```
unsigned int blk_rq_err_bytes (const struct request * rq);
```

## Arguments

*rq*

request to examine

## Description

A request could be merge of IOs which require different failure handling. This function determines the number of bytes which can be failed from the beginning of the request without crossing into area which need to be retried further.

## Return

The number of bytes to fail.

## Context

queue\_lock must be held.

# blk\_peek\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

blk\_peek\_request — peek at the top of a request queue

## Synopsis

```
struct request * blk_peek_request (struct request_queue * q);
```

## Arguments

*q*

request queue to peek at

## Description

Return the request at the top of *q*. The returned request should be started using `blk_start_request` before LLD starts processing it.

## Return

Pointer to the request at the top of *q* if available. Null otherwise.

## Context

`queue_lock` must be held.

# blk\_start\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_start_request` — start request processing on the driver

## Synopsis

```
void blk_start_request (struct request * req);
```

## Arguments

*req*

request to dequeue

## Description

Dequeue *req* and start timeout timer on it. This hands off the request to the driver.

Block internal functions which don't want to start timer should call

`blk_dequeue_request`.

## Context

`queue_lock` must be held.

# blk\_fetch\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_fetch_request` — fetch a request from a request queue

## Synopsis

```
struct request * blk_fetch_request (struct request_queue * q);
```

## Arguments

*q*

request queue to fetch a request from

## Description

Return the request at the top of *q*. The request is started on return and LLD can start processing it immediately.

## Return

Pointer to the request at the top of *q* if available. Null otherwise.

## Context

queue\_lock must be held.

# blk\_update\_request

**LINUX**

Kernel Hackers Manual June 2013

## Name

blk\_update\_request — Special helper function for request stacking drivers



## Synopsis

```
bool blk_update_request (struct request * req, int error,
unsigned int nr_bytes);
```

## Arguments

*req*

the request being processed

*error*

0 for success, < 0 for error

*nr\_bytes*

number of bytes to complete *req*

## Description

Ends I/O on a number of bytes attached to *req*, but doesn't complete the request structure even if *req* doesn't have leftover. If *req* has leftover, sets it up for the next range of segments.

This special helper function is only for request stacking drivers (e.g. request-based dm) so that they can handle partial completion. Actual device drivers should use `blk_end_request` instead.

Passing the result of `blk_rq_bytes` as *nr\_bytes* guarantees `false` return from this function.

## Return

`false` - this request doesn't have any more data `true` - this request has more data

# blk\_unprep\_request

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_unprep_request` — unprepare a request

### Synopsis

```
void blk_unprep_request (struct request * req);
```

### Arguments

*req*

the request

### Description

This function makes a request ready for complete resubmission (or completion). It happens only after all error handling is complete, so represents the appropriate moment to deallocate any resources that were allocated to the request in the `prep_rq_fn`. The queue lock is held when calling this.

# blk\_end\_request

## LINUX

## Name

`blk_end_request` — Helper function for drivers to complete the request.

## Synopsis

```
bool blk_end_request (struct request * rq, int error, unsigned  
int nr_bytes);
```

## Arguments

*rq*

the request being processed

*error*

0 for success, < 0 for error

*nr\_bytes*

number of bytes to complete

## Description

Ends I/O on a number of bytes attached to *rq*. If *rq* has leftover, sets it up for the next range of segments.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# blk\_end\_request\_all

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_end_request_all` — Helper function for drives to finish the request.

### Synopsis

```
void blk_end_request_all (struct request * rq, int error);
```

### Arguments

*rq*

the request to finish

*error*

0 for success, < 0 for error

### Description

Completely finish *rq*.

# blk\_end\_request\_cur

## LINUX

## Name

`blk_end_request_cur` — Helper function to finish the current request chunk.

## Synopsis

```
bool blk_end_request_cur (struct request * rq, int error);
```

## Arguments

*rq*

the request to finish the current chunk for

*error*

0 for success, < 0 for error

## Description

Complete the current consecutively mapped chunk from *rq*.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# blk\_end\_request\_err

**LINUX**

## Name

`blk_end_request_err` — Finish a request till the next failure boundary.

## Synopsis

```
bool blk_end_request_err (struct request * rq, int error);
```

## Arguments

*rq*

the request to finish till the next failure boundary for

*error*

must be negative errno

## Description

Complete *rq* till the next failure boundary.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

## \_\_blk\_end\_request

**LINUX**

## Name

`__blk_end_request` — Helper function for drivers to complete the request.

## Synopsis

```
bool __blk_end_request (struct request * rq, int error,  
unsigned int nr_bytes);
```

## Arguments

*rq*

the request being processed

*error*

0 for success, < 0 for error

*nr\_bytes*

number of bytes to complete

## Description

Must be called with queue lock held unlike `blk_end_request`.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# \_\_blk\_end\_request\_all

## LINUX

Kernel Hackers Manual June 2013

### Name

`__blk_end_request_all` — Helper function for drives to finish the request.

### Synopsis

```
void __blk_end_request_all (struct request * rq, int error);
```

### Arguments

*rq*

the request to finish

*error*

0 for success, < 0 for error

### Description

Completely finish *rq*. Must be called with queue lock held.

# \_\_blk\_end\_request\_cur

## LINUX



## Name

`__blk_end_request_cur` — Helper function to finish the current request chunk.

## Synopsis

```
bool __blk_end_request_cur (struct request * rq, int error);
```

## Arguments

*rq*

the request to finish the current chunk for

*error*

0 for success, < 0 for error

## Description

Complete the current consecutively mapped chunk from *rq*. Must be called with queue lock held.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# `__blk_end_request_err`

**LINUX**

## Name

`__blk_end_request_err` — Finish a request till the next failure boundary.

## Synopsis

```
bool __blk_end_request_err (struct request * rq, int error);
```

## Arguments

*rq*

the request to finish till the next failure boundary for

*error*

must be negative errno

## Description

Complete *rq* till the next failure boundary. Must be called with queue lock held.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# rq\_flush\_dcache\_pages

**LINUX**

## Name

`rq_flush_dcache_pages` — Helper function to flush all pages in a request

## Synopsis

```
void rq_flush_dcache_pages (struct request * rq);
```

## Arguments

*rq*

the request to be flushed

## Description

Flush all pages in *rq*.

# blk\_lld\_busy

## LINUX

## Name

`blk_lld_busy` — Check if underlying low-level drivers of a device are busy

## Synopsis

```
int blk_lld_busy (struct request_queue * q);
```

## Arguments

*q*

the queue of the device being checked

## Description

Check if underlying low-level drivers of a device are busy. If the drivers want to export their busy state, they must set own exporting function using `blk_queue_lld_busy` first.

Basically, this function is used only by request stacking drivers to stop dispatching requests to underlying devices when underlying devices are busy. This behavior helps more I/O merging on the queue of the request stacking driver and prevents I/O throughput regression on burst I/O load.

## Return

0 - Not busy (The request stacking driver should dispatch request) 1 - Busy (The request stacking driver should stop dispatching request)

## blk\_rq\_unprep\_clone

**LINUX**

## Name

`blk_rq_unprep_clone` — Helper function to free all bios in a cloned request

## Synopsis

```
void blk_rq_unprep_clone (struct request * rq);
```

## Arguments

*rq*

the clone request to be cleaned up

## Description

Free all bios in *rq* for a cloned request.

# blk\_rq\_prep\_clone

## LINUX

## Name

`blk_rq_prep_clone` — Helper function to setup clone request

## Synopsis

```
int blk_rq_prep_clone (struct request * rq, struct request *  
rq_src, struct bio_set * bs, gfp_t gfp_mask, int (*bio_ctr)  
(struct bio *, struct bio *, void *), void * data);
```

## Arguments

*rq*

the request to be setup

*rq\_src*

original request to be cloned

*bs*

bio\_set that bios for clone are allocated from

*gfp\_mask*

memory allocation mask for bio

*bio\_ctr*

setup function to be called for each clone bio. Returns 0 for success, non 0 for failure.

*data*

private data to be passed to *bio\_ctr*

## Description

Clones bios in *rq\_src* to *rq*, and copies attributes of *rq\_src* to *rq*. The actual data parts of *rq\_src* (e.g. ->cmd, ->buffer, ->sense) are not copied, and copying such parts is the caller's responsibility. Also, pages which the original bios are pointing to are not copied and the cloned bios just point same pages. So cloned bios must be completed before original bios, which means the caller must complete *rq* before *rq\_src*.

# \_\_blk\_run\_queue\_uncond

## LINUX

Kernel Hackers Manual June 2013

### Name

`__blk_run_queue_uncond` — run a queue whether or not it has been stopped

### Synopsis

```
void __blk_run_queue_uncond (struct request_queue * q);
```

### Arguments

*q*

The queue to run

### Description

Invoke request handling on a queue if there are any pending requests. May be used to restart request handling after a request has completed. This variant runs the queue whether or not the queue has been stopped. Must be called with the queue lock held and interrupts disabled. See also *blk\_run\_queue*.

# \_\_blk\_drain\_queue

## LINUX

## Name

`__blk_drain_queue` — drain requests from `request_queue`

## Synopsis

```
void __blk_drain_queue (struct request_queue * q, bool
                        drain_all);
```

## Arguments

*q*

queue to drain

*drain\_all*

whether to drain all requests or only the ones w/ ELVPRIV

## Description

Drain requests from *q*. If *drain\_all* is set, all requests are drained. If not, only ELVPRIV requests are drained. The caller is responsible for ensuring that no new requests which need to be drained are queued.

## get\_request

**LINUX**



## Name

`get_request` — get a free request

## Synopsis

```
struct request * get_request (struct request_queue * q, int  
rw_flags, struct bio * bio, gfp_t gfp_mask);
```

## Arguments

*q*

request\_queue to allocate request from

*rw\_flags*

RW and SYNC flags

*bio*

bio to allocate request for (can be NULL)

*gfp\_mask*

allocation mask

## Description

Get a free request from *q*. This function may fail under memory pressure or if *q* is dead.

Must be called with *q*->queue\_lock held and, Returns NULL on failure, with *q*->queue\_lock held. Returns !NULL on success, with *q*->queue\_lock \*not held\*.

# get\_request\_wait

## LINUX

Kernel Hackers Manual June 2013

### Name

`get_request_wait` — get a free request with retry

### Synopsis

```
struct request * get_request_wait (struct request_queue * q,  
int rw_flags, struct bio * bio);
```

### Arguments

*q*

request\_queue to allocate request from

*rw\_flags*

RW and SYNC flags

*bio*

bio to allocate request for (can be NULL)

### Description

Get a free request from *q*. This function keeps retrying under memory pressure and fails iff *q* is dead.

Must be called with *q*->queue\_lock held and, Returns NULL on failure, with *q*->queue\_lock held. Returns !NULL on success, with *q*->queue\_lock \*not held\*.

# attempt\_plug\_merge

## LINUX

Kernel Hackers Manual June 2013

### Name

`attempt_plug_merge` — try to merge with current's plugged list

### Synopsis

```
bool attempt_plug_merge (struct request_queue * q, struct bio
* bio, unsigned int * request_count);
```

### Arguments

*q*

request\_queue new bio is being queued at

*bio*

new bio being queued

*request\_count*

out parameter for number of traversed plugged requests

### Description

Determine whether *bio* being queued on *q* can be merged with a request on current's plugged list. Returns `true` if merge was successful, otherwise `false`.

Plugging coalesces IOs from the same issuer for the same purpose without going through *q->queue\_lock*. As such it's more of an issuing mechanism than scheduling, and the request, while may have `elvpriv` data, is not added on the elevator at this point. In addition, we don't have reliable access to the elevator

outside queue lock. Only check basic merging parameters without querying the elevator.

## **\_\_generic\_make\_request**

### **LINUX**

Kernel Hackers Manual June 2013

### **Name**

`__generic_make_request` — hand a buffer to its device driver for I/O

### **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`.

## blk\_end\_bidi\_request

### LINUX

Kernel Hackers Manual June 2013

### Name

`blk_end_bidi_request` — Complete a bidi request

### Synopsis

```
bool blk_end_bidi_request (struct request * rq, int error,
    unsigned int nr_bytes, unsigned int bidi_bytes);
```

### Arguments

*rq*

the request to complete

*error*

0 for success, < 0 for error

*nr\_bytes*

number of bytes to complete *rq*

*bidi\_bytes*

number of bytes to complete *rq->next\_rq*

## Description

Ends I/O on a number of bytes attached to *rq* and *rq->next\_rq*. Drivers that supports bidi can safely call this member for any type of request, bidi or uni. In the later case *bidi\_bytes* is just ignored.

## Return

*false* - we are done with this request *true* - still buffers pending for this request

# \_\_blk\_end\_bidi\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`__blk_end_bidi_request` — Complete a bidi request with queue lock held

## Synopsis

```
bool __blk_end_bidi_request (struct request * rq, int error,
unsigned int nr_bytes, unsigned int bidi_bytes);
```

## Arguments

*rq*

the request to complete

*error*

0 for success, < 0 for error

*nr\_bytes*

number of bytes to complete *rq*

*bidi\_bytes*

number of bytes to complete *rq->next\_rq*

## Description

Identical to `blk_end_bidi_request` except that queue lock is assumed to be locked on entry and remains so on return.

## Return

`false` - we are done with this request `true` - still buffers pending for this request

# blk\_rq\_map\_user

**LINUX**

Kernel Hackers Manual June 2013

## Name

`blk_rq_map_user` — map user data to a request, for `REQ_TYPE_BLOCK_PC` usage

## Synopsis

```
int blk_rq_map_user (struct request_queue * q, struct request
* rq, struct rq_map_data * map_data, void __user * ubuf,
unsigned long len, gfp_t gfp_mask);
```

## Arguments

*q*

request queue where request should be inserted

*rq*

request structure to fill

*map\_data*

pointer to the `rq_map_data` holding pages (if necessary)

*ubuf*

the user buffer

*len*

length of user data

*gfp\_mask*

memory allocation flags

## Description

Data will be mapped directly for zero copy I/O, if possible. Otherwise a kernel bounce buffer is used.

A matching `blk_rq_unmap_user` must be issued at the end of I/O, 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\_map\_user\_iov

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_rq_map_user_iov` — map user data to a request, for `REQ_TYPE_BLOCK_PC` usage

## Synopsis

```
int blk_rq_map_user_iov (struct request_queue * q, struct
request * rq, struct rq_map_data * map_data, struct sg_iovec *
iov, int iov_count, unsigned int len, gfp_t gfp_mask);
```

## Arguments

*q*

request queue where request should be inserted

*rq*

request to map data to

*map\_data*

pointer to the `rq_map_data` holding pages (if necessary)

*iov*

pointer to the iovec

*iov\_count*

number of elements in the iovec

*len*

I/O byte count

*gfp\_mask*

memory allocation flags

## Description

Data will be mapped directly for zero copy I/O, if possible. Otherwise a kernel bounce buffer is used.

A matching `blk_rq_unmap_user` must be issued at the end of I/O, 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_rq_unmap_user` — unmap a request with user data

## Synopsis

```
int blk_rq_unmap_user (struct bio * bio);
```

## Arguments

*bio*

start of bio list

## Description

Unmap a rq previously mapped by `blk_rq_map_user`. The caller must supply the original `rq->bio` from the `blk_rq_map_user` return, since the I/O completion may have changed `rq->bio`.

## blk\_rq\_map\_kern

### LINUX

Kernel Hackers Manual June 2013

## Name

`blk_rq_map_kern` — map kernel data to a request, for `REQ_TYPE_BLOCK_PC` usage

## Synopsis

```
int blk_rq_map_kern (struct request_queue * q, struct request
* rq, void * kbuf, unsigned int len, gfp_t gfp_mask);
```

## Arguments

*q*

request queue where request should be inserted

*rq*

request to fill

*kbuf*

the kernel buffer

*len*

length of user data

*gfp\_mask*

memory allocation flags

## Description

Data will be mapped directly if possible. Otherwise a bounce buffer is used. Can be called multiple times to append multiple buffers.

# blk\_release\_queue

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_release_queue` — release a struct `request_queue` when it is no longer needed

## Synopsis

```
void blk_release_queue (struct kobject * kobj);
```

## Arguments

*kobj*

the kobj belonging of 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\_queue\_prep\_rq

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_prep_rq` — set a `prepare_request` function for queue

## Synopsis

```
void blk_queue_prep_rq (struct request_queue * 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\_unprep\_rq

### LINUX

Kernel Hackers Manual June 2013

## Name

blk\_queue\_unprep\_rq — set an unprepare\_request function for queue

## Synopsis

```
void blk_queue_unprep_rq (struct request_queue * q,
unprep_rq_fn * ufn);
```

## Arguments

*q*

queue

*ufn*

unprepare\_request function

## Description

It's possible for a queue to register an unprepare\_request callback which is invoked before the request is finally completed. The goal of the function is to deallocate any data that was allocated in the prepare\_request callback.

# blk\_queue\_merge\_bvec

## LINUX

Kernel Hackers Manual June 2013

## Name

blk\_queue\_merge\_bvec — set a merge\_bvec function for queue

## Synopsis

```
void blk_queue_merge_bvec (struct request_queue * 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\_set\_default\_limits

### LINUX

Kernel Hackers Manual June 2013

## Name

blk\_set\_default\_limits — reset limits to default values



## Synopsis

```
void blk_set_default_limits (struct queue_limits * lim);
```

## Arguments

*lim*

the queue\_limits structure to reset

## Description

Returns a queue\_limit struct to its default state.

# blk\_set\_stacking\_limits

## LINUX

Kernel Hackers Manual June 2013

## Name

blk\_set\_stacking\_limits — set default limits for stacking devices

## Synopsis

```
void blk_set_stacking_limits (struct queue_limits * lim);
```

## Arguments

*lim*

the queue\_limits structure to reset

## Description

Returns a queue\_limit struct to its default state. Should be used by stacking drivers like DM that have no internal limits.

# blk\_queue\_make\_request

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_make_request` — define an alternate `make_request` function for a device

## Synopsis

```
void blk_queue_make_request (struct request_queue * 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\_bounce\_limit

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_bounce_limit` — set bounce buffer limit for queue

## Synopsis

```
void blk_queue_bounce_limit (struct request_queue * q, u64
dma_mask);
```

## Arguments

*q*

the request queue for the device

*dma\_mask*

the maximum address the device can handle

## 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 *dma\_mask*.

# blk\_limits\_max\_hw\_sectors

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_limits_max_hw_sectors` — set hard and soft limit of max sectors for request

## Synopsis

```
void blk_limits_max_hw_sectors (struct queue_limits * limits,  
unsigned int max_hw_sectors);
```

## Arguments

*limits*

the queue limits

*max\_hw\_sectors*

max hardware sectors in the usual 512b unit

## Description

Enables a low level driver to set a hard upper limit, `max_hw_sectors`, on the size of requests. `max_hw_sectors` is set by the device driver based upon the combined capabilities of I/O controller and storage device.

`max_sectors` is a soft limit imposed by the block layer for filesystem type requests. This value can be overridden on a per-device basis in `/sys/block/<device>/queue/max_sectors_kb`. The soft limit can not exceed `max_hw_sectors`.

## blk\_queue\_max\_hw\_sectors

### LINUX

Kernel Hackers Manual June 2013

### Name

`blk_queue_max_hw_sectors` — set max sectors for a request for this queue

### Synopsis

```
void blk_queue_max_hw_sectors (struct request_queue * q,
unsigned int max_hw_sectors);
```

## Arguments

*q*

the request queue for the device

*max\_hw\_sectors*

max hardware sectors in the usual 512b unit

## Description

See description for `blk_limits_max_hw_sectors`.

# blk\_queue\_max\_discard\_sectors

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_max_discard_sectors` — set max sectors for a single discard

## Synopsis

```
void blk_queue_max_discard_sectors (struct request_queue * q,  
unsigned int max_discard_sectors);
```

## Arguments

*q*

the request queue for the device

*max\_discard\_sectors*

maximum number of sectors to discard

## blk\_queue\_max\_segments

### LINUX

Kernel Hackers Manual June 2013

### Name

`blk_queue_max_segments` — set max hw segments for a request for this queue

### Synopsis

```
void blk_queue_max_segments (struct request_queue * 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.

# blk\_queue\_max\_segment\_size

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_queue_max_segment_size` — set max segment size for `blk_rq_map_sg`

### Synopsis

```
void blk_queue_max_segment_size (struct request_queue * 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\_logical\_block\_size

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_queue_logical_block_size` — set logical block size for the queue

### Synopsis

```
void blk_queue_logical_block_size (struct request_queue * q,  
unsigned short size);
```

### Arguments

*q*

the request queue for the device

*size*

the logical block size, in bytes

### Description

This should be set to the lowest possible block size that the storage device can address. The default of 512 covers most hardware.

# blk\_queue\_physical\_block\_size

## LINUX

## Name

`blk_queue_physical_block_size` — set physical block size for the queue

## Synopsis

```
void blk_queue_physical_block_size (struct request_queue * q,  
unsigned int size);
```

## Arguments

*q*

the request queue for the device

*size*

the physical block size, in bytes

## Description

This should be set to the lowest possible sector size that the hardware can operate on without reverting to read-modify-write operations.

# blk\_queue\_alignment\_offset

**LINUX**

## Name

`blk_queue_alignment_offset` — set physical block alignment offset

## Synopsis

```
void blk_queue_alignment_offset (struct request_queue * q,  
unsigned int offset);
```

## Arguments

*q*

the request queue for the device

*offset*

alignment offset in bytes

## Description

Some devices are naturally misaligned to compensate for things like the legacy DOS partition table 63-sector offset. Low-level drivers should call this function for devices whose first sector is not naturally aligned.

## `blk_limits_io_min`

**LINUX**

## Name

`blk_limits_io_min` — set minimum request size for a device

## Synopsis

```
void blk_limits_io_min (struct queue_limits * limits, unsigned  
int min);
```

## Arguments

*limits*

the queue limits

*min*

smallest I/O size in bytes

## Description

Some devices have an internal block size bigger than the reported hardware sector size. This function can be used to signal the smallest I/O the device can perform without incurring a performance penalty.

## `blk_queue_io_min`

**LINUX**

## Name

`blk_queue_io_min` — set minimum request size for the queue

## Synopsis

```
void blk_queue_io_min (struct request_queue * q, unsigned int min);
```

## Arguments

*q*

the request queue for the device

*min*

smallest I/O size in bytes

## Description

Storage devices may report a granularity or preferred minimum I/O size which is the smallest request the device can perform without incurring a performance penalty. For disk drives this is often the physical block size. For RAID arrays it is often the stripe chunk size. A properly aligned multiple of `minimum_io_size` is the preferred request size for workloads where a high number of I/O operations is desired.

## `blk_limits_io_opt`

**LINUX**

## Name

`blk_limits_io_opt` — set optimal request size for a device

## Synopsis

```
void blk_limits_io_opt (struct queue_limits * limits, unsigned  
int opt);
```

## Arguments

*limits*

the queue limits

*opt*

smallest I/O size in bytes

## Description

Storage devices may report an optimal I/O size, which is the device's preferred unit for sustained I/O. This is rarely reported for disk drives. For RAID arrays it is usually the stripe width or the internal track size. A properly aligned multiple of `optimal_io_size` is the preferred request size for workloads where sustained throughput is desired.

## `blk_queue_io_opt`

**LINUX**

## Name

`blk_queue_io_opt` — set optimal request size for the queue

## Synopsis

```
void blk_queue_io_opt (struct request_queue * q, unsigned int opt);
```

## Arguments

*q*

the request queue for the device

*opt*

optimal request size in bytes

## Description

Storage devices may report an optimal I/O size, which is the device's preferred unit for sustained I/O. This is rarely reported for disk drives. For RAID arrays it is usually the stripe width or the internal track size. A properly aligned multiple of `optimal_io_size` is the preferred request size for workloads where sustained throughput is desired.

## `blk_queue_stack_limits`

**LINUX**

## Name

`blk_queue_stack_limits` — inherit underlying queue limits for stacked drivers

## Synopsis

```
void blk_queue_stack_limits (struct request_queue * t, struct  
request_queue * b);
```

## Arguments

*t*  
the stacking driver (top)

*b*  
the underlying device (bottom)

# blk\_stack\_limits

## LINUX

## Name

`blk_stack_limits` — adjust queue\_limits for stacked devices



## Synopsis

```
int blk_stack_limits (struct queue_limits * t, struct
queue_limits * b, sector_t start);
```

## Arguments

*t*

the stacking driver limits (top device)

*b*

the underlying queue limits (bottom, component device)

*start*

first data sector within component device

## Description

This function is used by stacking drivers like MD and DM to ensure that all component devices have compatible block sizes and alignments. The stacking driver must provide a `queue_limits` struct (top) and then iteratively call the stacking function for all component (bottom) devices. The stacking function will attempt to combine the values and ensure proper alignment.

Returns 0 if the top and bottom `queue_limits` are compatible. The top device's block sizes and alignment offsets may be adjusted to ensure alignment with the bottom device. If no compatible sizes and alignments exist, -1 is returned and the resulting top `queue_limits` will have the `misaligned` flag set to indicate that the `alignment_offset` is undefined.

## `bdev_stack_limits`

**LINUX**

## Name

`bdev_stack_limits` — adjust queue limits for stacked drivers

## Synopsis

```
int bdev_stack_limits (struct queue_limits * t, struct
block_device * bdev, sector_t start);
```

## Arguments

*t*

the stacking driver limits (top device)

*bdev*

the component `block_device` (bottom)

*start*

first data sector within component device

## Description

Merges queue limits for a top device and a `block_device`. Returns 0 if alignment didn't change. Returns -1 if adding the bottom device caused misalignment.

## `disk_stack_limits`

**LINUX**

## Name

`disk_stack_limits` — adjust queue limits for stacked drivers

## Synopsis

```
void disk_stack_limits (struct gendisk * disk, struct  
block_device * bdev, sector_t offset);
```

## Arguments

*disk*

MD/DM gendisk (top)

*bdev*

the underlying block device (bottom)

*offset*

offset to beginning of data within component device

## Description

Merges the limits for a top level gendisk and a bottom level block\_device.

# blk\_queue\_dma\_pad

**LINUX**

## Name

`blk_queue_dma_pad` — set pad mask

## Synopsis

```
void blk_queue_dma_pad (struct request_queue * q, unsigned int  
mask);
```

## Arguments

*q*

the request queue for the device

*mask*

pad mask

## Description

Set dma pad mask.

Appending pad buffer to a request modifies the last entry of a scatter list such that it includes the pad buffer.

# `blk_queue_update_dma_pad`

**LINUX**

## Name

`blk_queue_update_dma_pad` — update pad mask

## Synopsis

```
void blk_queue_update_dma_pad (struct request_queue * q,  
unsigned int mask);
```

## Arguments

*q*

the request queue for the device

*mask*

pad mask

## Description

Update dma pad mask.

Appending pad buffer to a request modifies the last entry of a scatter list such that it includes the pad buffer.

# blk\_queue\_dma\_drain

**LINUX**

## Name

`blk_queue_dma_drain` — Set up a drain buffer for excess dma.

## Synopsis

```
int blk_queue_dma_drain (struct request_queue * q,  
dma_drain_needed_fn * dma_drain_needed, void * buf, unsigned  
int size);
```

## Arguments

*q*

the request queue for the device

*dma\_drain\_needed*

fn which returns non-zero if drain is necessary

*buf*

physically contiguous buffer

*size*

size of the buffer in bytes

## Description

Some devices have excess DMA problems and can't simply discard (or zero fill) the unwanted piece of the transfer. They have to have a real area of memory to transfer it into. The use case for this is ATAPI devices in DMA mode. If the packet command causes a transfer bigger than the transfer size some HBAs will lock up if there aren't DMA elements to contain the excess transfer. What this API does is adjust the queue so that the *buf* is always appended silently to the scatterlist.

## Note

This routine adjusts `max_hw_segments` to make room for appending the drain buffer. If you call `blk_queue_max_segments` after calling this routine, you must set the limit to one fewer than your device can support otherwise there won't be room for the drain buffer.

# blk\_queue\_segment\_boundary

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_segment_boundary` — set boundary rules for segment merging

## Synopsis

```
void blk_queue_segment_boundary (struct request_queue * q,
unsigned long mask);
```

## Arguments

*q*  
the request queue for the device

*mask*  
the memory boundary mask

# blk\_queue\_dma\_alignment

## LINUX

Kernel Hackers Manual June 2013

### Name

`blk_queue_dma_alignment` — set dma length and memory alignment

### Synopsis

```
void blk_queue_dma_alignment (struct request_queue * 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\_update\_dma\_alignment

## LINUX



## Name

`blk_queue_update_dma_alignment` — update dma length and memory alignment

## Synopsis

```
void blk_queue_update_dma_alignment (struct request_queue * q,  
int mask);
```

## Arguments

*q*

the request queue for the device

*mask*

alignment mask

## description

update required memory and length alignment for direct dma transactions. If the requested alignment is larger than the current alignment, then the current queue alignment is updated to the new value, otherwise it is left alone. The design of this is to allow multiple objects (driver, device, transport etc) to set their respective alignments without having them interfere.

## `blk_queue_flush`

**LINUX**

## Name

`blk_queue_flush` — configure queue’s cache flush capability

## Synopsis

```
void blk_queue_flush (struct request_queue * q, unsigned int  
flush);
```

## Arguments

*q*

the request queue for the device

*flush*

0, REQ\_FLUSH or REQ\_FLUSH | REQ\_FUA

## Description

Tell block layer cache flush capability of *q*. If it supports flushing, REQ\_FLUSH should be set. If it supports bypassing write cache for individual writes, REQ\_FUA should be set.

# blk\_execute\_rq\_nowait

**LINUX**

## Name

`blk_execute_rq_nowait` — insert a request into queue for execution

## Synopsis

```
void blk_execute_rq_nowait (struct request_queue * q, struct  
gendisk * bd_disk, struct request * rq, int at_head,  
rq_end_io_fn * done);
```

## Arguments

*q*

queue to insert the request in

*bd\_disk*

matching gendisk

*rq*

request to insert

*at\_head*

insert request at head or tail of queue

*done*

I/O completion handler

## Description

Insert a fully prepared request at the back of the I/O scheduler queue for execution. Don't wait for completion.

## Note

This function will invoke *done* directly if the queue is dead.

# blk\_execute\_rq

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_execute_rq` — insert a request into queue for execution

## Synopsis

```
int blk_execute_rq (struct request_queue * q, struct gendisk *  
bd_disk, struct request * rq, int at_head);
```

## Arguments

*q*

queue to insert the request in

*bd\_disk*

matching gendisk

*rq*

request to insert

*at\_head*

insert request at head or tail of queue

## Description

Insert a fully prepared request at the back of the I/O scheduler queue for execution and wait for completion.

# blkdev\_issue\_flush

## LINUX

Kernel Hackers Manual June 2013

## Name

`blkdev_issue_flush` — queue a flush

## Synopsis

```
int blkdev_issue_flush (struct block_device * bdev, gfp_t  
gfp_mask, sector_t * error_sector);
```

## Arguments

*bdev*

blockdev to issue flush for

*gfp\_mask*

memory allocation flags (for `bio_alloc`)

*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. If WAIT flag is not passed then caller may check only what request was pushed in some internal queue for later handling.

# blkdev\_issue\_discard

## LINUX

Kernel Hackers Manual June 2013

## Name

`blkdev_issue_discard` — queue a discard

## Synopsis

```
int blkdev_issue_discard (struct block_device * bdev, sector_t
sector, sector_t nr_sects, gfp_t gfp_mask, unsigned long
flags);
```

## Arguments

*bdev*

blockdev to issue discard for

*sector*

start sector

*nr\_sects*

number of sectors to discard

*gfp\_mask*memory allocation flags (for `bio_alloc`)*flags*

BLKDEV\_IFL\_\* flags to control behaviour

## Description

Issue a discard request for the sectors in question.

# blkdev\_issue\_zeroout

## LINUX

Kernel Hackers Manual June 2013

## Name

`blkdev_issue_zeroout` — generate number of zero filled write bios

## Synopsis

```
int blkdev_issue_zeroout (struct block_device * bdev, sector_t
sector, sector_t nr_sects, gfp_t gfp_mask);
```

## Arguments

*bdev*

blockdev to issue

*sector*

start sector

*nr\_sects*

number of sectors to write

*gfp\_mask*

memory allocation flags (for bio\_alloc)

## Description

Generate and issue number of bios with zeroed pages.

# blk\_queue\_find\_tag

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_find_tag` — find a request by its tag and queue

## Synopsis

```
struct request * blk_queue_find_tag (struct request_queue * q,  
int tag);
```

## Arguments

*q*

The request queue for the device

*tag*

The tag of the request



## 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\_free\_tags

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_free_tags` — release a given set of tag maintenance info

## Synopsis

```
void blk_free_tags (struct blk_queue_tag * bqt);
```

## Arguments

*bqt*

the tag map to free

## Description

For externally managed *bqt* frees the map. Callers of this function must guarantee to have released all the queues that might have been using this tag map.

# blk\_queue\_free\_tags

## LINUX

Kernel Hackers Manual June 2013

### Name

blk\_queue\_free\_tags — release tag maintenance info

### Synopsis

```
void blk_queue_free_tags (struct request_queue * q);
```

### Arguments

*q*

the request queue for the device

### Notes

This is used to disable tagged queuing to a device, yet leave queue in function.

# blk\_init\_tags

## LINUX

Kernel Hackers Manual June 2013

### Name

blk\_init\_tags — initialize the tag info for an external tag map

## Synopsis

```
struct blk_queue_tag * blk_init_tags (int depth);
```

## Arguments

*depth*

the maximum queue depth supported

# blk\_queue\_init\_tags

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_init_tags` — initialize the queue tag info

## Synopsis

```
int blk_queue_init_tags (struct request_queue * q, int depth,  
struct blk_queue_tag * tags);
```

## Arguments

*q*

the request queue for the device

*depth*

the maximum queue depth supported

*tags*

the tag to use

## Description

Queue lock must be held here if the function is called to resize an existing map.

# blk\_queue\_resize\_tags

**LINUX**

Kernel Hackers Manual June 2013

## Name

`blk_queue_resize_tags` — change the queueing depth

## Synopsis

```
int blk_queue_resize_tags (struct request_queue * 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_end_tag` — end tag operations for a request

## Synopsis

```
void blk_queue_end_tag (struct request_queue * 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

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_start_tag` — find a free tag and assign it

## Synopsis

```
int blk_queue_start_tag (struct request_queue * 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 readd it if it should need to be restarted for some reason.

## Notes

queue lock must be held.

# blk\_queue\_invalidate\_tags

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_queue_invalidate_tags` — invalidate all pending tags

## Synopsis

```
void blk_queue_invalidate_tags (struct request_queue * 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.

# \_\_blk\_free\_tags

## LINUX

Kernel Hackers Manual June 2013

## Name

`__blk_free_tags` — release a given set of tag maintenance info

## Synopsis

```
int __blk_free_tags (struct blk_queue_tag * bqt);
```

## Arguments

*bqt*

the tag map to free

## Description

Tries to free the specified *bqt*. Returns true if it was actually freed and false if there are still references using it



# \_\_blk\_queue\_free\_tags

## LINUX

Kernel Hackers Manual June 2013

### Name

`__blk_queue_free_tags` — release tag maintenance info

### Synopsis

```
void __blk_queue_free_tags (struct request_queue * q);
```

### Arguments

*q*

the request queue for the device

### Notes

`blk_cleanup_queue` will take care of calling this function, if tagging has been used. So there's no need to call this directly.

# blk\_rq\_count\_integrity\_sg

## LINUX

## Name

`blk_rq_count_integrity_sg` — Count number of integrity scatterlist elements

## Synopsis

```
int blk_rq_count_integrity_sg (struct request_queue * q,  
struct bio * bio);
```

## Arguments

*q*

request queue

*bio*

bio with integrity metadata attached

## Description

Returns the number of elements required in a scatterlist corresponding to the integrity metadata in a bio.

## `blk_rq_map_integrity_sg`

**LINUX**

## Name

`blk_rq_map_integrity_sg` — Map integrity metadata into a scatterlist

## Synopsis

```
int blk_rq_map_integrity_sg (struct request_queue * q, struct
bio * bio, struct scatterlist * sglist);
```

## Arguments

*q*

request queue

*bio*

bio with integrity metadata attached

*sglist*

target scatterlist

## Description

Map the integrity vectors in request into a scatterlist. The scatterlist must be big enough to hold all elements. I.e. sized using `blk_rq_count_integrity_sg`.

## `blk_integrity_compare`

**LINUX**

## Name

`blk_integrity_compare` — Compare integrity profile of two disks

## Synopsis

```
int blk_integrity_compare (struct gendisk * gd1, struct
gendisk * gd2);
```

## Arguments

*gd1*

Disk to compare

*gd2*

Disk to compare

## Description

Meta-devices like DM and MD need to verify that all sub-devices use the same integrity format before advertising to upper layers that they can send/receive integrity metadata. This function can be used to check whether two gendisk devices have compatible integrity formats.

## `blk_integrity_register`

**LINUX**

## Name

`blk_integrity_register` — Register a gendisk as being integrity-capable

## Synopsis

```
int blk_integrity_register (struct gendisk * disk, struct  
blk_integrity * template);
```

## Arguments

*disk*

struct gendisk pointer to make integrity-aware

*template*

optional integrity profile to register

## Description

When a device needs to advertise itself as being able to send/receive integrity metadata it must use this function to register the capability with the block layer. The *template* is a `blk_integrity` struct with values appropriate for the underlying hardware. If *template* is `NULL` the new profile is allocated but not filled out. See [Documentation/block/data-integrity.txt](#).

## `blk_integrity_unregister`

**LINUX**

## Name

`blk_integrity_unregister` — Remove block integrity profile

## Synopsis

```
void blk_integrity_unregister (struct gendisk * disk);
```

## Arguments

*disk*

disk whose integrity profile to deallocate

## Description

This function frees all memory used by the block integrity profile. To be called at device teardown.

# blk\_trace\_ioctl

## LINUX

## Name

`blk_trace_ioctl` — handle the ioctls associated with tracing

## Synopsis

```
int blk_trace_ioctl (struct block_device * bdev, unsigned cmd,
char __user * arg);
```

## Arguments

*bdev*

the block device

*cmd*

the ioctl cmd

*arg*

the argument data, if any

## blk\_trace\_shutdown

### LINUX

Kernel Hackers Manual June 2013

## Name

blk\_trace\_shutdown — stop and cleanup trace structures

## Synopsis

```
void blk_trace_shutdown (struct request_queue * q);
```

## Arguments

*q*

the request queue associated with the device

## blk\_add\_trace\_rq

### LINUX

Kernel Hackers Manual June 2013

## Name

`blk_add_trace_rq` — Add a trace for a request oriented action

## Synopsis

```
void blk_add_trace_rq (struct request_queue * q, struct  
request * rq, u32 what);
```

## Arguments

*q*

queue the io is for

*rq*

the source request

*what*

the action



## Description

Records an action against a request. Will log the bio offset + size.

# blk\_add\_trace\_bio

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_add_trace_bio` — Add a trace for a bio oriented action

## Synopsis

```
void blk_add_trace_bio (struct request_queue * q, struct bio *  
bio, u32 what, int error);
```

## Arguments

*q*

queue the io is for

*bio*

the source bio

*what*

the action

*error*

error, if any

## Description

Records an action against a bio. Will log the bio offset + size.

# blk\_add\_trace\_bio\_remap

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_add_trace_bio_remap` — Add a trace for a bio-remap operation

## Synopsis

```
void blk_add_trace_bio_remap (void * ignore, struct  
request_queue * q, struct bio * bio, dev_t dev, sector_t  
from);
```

## Arguments

*ignore*

trace callback data parameter (not used)

*q*

queue the io is for

*bio*

the source bio

*dev*

target device

*from*

source sector

## Description

Device mapper or raid target sometimes need to split a bio because it spans a stripe (or similar). Add a trace for that action.

# blk\_add\_trace\_rq\_remap

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_add_trace_rq_remap` — Add a trace for a request-remap operation

## Synopsis

```
void blk_add_trace_rq_remap (void * ignore, struct
request_queue * q, struct request * rq, dev_t dev, sector_t
from);
```

## Arguments

*ignore*

trace callback data parameter (not used)

*q*

queue the io is for

*rq*

the source request

*dev*

target device

*from*

source sector

## Description

Device mapper remaps request to other devices. Add a trace for that action.

# blk\_mangle\_minor

**LINUX**

Kernel Hackers Manual June 2013

## Name

`blk_mangle_minor` — scatter minor numbers apart

## Synopsis

```
int blk_mangle_minor (int minor);
```

## Arguments

*minor*

minor number to mangle

## Description

Scatter consecutively allocated *minor* number apart if MANGLE\_DEVT is enabled. Mangling twice gives the original value.

## RETURNS

Mangled value.

## CONTEXT

Don't care.

# blk\_alloc\_devt

## LINUX

Kernel Hackers Manual June 2013

## Name

`blk_alloc_devt` — allocate a `dev_t` for a partition

## Synopsis

```
int blk_alloc_devt (struct hd_struct * part, dev_t * devt);
```

## Arguments

*part*

partition to allocate `dev_t` for

*dev\_t*

out parameter for resulting dev\_t

## Description

Allocate a dev\_t for block device.

## RETURNS

0 on success, allocated dev\_t is returned in *\*dev\_t*. -errno on failure.

## CONTEXT

Might sleep.

## blk\_free\_devt

### LINUX

Kernel Hackers Manual June 2013

## Name

blk\_free\_devt — free a dev\_t

## Synopsis

```
void blk_free_devt (dev_t devt);
```

## Arguments

*devt*

dev\_t to free

## Description

Free *devt* which was allocated using `blk_alloc_devt`.

## CONTEXT

Might sleep.

# disk\_replace\_part\_tbl

## LINUX

Kernel Hackers Manual June 2013

## Name

`disk_replace_part_tbl` — replace `disk->part_tbl` in RCU-safe way

## Synopsis

```
void disk_replace_part_tbl (struct gendisk * disk, struct  
disk_part_tbl * new_ptbl);
```

## Arguments

*disk*

disk to replace `part_tbl` for

*new\_ptbl*

new `part_tbl` to install

## Description

Replace `disk->part_tbl` with *new\_ptbl* in RCU-safe way. The original `ptbl` is freed using RCU callback.

## LOCKING

Matching `bd_mutex` locked.

# disk\_expand\_part\_tbl

## LINUX

Kernel Hackers Manual June 2013

## Name

`disk_expand_part_tbl` — expand `disk->part_tbl`

## Synopsis

```
int disk_expand_part_tbl (struct gendisk * disk, int partno);
```



## Arguments

*disk*

disk to expand part\_tbl for

*partno*

expand such that this partno can fit in

## Description

Expand disk->part\_tbl such that *partno* can fit in. disk->part\_tbl uses RCU to allow unlocked dereferencing for stats and other stuff.

## LOCKING

Matching bd\_mutex locked, might sleep.

## RETURNS

0 on success, -errno on failure.

# disk\_block\_events

## LINUX

Kernel Hackers Manual June 2013

## Name

disk\_block\_events — block and flush disk event checking

## Synopsis

```
void disk_block_events (struct gendisk * disk);
```

## Arguments

*disk*

disk to block events for

## Description

On return from this function, it is guaranteed that event checking isn't in progress and won't happen until unblocked by `disk_unblock_events`. Events blocking is counted and the actual unblocking happens after the matching number of unblocks are done.

Note that this intentionally does not block event checking from `disk_clear_events`.

## CONTEXT

Might sleep.

# disk\_unblock\_events

## LINUX

Kernel Hackers Manual June 2013

## Name

`disk_unblock_events` — unblock disk event checking

## Synopsis

```
void disk_unblock_events (struct gendisk * disk);
```

## Arguments

*disk*

disk to unblock events for

## Description

Undo `disk_block_events`. When the block count reaches zero, it starts events polling if configured.

## CONTEXT

Don't care. Safe to call from irq context.

# disk\_clear\_events

## LINUX

Kernel Hackers Manual June 2013

## Name

`disk_clear_events` — synchronously check, clear and return pending events

## Synopsis

```
unsigned int disk_clear_events (struct gendisk * disk,  
unsigned int mask);
```

## Arguments

*disk*

disk to fetch and clear events from

*mask*

mask of events to be fetched and cleared

## Description

Disk events are synchronously checked and pending events in *mask* are cleared and returned. This ignores the block count.

## CONTEXT

Might sleep.

## disk\_get\_part

### LINUX

Kernel Hackers Manual June 2013

## Name

`disk_get_part` — get partition

## Synopsis

```
struct hd_struct * disk_get_part (struct gendisk * disk, int  
partno);
```

## Arguments

*disk*

disk to look partition from

*partno*

partition number

## Description

Look for partition *partno* from *disk*. If found, increment reference count and return it.

## CONTEXT

Don't care.

## RETURNS

Pointer to the found partition on success, NULL if not found.

# disk\_part\_iter\_init

**LINUX**

## Name

`disk_part_iter_init` — initialize partition iterator

## Synopsis

```
void disk_part_iter_init (struct disk_part_iter * piter,  
struct gendisk * disk, unsigned int flags);
```

## Arguments

*piter*

iterator to initialize

*disk*

disk to iterate over

*flags*

DISK\_PITER\_\* flags

## Description

Initialize *piter* so that it iterates over partitions of *disk*.

## CONTEXT

Don't care.

# disk\_part\_iter\_next

## LINUX

Kernel Hackers Manual June 2013

### Name

`disk_part_iter_next` — proceed iterator to the next partition and return it

### Synopsis

```
struct hd_struct * disk_part_iter_next (struct disk_part_iter  
* piter);
```

### Arguments

*piter*

iterator of interest

### Description

Proceed *piter* to the next partition and return it.

### CONTEXT

Don't care.

# disk\_part\_iter\_exit

## LINUX

## Name

`disk_part_iter_exit` — finish up partition iteration

## Synopsis

```
void disk_part_iter_exit (struct disk_part_iter * piter);
```

## Arguments

*piter*

iter of interest

## Description

Called when iteration is over. Cleans up *piter*.

## CONTEXT

Don't care.

# disk\_map\_sector\_rcu

**LINUX**



## Name

`disk_map_sector_rcu` — map sector to partition

## Synopsis

```
struct hd_struct * disk_map_sector_rcu (struct gendisk * disk,  
sector_t sector);
```

## Arguments

*disk*

gendisk of interest

*sector*

sector to map

## Description

Find out which partition *sector* maps to on *disk*. This is primarily used for stats accounting.

## CONTEXT

RCU read locked. The returned partition pointer is valid only while preemption is disabled.

## RETURNS

Found partition on success, `part0` is returned if no partition matches

# register\_blkdev

## LINUX

Kernel Hackers Manual June 2013

### Name

`register_blkdev` — register a new block device

### Synopsis

```
int register_blkdev (unsigned int major, const char * name);
```

### Arguments

*major*

the requested major device number [1..255]. If *major*=0, try to allocate any unused major number.

*name*

the name of the new block device as a zero terminated string

### Description

The *name* must be unique within the system.

The return value depends on the *major* input parameter. - if a major device number was requested in range [1..255] then the function returns zero on success, or a negative error code - if any unused major number was requested with *major*=0 parameter then the return value is the allocated major number in range [1..255] or a negative error code otherwise

# add\_disk

## LINUX

Kernel Hackers Manual June 2013

### Name

`add_disk` — add partitioning information to kernel list

### Synopsis

```
void add_disk (struct gendisk * disk);
```

### Arguments

*disk*

per-device partitioning information

### Description

This function registers the partitioning information in *disk* with the kernel.

### FIXME

error handling

# get\_gendisk

## LINUX

## Name

`get_gendisk` — get partitioning information for a given device

## Synopsis

```
struct gendisk * get_gendisk (dev_t devt, int * partno);
```

## Arguments

*devt*

device to get partitioning information for

*partno*

returned partition index

## Description

This function gets the structure containing partitioning information for the given device *devt*.

# bdget\_disk

## LINUX

## Name

`bdget_disk` — do bdget by gendisk and partition number

## Synopsis

```
struct block_device * bdget_disk (struct gendisk * disk, int  
partno);
```

## Arguments

*disk*

gendisk of interest

*partno*

partition number

## Description

Find partition *partno* from *disk*, do `bdget` on it.

## CONTEXT

Don't care.

## RETURNS

Resulting `block_device` on success, `NULL` on failure.

# disk\_check\_events

**LINUX**

## Name

`disk_check_events` — schedule immediate event checking

## Synopsis

```
void disk_check_events (struct gendisk * disk);
```

## Arguments

*disk*

disk to check events for

## Description

Schedule immediate event checking on *disk* if not blocked.

## CONTEXT

Don't care. Safe to call from irq context.

# Chapter 15. Char devices

## register\_chrdev\_region

### LINUX

Kernel Hackers Manual June 2013

### Name

`register_chrdev_region` — register a range of device numbers

### Synopsis

```
int register_chrdev_region (dev_t from, unsigned count, const
char * name);
```

### Arguments

*from*

the first in the desired range of device numbers; must include the major number.

*count*

the number of consecutive device numbers required

*name*

the name of the device or driver.

### Description

Return value is zero on success, a negative error code on failure.

# alloc\_chrdev\_region

## LINUX

Kernel Hackers Manual June 2013

### Name

`alloc_chrdev_region` — register a range of char device numbers

### Synopsis

```
int alloc_chrdev_region (dev_t * dev, unsigned baseminor,  
unsigned count, const char * name);
```

### Arguments

*dev*

output parameter for first assigned number

*baseminor*

first of the requested range of minor numbers

*count*

the number of minor numbers required

*name*

the name of the associated device or driver



## Description

Allocates a range of char device numbers. The major number will be chosen dynamically, and returned (along with the first minor number) in *dev*. Returns zero or a negative error code.

## \_\_register\_chrdev

### LINUX

Kernel Hackers Manual June 2013

## Name

`__register_chrdev` — create and register a cdev occupying a range of minors

## Synopsis

```
int __register_chrdev (unsigned int major, unsigned int
baseminor, unsigned int count, const char * name, const struct
file_operations * fops);
```

## Arguments

*major*

major device number or 0 for dynamic allocation

*baseminor*

first of the requested range of minor numbers

*count*

the number of minor numbers required

*name*

name of this range of devices

*fops*

file operations associated with this devices

## Description

If *major* == 0 this functions will dynamically allocate a major and return its number.

If *major* > 0 this function will attempt to reserve a device with the given major number and will return zero on success.

Returns a -ve errno on failure.

The name of this device has nothing to do with the name of the device in /dev. It only helps to keep track of the different owners of devices. If your module name has only one type of devices it's ok to use e.g. the name of the module here.

## unregister\_chrdev\_region

### LINUX

Kernel Hackers Manual June 2013

### Name

`unregister_chrdev_region` — return a range of device numbers

### Synopsis

```
void unregister_chrdev_region (dev_t from, unsigned count);
```

## Arguments

*from*

the first in the range of numbers to unregister

*count*

the number of device numbers to unregister

## Description

This function will unregister a range of *count* device numbers, starting with *from*. The caller should normally be the one who allocated those numbers in the first place...

# \_\_unregister\_chrdev

## LINUX

Kernel Hackers Manual June 2013

## Name

`__unregister_chrdev` — unregister and destroy a cdev

## Synopsis

```
void __unregister_chrdev (unsigned int major, unsigned int
baseminor, unsigned int count, const char * name);
```

## Arguments

*major*

major device number

*baseminor*

first of the range of minor numbers

*count*

the number of minor numbers this cdev is occupying

*name*

name of this range of devices

## Description

Unregister and destroy the cdev occupying the region described by *major*, *baseminor* and *count*. This function undoes what `__register_chrdev` did.

## cdev\_add

### LINUX

Kernel Hackers Manual June 2013

## Name

`cdev_add` — add a char device to the system

## Synopsis

```
int cdev_add (struct cdev * p, dev_t dev, unsigned count);
```

## Arguments

*p*

the cdev structure for the device

*dev*

the first device number for which this device is responsible

*count*

the number of consecutive minor numbers corresponding to this device

## Description

`cdev_add` adds the device represented by *p* to the system, making it live immediately. A negative error code is returned on failure.

## cdev\_del

### LINUX

Kernel Hackers Manual June 2013

## Name

`cdev_del` — remove a cdev from the system

## Synopsis

```
void cdev_del (struct cdev * p);
```

## Arguments

*p*

the cdev structure to be removed

## Description

`cdev_del` removes *p* from the system, possibly freeing the structure itself.

## cdev\_alloc

### LINUX

Kernel Hackers Manual June 2013

## Name

`cdev_alloc` — allocate a cdev structure

## Synopsis

```
struct cdev * cdev_alloc ( void );
```

## Arguments

*void*

no arguments

## Description

Allocates and returns a `cdev` structure, or `NULL` on failure.

## **cdev\_init**

### **LINUX**

Kernel Hackers Manual June 2013

### **Name**

`cdev_init` — initialize a `cdev` structure

### **Synopsis**

```
void cdev_init (struct cdev * cdev, const struct  
file_operations * fops);
```

### **Arguments**

*cdev*

the structure to initialize

*fops*

the `file_operations` for this device

### **Description**

Initializes *cdev*, remembering *fops*, making it ready to add to the system with `cdev_add`.





# Chapter 16. Miscellaneous Devices

## misc\_register

### LINUX

Kernel Hackers Manual June 2013

### 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

## LINUX

Kernel Hackers Manual June 2013

### 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. Clock Framework

The clock framework defines programming interfaces to support software management of the system clock tree. This framework is widely used with System-On-Chip (SOC) platforms to support power management and various devices which may need custom clock rates. Note that these "clocks" don't relate to timekeeping or real time clocks (RTCs), each of which have separate frameworks. These struct `clk` instances may be used to manage for example a 96 MHz signal that is used to shift bits into and out of peripherals or busses, or otherwise trigger synchronous state machine transitions in system hardware.

Power management is supported by explicit software clock gating: unused clocks are disabled, so the system doesn't waste power changing the state of transistors that aren't in active use. On some systems this may be backed by hardware clock gating, where clocks are gated without being disabled in software. Sections of chips that are powered but not clocked may be able to retain their last state. This low power state is often called a *retention mode*. This mode still incurs leakage currents, especially with finer circuit geometries, but for CMOS circuits power is mostly used by clocked state changes.

Power-aware drivers only enable their clocks when the device they manage is in active use. Also, system sleep states often differ according to which clock domains are active: while a "standby" state may allow wakeup from several active domains, a "mem" (suspend-to-RAM) state may require a more wholesale shutdown of clocks derived from higher speed PLLs and oscillators, limiting the number of possible wakeup event sources. A driver's suspend method may need to be aware of system-specific clock constraints on the target sleep state.

Some platforms support programmable clock generators. These can be used by external chips of various kinds, such as other CPUs, multimedia codecs, and devices with strict requirements for interface clocking.

## clk\_get

### LINUX

Kernel Hackers Manual June 2013

### Name

`clk_get` — lookup and obtain a reference to a clock producer.

## Synopsis

```
struct clk * clk_get (struct device * dev, const char * id);
```

## Arguments

*dev*

device for clock “consumer”

*id*

clock consumer ID

## Description

Returns a struct clk corresponding to the clock producer, or valid IS\_ERR condition containing errno. The implementation uses *dev* and *id* to determine the clock consumer, and thereby the clock producer. (IOW, *id* may be identical strings, but clk\_get may return different clock producers depending on *dev*.)

Drivers must assume that the clock source is not enabled.

clk\_get should not be called from within interrupt context.

## clk\_enable

### LINUX

Kernel Hackers Manual June 2013

## Name

clk\_enable — inform the system when the clock source should be running.

## Synopsis

```
int clk_enable (struct clk * clk);
```

## Arguments

*clk*

clock source

## Description

If the clock can not be enabled/disabled, this should return success.

Returns success (0) or negative errno.

## clk\_disable

### LINUX

Kernel Hackers Manual June 2013

## Name

`clk_disable` — inform the system when the clock source is no longer required.

## Synopsis

```
void clk_disable (struct clk * clk);
```

## Arguments

*clk*

clock source

## Description

Inform the system that a clock source is no longer required by a driver and may be shut down.

## Implementation detail

if the clock source is shared between multiple drivers, `clk_enable` calls must be balanced by the same number of `clk_disable` calls for the clock source to be disabled.

## clk\_get\_rate

### LINUX

Kernel Hackers Manual June 2013

## Name

`clk_get_rate` — obtain the current clock rate (in Hz) for a clock source. This is only valid once the clock source has been enabled.

## Synopsis

```
unsigned long clk_get_rate (struct clk * clk);
```

## Arguments

*clk*

clock source

## clk\_put

### LINUX

Kernel Hackers Manual June 2013

## Name

`clk_put` — "free" the clock source

## Synopsis

```
void clk_put (struct clk * clk);
```

## Arguments

*clk*

clock source

## Note

drivers must ensure that all `clk_enable` calls made on this clock source are balanced by `clk_disable` calls prior to calling this function.

`clk_put` should not be called from within interrupt context.

# clk\_round\_rate

## LINUX

Kernel Hackers Manual June 2013

### Name

`clk_round_rate` — adjust a rate to the exact rate a clock can provide

### Synopsis

```
long clk_round_rate (struct clk * clk, unsigned long rate);
```

### Arguments

*clk*

clock source

*rate*

desired clock rate in Hz

### Description

Returns rounded clock rate in Hz, or negative errno.

# clk\_set\_rate

## LINUX



## Name

`clk_set_rate` — set the clock rate for a clock source

## Synopsis

```
int clk_set_rate (struct clk * clk, unsigned long rate);
```

## Arguments

*clk*

clock source

*rate*

desired clock rate in Hz

## Description

Returns success (0) or negative errno.

# clk\_set\_parent

## LINUX

## Name

`clk_set_parent` — set the parent clock source for this clock

## Synopsis

```
int clk_set_parent (struct clk * clk, struct clk * parent);
```

## Arguments

*clk*

clock source

*parent*

parent clock source

## Description

Returns success (0) or negative errno.

## clk\_get\_parent

### LINUX

Kernel Hackers Manual June 2013

## Name

`clk_get_parent` — get the parent clock source for this clock

## Synopsis

```
struct clk * clk_get_parent (struct clk * clk);
```

## Arguments

*clk*

clock source

## Description

Returns struct `clk` corresponding to parent clock source, or valid `IS_ERR` condition containing `errno`.

# clk\_get\_sys

## LINUX

Kernel Hackers Manual June 2013

## Name

`clk_get_sys` — get a clock based upon the device name

## Synopsis

```
struct clk * clk_get_sys (const char * dev_id, const char *
con_id);
```

## Arguments

*dev\_id*

device name

*con\_id*

connection ID

## Description

Returns a struct `clk` corresponding to the clock producer, or valid `IS_ERR` condition containing `errno`. The implementation uses `dev_id` and `con_id` to determine the clock consumer, and thereby the clock producer. In contrast to `clk_get` this function takes the device name instead of the device itself for identification.

Drivers must assume that the clock source is not enabled.

`clk_get_sys` should not be called from within interrupt context.

## clk\_add\_alias

### LINUX

Kernel Hackers Manual June 2013

## Name

`clk_add_alias` — add a new clock alias

## Synopsis

```
int clk_add_alias (const char * alias, const char *  
alias_dev_name, char * id, struct device * dev);
```

## Arguments

*alias*

name for clock alias

*alias\_dev\_name*

device name

*id*

platform specific clock name

*dev*

device

## Description

Allows using generic clock names for drivers by adding a new alias. Assumes `clkdev`, see `clkdev.h` for more info.

